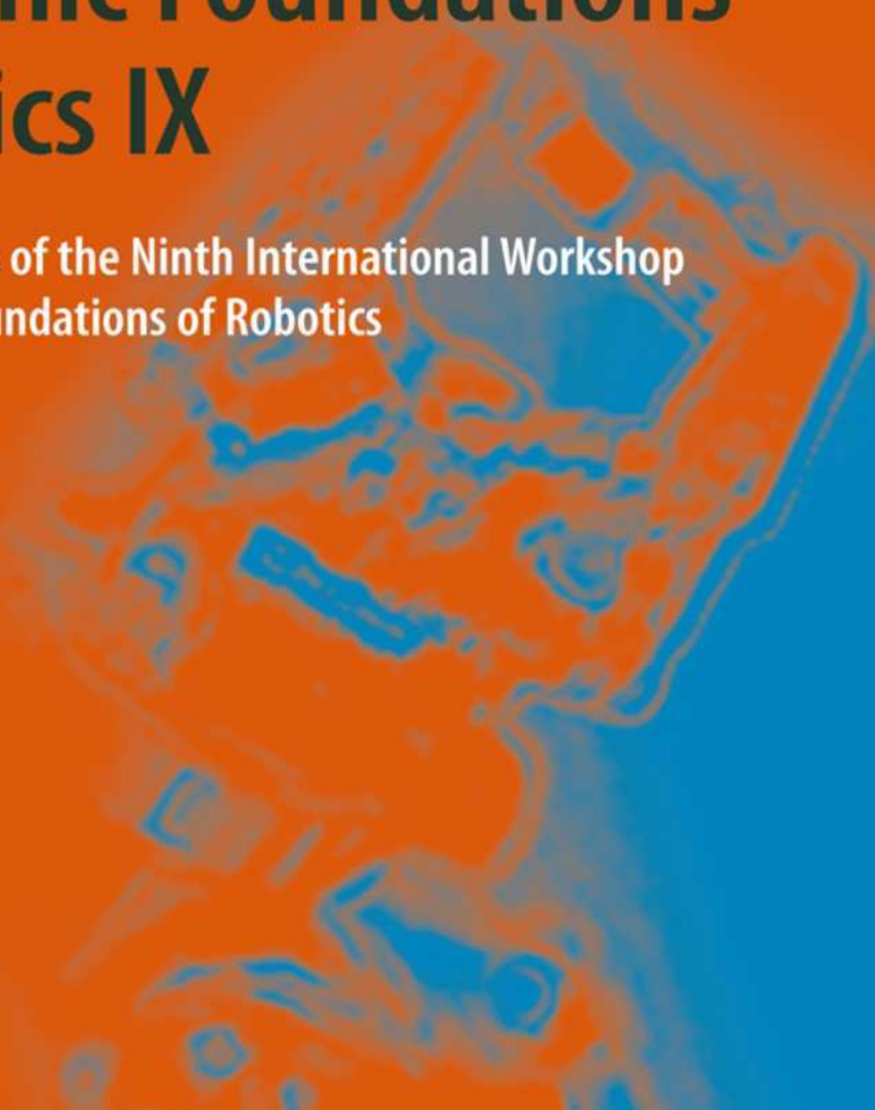
star springer tracts in advanced robotics 68

David Hsu
Volkan Isler
Jean-Claude Latombe
Ming C. Lin (Eds.)

Algorithmic Foundations of Robotics IX

Selected Contributions of the Ninth International Workshop on the Algorithmic Foundations of Robotics



Springer

Springer Tracts in Advanced Robotics

Volume 68

Editors: Bruno Siciliano · Oussama Khatib · Frans Groen

David Hsu, Volkan Isler, Jean-Claude Latombe,
Ming C. Lin (Eds.)

Algorithmic Foundations of Robotics IX

Selected Contributions of the Ninth International Workshop on the Algorithmic Foundations of Robotics

Springer

Professor Bruno Siciliano, Dipartimento di Informatica e Sistemistica, Università di Napoli Federico II, Via Claudio 21, 80125 Napoli, Italy, E-mail: siciliano@unina.it

Professor Oussama Khatib, Artificial Intelligence Laboratory, Department of Computer Science, Stanford University, Stanford, CA 94305-9010, USA, E-mail: khatib@cs.stanford.edu

Professor Frans Groen, Department of Computer Science, Universiteit van Amsterdam, Kruislaan 403, 1098 SJ Amsterdam, The Netherlands, E-mail: groen@science.uva.nl

Editors

Dr. David Hsu
National University of Singapore
School of Computing
13 Computing Drive
Singapore 117417
dyhsu@comp.nus.edu.sg
http://www.comp.nus.edu.sg/∼dyhsu

Dr. Volkan Isler
University of Minnesota
Department of Computer Science
200 Union Street SE
Minneapolis, MN 55455, USA
isler@cs.umn.edu
http://www-users.cs.umn.edu/∼isler/

Dr. Jean-Claude Latombe
Stanford University
Computer Science Department
Stanford, CA 94305-9010, USA
latombe@cs.stanford.edu
http://robotics.stanford.edu/∼latombe/

Dr. Ming C. Lin
University of North Carolina, Chapel Hill
Department of Computer Science
254 Brooks Building
Chapel Hill, NC 27599, USA
lin@cs.unc.ed
http://www.cs.unc.edu/∼lin/

ISBN 978-3-642-17451-3 e-ISBN 978-3-642-17452-0

DOI 10.1007/978-3-642-17452-0

Springer Tracts in Advanced Robotics ISSN 1610-7438

Typeset & Cover Design: Scientific Publishing Services Pvt. Ltd., Chennai, India.

Printed on acid-free paper

5 4 3 2 1 0

springer.com

Foreword

By the dawn of the new millennium, robotics has undergone a major transformation in scope and dimensions. This expansion has been brought about by the maturity of the field and the advances in its related technologies. From a largely dominant industrial focus, robotics has been rapidly expanding into the challenges of the human world (human-centered and life-like robotics). The new generation of robots is expected to safely and dependably interact and work with humans in homes, workplaces, and communities providing support in services, entertainment, education, exploration, healthcare, manufacturing, and assistance.

Beyond its impact on physical robots, the body of knowledge that robotics has produced is revealing a much wider range of applications reaching across diverse research areas and scientific disciplines, such as: biomechanics, haptics, neurosciences, and virtual simulation, animation, surgery, and sensor networks among others. In return, the challenges of the new emerging areas are proving an abundant source of stimulation and insights for the field of robotics. It is indeed at the intersection of disciplines where the most striking advances happen.

The Springer Tracts in Advanced Robotics (STAR) is devoted to bringing to the research community the latest advances in the robotics field on the basis of their significance and quality. Through a wide and timely dissemination of critical research developments in robotics, our objective with this series is to promote more exchanges and collaborations among the researchers in the community and contribute to further advancements in this rapidly growing field.

Since its inception in 1994, the biennial *Workshop Algorithmic Foundations of Robotics (WAFR)* has established some of the field's most fundamental and lasting contributions. Since the launching of STAR, WAFR and several other thematic symposia in robotics find an important platform for closer links and extended reach within the robotics community.

This volume is the outcome of the WAFR ninth edition and is edited by D. Hsu, V. Isler, J.-C. Latombe and M.C. Lin. The book offers a collection of a wide range of topics in advanced robotics, including motion planning, multiagents, modular and reconfigurable robots, localization and mapping, grasping, and sensing.

The contents of the twenty-four contributions represent a cross-section of the current state of research from one particular aspect: algorithms, and how they are inspired by classical disciplines, such as discrete and computational geometry, differential geometry, mechanics, optimization, operations research, computer science, probability and statistics, and information theory. Validation of algorithms, design concepts, or techniques is the common thread running through this focused collection.

Rich by topics and authoritative contributors, WAFR culminates with this unique reference on the current developments and new directions in the field of algorithmic foundations. A very fine addition to the series!

Naples, Italy
October 2010

Bruno Siciliano
STAR Editor

Preface

Robot algorithms are a fundamental build block of robotic systems. They enable robots to perceive, plan, control, and learn, in order to achieve greater autonomy. Today, the design and analysis of robot algorithms are more crucial than ever for at least two reasons:

- Robotics is undergoing major transformation. Originally focused on industrial manufacturing, it is now rapidly expanding into new domains, such as homes and offices, elderly care, medical surgery, entertainment, ocean and space exploration, and search-and-rescue missions. In these new domains, tasks are less repetitive, environments are less structured, events are more unpredictable, and greater autonomy is required over long periods of time. It is often impossible to anticipate all events explicitly and to program the robots specifically to handle them. New algorithms that are adaptive to environment uncertainties and changes are needed to conquer these challenges.
- Robot algorithms are finding new applications beyond robotics, for example, in designing mechanical assemblies, modeling molecular motion, creating digital characters for video games and computer-generated movies, architectural simulation, and ergonomic studies. These non-traditional applications of robot algorithms pose new challenges: hundreds or thousands of degrees of freedom, large crowds of characters, complex physical constraints, and natural-looking motions. The resulting new algorithms may in turn benefit future robots.

Robot algorithms are also rapidly evolving as a result of new technologies, e.g., low-cost parallel computers, cheaper and more diverse sensors, and new interaction technologies ranging from haptic to neuroprosthetic devices.

Unlike traditional computer algorithms, robot algorithms interact with the physical world. They must operate safely, reliably, and efficiently under tight time constraints in imperfectly known environments. So, it is not surprising that the design and analysis of robot algorithms raise unique combinations of fundamental questions in computer science, electrical engineering, mechanical engineering, and mathematics. For example, minimalist robotics studies the minimal sensing and actuation capabilities required for robots to complete a given task. It addresses not only computational complexity issues, but also "physical" complexity issues. Probabilistic methods are widely used as a modeling tool to handle uncertainties due to sensing and actuation noise, but they are also used as a computational tool that avoids costly computations by extracting partial information and handling the resulting uncertainties. Many other such examples abound.

The Workshop on Algorithmic Foundations of Robotics (WAFR) is a highly selective single-track meeting of leading researchers in the field of robot algorithms. Since its creation in 1994, WAFR has been held every two year and has published some of the field's most important and lasting contributions.

The ninth WAFR was held on December 13-15, 2010 at the National University of Singapore. It had a strong program of 24 contributed papers selected from 62 submissions. Each paper was rigorously reviewed by at least three reviewers with additional input from two program committee members. The workshop also featured 6 invited speakers: Leonidas Guibas (Stanford University), Leslie Kaelbling (Massachusetts Institute of Technology), Jean-Pierre Merlet (INRIA Sophia-Antipolis), Jose del Millan (École Polytechnique Fédérale de Lausanne), Yoshihiko Nakamura (University of Tokyo), Moshe Shoham (Technion – Israel Institute of Technology). A vibrant poster and video session was a new addition to this WAFR program to encourage open exchange of ideas in an informal atmosphere.

In addition to the editors of volume, the program committee consists of Srinivas Akella (University of North Carolina at Charlotte), Dan Halperin (Tel Aviv University), Seth Hutchinson (University of Illinois at Urbana-Champaign), Vijay Kumar (University of Pennsylvania), Jean-Paul Laumond (LAAS-CNRS), Stephane Redon (INRIA Grenoble - Rhone-Alpes), Daniela Rus (Massachusetts Institute of Technology), Katsu Yamane (Disney Research and Carnegie Mellon University).

It was a real pleasure to organize the workshop and to work with such a talented group of people. We owe many thanks to all the authors for submitting their exciting work, to the program committee members and reviewers for their dedication to ensure the finest quality of WAFR, to the speakers for inspiring thoughts and ideas, and to all the participants for making the workshop a great success. We also wish to thank the following organizations for their generous financial support of this WAFR:

- National University of Singapore, School of Computing
- United States Air Force Office of Scientific Research, Asian Office of Scientific Research & Development.

Finally we wish to thank the School of Computing at National University of Singapore for providing the logistic and technical support necessary to make this WAFR successful.

David Hsu
Volkan Isler
Jean-Claude Latombe
Ming C. Lin

Contents

Homotopic Path Planning on Manifolds for Cabled Mobile Robots

Takeo Igarashi and Mike Stilman

Abstract. We present two path planning algorithms for mobile robots that are connected by cable to a fixed base. Our algorithms efficiently compute the shortest path and control strategy that lead the robot to the target location considering cable length and obstacle interactions. First, we focus on cable-obstacle collisions. We introduce and formally analyze algorithms that build and search an overlapped configuration space manifold. Next, we present an extension that considers cable-robot collisions. All algorithms are experimentally validated using a real robot.

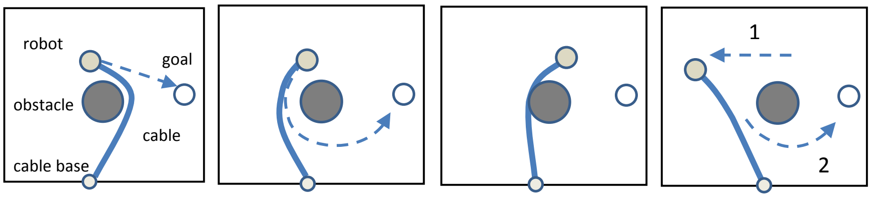
1 Introduction

Mobile robots are typically untethered. This is not always desirable in household and high-power robotics. Wireless communication can be unreliable and batteries need to be charged regularly. These challenges can be solved by using cables for communication and power. Currently, cables are a viable option for robots that work in fixed environments such as homes and offices. The challenge addressed in this paper is that cables impose additional constraints on robot motion. First, robots cannot go further than the cable length. Second, they are blocked by the cable itself when the robots are not capable of crossing it. We present two practical planning algorithms that handle these constraints and validate them on a real robot system.

The first challenge is that a cabled robot's movement is constrained by the length of the cable. If there is no obstacle, the robot's motion is limited to stay within a circle around the fixed end-point of the cable. If there is an obstacle in the environment, the robot's movement is further constrained by the interaction between the cable and

Takeo Igarashi
Department of Computer Science, The University of Tokyo
JST ERATO Igarashi Design Interface Project
e-mail: takeo@acm.org

Mike Stilman
Center for Robotics and Intelligent Machines, Georgia Institute of Technology
e-mail: mstilman@cc.gatech.edu



(a) Direct Path (b) Detour Path (c) Blocked Detour (d) Solution to (c)

Fig. 1 The problem domain and a challenging example where the robot must untangle its cable.

the obstacle as shown in Fig. 1(a) and (b). The locations of the robots are the same, but the shortest paths to the goal are different because the cable configuration in Fig. 1(b) cannot stretch to the goal. The second problem is that the robot's movement may be blocked by the cable when the robot is not capable of crossing it as shown in Fig. 1(c). The robot must make an auxiliary motion to move the blocking cable out of the way (Fig. 1(d)). This is difficult because the robot cannot directly control the cable. It must indirectly control it by pulling.

In order to address these challenges, Section 3 introduces the *overlapped manifold* representation for the configuration space of cabled robots. We develop an efficient, resolution complete and optimal algorithm that constructs the manifold and solves practical planning problems. To handle collisions between the cable and the robot, Section 4 presents a second search method that applies physics-based simulation combined with heuristics to choose intermediate subgoals that maximize robot mobility. Section 5 experimentally demonstrates that both algorithms generate appropriate paths for a real robot that reaches targets in the presence of a cable.

2 Related Work

The topic presented in this paper is far more complex than general path planning [1, 2]. While the robot itself only operates in two dimensions, the cable is also part of the complete system or plant. By including the cable, the challenge is lifted to planning for an infinite-dimensional underactuated system. Previous work on tethered robots [3] treated the problem as multi-robot scheduling. Our approach focuses on a single agent and handles environment geometry.

Considerable research on high degree-of-freedom (DOF) robot systems such as [4, 5] has direct applications to domains with dozens of DOF and non-holonomic constraints. Our problem, however, requires handling even higher DOF and underactuation. Hence, the challenge is also distinct from deformable motion planning as presented in [6, 7]. Likewise, cable-routing [8] assumes that shape of the cable is directly controllable. However, a cabled robot cannot control all of its degrees of freedom and must rely on predictions of cable motion due to stretch.

Existing planning methods for underactuated deformable objects typically focus on local deformations [9, 10]. Studies on deformable needle steering also consider the path to a robot configuration [11, 12] with a focus on local environment deformation and curvature constraints. Our work complements these studies since our task is to determine globally optimal robot paths. Global constraints are imposed by the cable length, wrapping around obstacles and potentially colliding with the robot.

Typically, globally constrained underactuated planning and control is restricted to four DOF systems as shown in [13, 14, 15]. To handle the global problem complexity *our domain requires a different approach based on topological path homotopy.* Existing work in knot-tying [16] plans with distinct topological states. However it explicitly encodes and plans rope overlaps. Other planners that distinguish homotopic paths, [17, 18, 19], typically operate in a standard high-DOF configuration space. Instead, we build a configuration space manifold that implicitly encodes the homotopy of cable configuration and then search for shortest paths on the manifold.

In direct homotopic planning, [20] studies shortest paths but restricts the domain to a boundary-triangulated space. [21] requires semi-algebraic models of obstacles. [22] gives a configuration space representation that closely related to our work. Their complex-plane mapping of paths may increase the efficiency of our methods for single-query search. In contrast to our proposed manifold, existing techniques do not address global cable-length constraints or cable-robot interactions.

Existing methods for manifold construction tend to focus on relationships in recorded data [23, 24]. We present a novel, simple algorithm for global path planning with distance constraints on paths. The algorithm not only generates paths, but a complete vector field [25, 26] for robot motion on a manifold of homotopic paths. Our extensions to this algorithm also consider cable dynamics [27, 28] and evaluate strategies for robot motion when the cable itself is an obstacle in the space [29].

3 Distance Manifolds: Cable-Obstacle Interaction

We present a path planner for cabled mobile robots. The initial configuration, q_i, includes initial cable displacement. The goal is any configuration q_g that places the robot at p_g in a 2D environment. The robot is connected to a fixed base location, p_0, by a cable resting on the floor. The cable is a flexible, passive entity whose shape is determined solely by the previous motions of the robot. The environment contains fixed obstacles that restrict both cable and robot motion. For simplicity, we assume a disk-shaped robot with a given diameter and a cable attached by a freely rotating joint. Furthermore, we represent space by a grid where configuration space obstacles must occupy at least one grid vertex. This section introduces an algorithm that handles the constraint given by cable length.

First, we build the configuration space that represents the structure of the problem and then compute a vector field that guides the robot in the configuration space. Given a static environment the configuration space is generated off-line, reducing the cost of online planning. Sections 3.1,2 describe the space, its graph representation and formalize the problem statement. Sections 3.3,4 introduce the algorithms for graph construction and planning. Section 3.5 proves algorithm correctness.

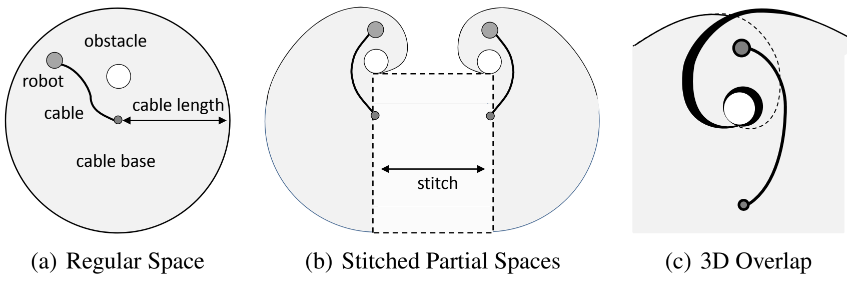


Fig. 2 Simple and overlapped manifold configuration space for tethered robots.

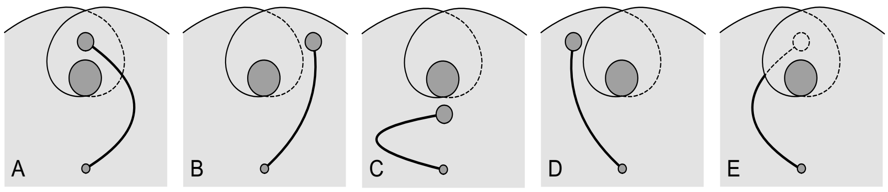


Fig. 3 Example traversal of configuration space.

3.1 Configuration Space

In order to build a complete planner for tethered robots, we consider the configuration space. Notice that the space must distinguish distinct homotopic paths. Some configurations that have identical robot locations have different cable configurations. If we ignore collisions between the cable and obstacles, the configuration space is a 2D circular region defined by the 2D environment (Fig. 2a). However, this representation cannot distinguish configurations with different cable positions (Fig. 3 A,E). The cable location determines the region of space that is immediately reachable by the robot. In order to differentiate between A and E, we define configuration space by an *overlapped manifold*. The manifold is planar, but it can be visualized with stitched or overlapped free space components (Fig. 2b,c).

Distinct configurations on the manifold with the same locations represent distinct cable configurations. A continuous region in the manifold corresponds to a set of configurations that can be reached by continuous robot motion. In Fig. 3, straight trajectories change the configuration from A to E via B, C, D. However, there is no straight path that can displace the robot from A to D or B to E.

Notice that the number of overlaps in the manifold increases exponentially with each additional obstacle. The robot has two options for circumnavigating each obstacle. It can go around to the left or to the right. Hence, for n reachable obstacles, there exist at least 2^n paths or cable routes that reach the same goal. This corresponds to at least 2^n possible overlaps in the configuration space manifold. We say "at least" because winding the cable around an obstacle also doubles the overlaps.

3.2 Graph Representation of Configuration Space

This section gives a formal representation of the problem domain and the problem statement. Our algorithm constructs a graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ that represents the configuration space manifold (Section 3.3) and then searches the graph (Section 3.4) for vector fields or paths. The key challenge is to construct a graph that completely encodes the configuration space. This section defines the graph properties that must be assured in graph construction. Section 3.5 verifies these properties.

First, consider the domain definitions: Manifold *vertices* are located at physical grid nodes. v_0 is the vertex that represents the cable base. D_{ij} is the manhattan distance on the grid between v_i and v_j. When $D_{ij} = 1$, the two vertices are referred to as *neighbors*. Notice that neighboring vertices are not necessarily connected by an edge since they may be on distinct overlapping folds of the manifold. P_{ij} represents paths between vertices on the manifold and $|P_{ij}|$ is path length.
The problem is to build a graph $\mathbf{G}$ *such that any shortest path from* v_i *to* v_g *corresponds to a shortest path from* q_i *to any* q_g *where* p_g *is the target and the cable does not cross an obstacle.*

Definition 1. Two paths from v_0 to any point are path *homotopic* if and only if there exists no obstacle in the area enclosed by the two paths. Otherwise they are not homotopic or *ahomotopic*.

While grid nodes are simply positions, p_i, manifold vertices, v_i are defined by the set of homotopic paths from v_0 to p_i. Each vertex is associated with a path of adjacent vertices of length less than D_{max}. In every set of homotopic paths, there exist minimal paths. Let us call them *m-paths* ($mP(v_0, v_i)$ or mP_i).

Definition 2. v_2 is an *m-child* of v_1 ($v_2 \succ v_1$) and v_1 is an *m-parent* of v_2 ($v_1 \prec v_2$) if and only if there exists a minimal path, mP_2, where v_1 is the last vertex before v_2.

Definition 3. v_1, v_2 are *m-adjacent* ($v_1 \sim v_2$) if and only if v_1 is an m-parent of v_2 or v_2 is an m-parent of v_1.

Definition 4. Collocated vertices v_1 and v_2 are manifold-equivalent, *m-equivalent* ($v_1 \equiv v_2$) if and only if every path to v_1 is homotopic to every path to v_2.

All m-equivalent vertices are collocated, $v_1 \equiv v_2 \Rightarrow v_1 \simeq v2$. However, not all collocated vertices are m-equivalent. This occurs when the paths to v_1 go around some obstacle while those to v_2 do not. In this case, the distance between vertices on the manifold can be greater than physical distance between their positions.

Definition 5. The *m-distance* between v_1 and v_2, mD_{12} is the length of any shortest path between v_1 and v_2 such that all consecutive nodes on the path are m-adjacent.

Lemma 1. *Let* v_1 *and* v_2 *be neighboring vertices such that* $D_{12} = 1$. *For any vertex,* v_3, *if* $D_{13} \geq D_{23}$ *then strictly* $D_{13} > D_{23}$. *Likewise,* $mD_{13} \geq mD_{23} \Rightarrow mD_{13} > mD_{23}$.

Proof. Any paths P_{13} and P_{23} must have distinct parity since they are separated by one edge [30]. One path has even length while the other is odd. Hence $|P_{13}| \neq |P_{23}|$. □

Proposition 1 (Adjacency). *For neighboring vertices: $v_1 \sim v_2$ (a) if and (b) only if there is no obstacle in the area enclosed by any mP_1 and mP_2.*

Proof. Since v_1 and v_2 are neighbors, $mD_{01} \neq mD_{02}$ by Lemma 1. Without loss of generality, consider $mD_{01} < mD_{02}$. (a) For any path mP_1, construct path $P_2 = \{mP_1, v_2\}$. This is a minimal path to v_2 with v_1 as the last vertex since $mD_{01} < mD_{02}$ and $|P_2| = |mP_1| + 1$. Hence, $v_1 \prec v_2$ and $v_1 \sim v_2$. (b) By contradiction: assume there exists an obstacle enclosed by some mP_1, mP_2. By the premise, $v_1 \sim v_2$ and therefore $v_1 \prec v_2$. Hence by Def. 2, there exists mP_2' with v_1 as the last vertex and mP_1 as a sub-path. mP_2' and mP_2 enclose an obstacle so they are not homotopic. Thus $v_2 \not\equiv v_2$. Contradiction. Likewise if $mD_{01} > mD_{02}$. □

Consider again the problem statement: *Build a graph* **G** *such that any shortest path from v_i to v_g corresponds to a shortest path from q_i to any q_g s.t. the cable does not cross an obstacle.* Following Proposition 1, this graph must have the following property: *two neighboring vertices are connected by an edge if and only if they are m-adjacent.* Section 3.3 introduces our algorithm for constructing **G**.

3.3 Manifold Construction: Forward Search

Our algorithm in Fig. 4 incrementally adds vertices and builds graph edges by connecting adjacent vertices to the north, south, east and west of each vertex. SUCCESSORS$(v_a, \mathbf{V}, \mathbf{E})$ returns the set of neighboring, collision-free vertices that are not yet in the graph. Since we assume that obstacles occupy at least one grid node, COLLISIONFREE(p) returns *true* when a node does not intersect an obstacle.

Multiple manifold vertices can share a single grid position as in Fig. 3 (A,E). Our algorithm, distinguishes grid positions, p_i, from manifold vertices, v_i. The position of vertex v_i is obtained by POS$[v_i]$. The function NEIGHBORS(v_i) returns the set of four neighboring positions of the vertex. The function ADJACENT(v_i) returns the set of all vertices in **V** that are adjacent/connected to v_i in **G** as follows: $\{v_j | \exists e(v_i, v_j) \in \mathbf{E}\}$. There are at most four such vertices. Likewise, ADJACENT(ADJACENT(v_i)) returns at most eight vertices $\neq v_i$ that are adjacent to the first four.

BUILDMANIFOLD is a variant of breadth-first search or wavefront expansion. Standard expansion adds all edges to existing neighbors when adding a new vertex. In contrast, we add an edge to a neighbor only when there is a common vertex that has edges to both the neighbor and the parent of the new vertex (Lines 10-13). This is illustrated by Fig. 5. An edge is added between v_t and v_b since both v_a and v_b have edges to a common vertex v_c. However, an edge is not added between v_t and v_d. Likewise, in Fig. 6, no edge is added between v_t and v_d, generating the manifold with overlaps as presented in Section 3.1.

3.4 Plan Generation: Backward Search

Given the graph representing the configuration space, we construct a vector field to guide the robot from any given starting location to any desired goal. This is required

```
BuildManifold($v_0$, V, E)
 1  Q ← Enqueue($v_0$)
 2  while Q ≠ ∅
 3  do $v_a$ ← Dequeue(Q)
 4      S ← Successors($v_a$, V, E)
 5      for all $v_t \in$ S
 6      do V ← Insert($v_t$)
 7          E ← Insert($v_t, v_a$)
 8          if Dist[$v_t$] < DistMax
 9             then Q ← Enqueue($v_t$)
10          B ← Adjacent(Adjacent($v_a$))
11          for all $v_b \in$ B
12          do if Pos[$v_b$] ∈ Neighbors($v_t$)
13                then E ← Insert($v_t, v_b$)
```

```
Successors($v_a$, V, E)
 1  N ← Neighbors($v_a$)
 2  S ← ∅
 3  for all $p_i \in$ N
 4  do if CollisionFree($p_i$) and
 5        $p_i \notin$ Pos[Adjacent($v_a$)]
 6        then $v_i$ ← NewVertex
 7             Pos[$v_i$] ← $p_i$
 8             Dist[$v_i$] ← Dist[$v_a$] + 1
 9             S ← Insert($v_i$)
10   return S
```

Fig. 4 Manifold Construction Pseudo-code.

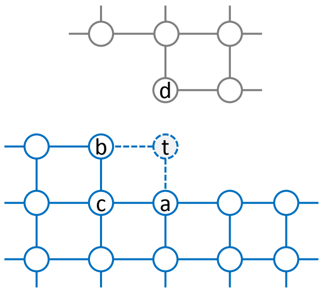


Fig. 5 BuildManifold Line 11

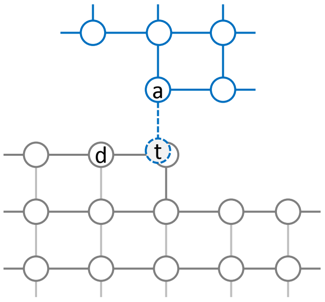


Fig. 6 Illustration of overlap

since the manifold is a roadmap that is created for all possible start and goal states. Basic dynamic programming or wavefront expansion is used compute a distance field over the configuration space starting from the target location. The gradient of the distance field is used to control the robot. Note that the target location can be associated with multiple vertices in the graph. Starting from all these vertices, we assign minimal distance values to the remaining vertices by breadth-first traversal. Consequently, the robot always follows the minimal path on the manifold.

3.5 Algorithm Analysis

This section analyzes the complexity, optimality and correctness of our algorithms. First of all, the computational complexity of manifold construction is $O(n)$ where n is the number of vertices in the configuration space. Likewise, the computational complexity of search is $O(n)$. Hence the entire algorithm is executed in $O(n)$. Furthermore, *manifold generation must only be computed once for static environments*, regardless of start state and goal. This yields efficient multi-query planning.

Given that **G** is correctly constructed and completely represents the manifold that the robot can traverse then dynamic programming is a complete and optimal method for finding a solution. Therefore plan generation is complete and optimal. The remaining task is to prove the correctness and completeness of manifold construction. We will use Proposition 2 in the validation of BUILDMANIFOLD in Proposition 3.

Proposition 2 (Equivalence). *If $v_1 \simeq v_2$ are both m-adjacent to v_a then $v_1 \equiv v_2$.*

Proof. Let P_1 and P_2 be any paths to v_1 and v_2. There exist shortest paths mP_1, mP_2 homotopic to P_1, P_2 respectively. Let mP_a be a shortest path to v_a. Since $v_1 \sim v_a$ and $v_2 \sim v_a$, by Prop. 1, there is no obstacle enclosed by $\{mP_1, mP_a\}$ and $\{mP_2, mP_a\}$. Hence, there is no obstacle enclosed by mP_1 and mP_2, so they are homotopic. Since $v_1 \simeq v_2$ and all paths to v_1 are homotopic to all paths to v_2 we have $v_1 \equiv v_2$. □

Proposition 3. *Prior to adding v_3 with $mD_{03} \geq D < D_{max}$,* BUILDMANIFOLD *maintains the following invariant. Let vertices v_1 and v_2 have $mD_{01} < D$, $mD_{02} < D$.*
(a) $v_1 \in \mathbf{V}$ if and only if v_1 is not m-equivalent to any other vertex, $v_2 \in \mathbf{V}$.
(b) $e(v_1, v_2) \in \mathbf{E}$ if and only if v_1 is m-adjacent to v_2 ($v_1 \sim v_2$).

Proof. We proceed by induction. Base case, $D = 1$, there are no edges and the only vertex is v_0, added in Line 1. The inductive step is split into the following Lemmas.

For Lemmas 2-5 assume Prop. 3. *Prior to adding any v_3 such that $mD_{03} \geq D + 1$:* (BMY and SY refer to Line Y in BUILDMANIFOLD and SUCCESSORS respectively)

Lemma 2. *If $v_1 \in \mathbf{V}$ then v_1 is not m-equivalent to any other vertex, $v_2 \in \mathbf{V}$.*

Proof. BUILDMANIFOLD adds v_1 to $\mathbf{V}$ by expanding v_a only if v_a has no edge to any vertex at its position, p_1 (S5). By assumption, v_a is not m-adjacent to any vertex at p_1. Hence, v_1 is the only vertex at p_1 such that $v_a \prec v_1$. Therefore it is the only vertex with a minimal path mP_1 such that v_a is the last vertex. □

Lemma 3. *If $\exists v_1$ ($mD_{01} \leq D$) not m-equivalent to any other vertex in $\mathbf{V}$ then $v_1 \in \mathbf{V}$.*

Proof. By contradiction: Suppose $v_1 \notin \mathbf{V}$. v_1 is an m-child of some v_a where $mD_{0a} = mD_{01} - 1$. By the assumption, $mD_{0a} < D$ so $v_a \in V$ and by BM9, $v_a \in Q$. Since there are finite vertices with $mD_{0a} < D$, v_a is dequeued and expanded in BM4. Since v_1 is a neighbor of v_a one of the following must hold: (1) By S4, v_1's position is not collision free. Contradiction. (2) By S5 and the inductive assumption, v_a has an m-adjacent vertex, $v_2 \simeq v_1$. By Prop. 2, $v_2 \equiv v_1$. Contradiction. □

Lemma 4. *If $\mathbf{E}$ contains edge (v_1, v_2) then v_1, v_2 are m-adjacent.*

Proof. Without loss of generality, suppose v_1 is added after v_2. An edge is added at (a)BM7 or (b)BM13. (a) $v_2 = v_a$ and v_1 is newly defined and implicitly associated with minimal paths homotopic to $mP_1 = \{mP_a, v_1\}$. Since v_2 is the last vertex on mP_1, $v_2 \prec v_1$. (b) $v_2 = v_b$. By BM7 there exists $v_a \prec v_1$. By BM10 and the inductive assumption there exists v_c such that $v_a \sim v_c \sim v_b$. Given that $v_b \sim v_c$ and $v_c \sim v_a$ and $v_a \sim v_1$, Prop. 1 states that there is no obstacle between any mP_b and mP_1 as shown by regions $(bc), (ca), (a1)$ and R in Fig. 7(a). Hence, by Prop. 1 $v_2 = v_b \sim v_1$. □

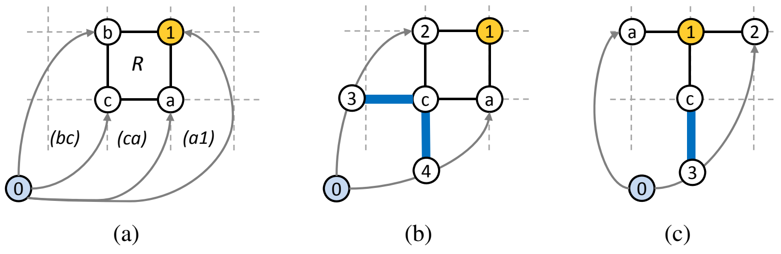


Fig. 7 Illustrations for the proof of Lemmas 4,5. Straight lines indicate precise grid displacements. Curves represent paths that preserve relative position but not necessarily distance.

Lemma 5. *If* $v_1, v_2 (mD_{01}, mD_{02} \leq D)$ *are m-adjacent,* $\mathbf{E}$ *contains edge* (v_1, v_2).

Proof. By the inductive assumption: $v_1, v_2 \in \mathbf{V}$. Without loss of generality, $v_2 \prec v_1$.

(a) $mD_{01}, mD_{02} < D$ then $(v_1, v_2) \in \mathbf{E}$ by the inductive assumption.

For the remaining cases $mD_{01} = D$ and $mD_{02} = D - 1$ by Lemma 1.

(b) $v_2 = v_a$ is the first m-parent of v_1 added to $\mathbf{V}$. Then $(v_1, v_2) \in \mathbf{E}$ by BM7.

For the remaining cases there exists $v_a \prec v_1$ $(v_a \neq v_2)$ that was added prior to v_2. Since $v_1 \equiv v_1$, there are two minimal paths $mP_1(a) = \{mP_a, v_1\}$ and $mP_1(2) = \{mP_2, v_1\}$ that enclose a region $\mathbf{R}$ with no obstacles. By Prop. 2 there are three relative positions for $v_2 \not\equiv v_a$. Due to symmetry of Fig. 7(b), that yields two cases.

(c) In the case of Fig. 7(b) there exists v_c, neighbor of v_a and v_2 contained in $\mathbf{R}$. Extend two straight paths P_{3c} and P_{4c} opposite v_a and v_2 respectively. Since $\mathbf{R}$ is closed, these paths must intersect mP_2 and mP_a at some vertices v_3, v_4 respectively. Since P_{3c} is straight, $|P_{3c}| < |P_{32}|$. Hence, the path $S_2 = \{P_{03}, v_c, v_2\}$ has length $|S_2| \leq |mP_2|$. Therefore $v_c \prec v_2$. Likewise, since P_{4c} is straight, $|P_{4c}| < |P_{4a}|$. Hence, $S_a = \{P_{04}, v_c, v_a\}$ has length $|S_a| \leq |mP_a|$. Therefore $v_c \prec v_2$. Thus there exists v_c such that $v_2 \sim v_c \sim v_a$. This satisfies BM10-13, thus $(v_1, v_2) \in \mathbf{E}$.

(d) In the case of Fig. 7(c) there exists v_c neighbor of v_1 that is contained in $\mathbf{R}$. Extend a straight path P_{3c} from v_3 opposite v_1. Since $\mathbf{R}$ is closed, P_{3c} must intersect either mP_2 or mP_a at some vertex v_3 respectively. Without loss of generality, assume it intersects mP_2. Since P_{3c} is straight, $|P_{3c}| < |P_{32}|$. Hence, the path $S = \{P_{03}, v_c, v_1\}$ has length $|S_2| \leq |mP_2|$. Since S is a path from v_0 to v_1 homotopic to $mP_1(2)$, we have $mD_1 < |mP_2| + 1$. Thus $mP_1(2)$ is not a minimal path. Contradiction. Likewise, if P_{3c} intersects mP_a, we find $mP_1(a)$ is not a minimal path. Contradiction.

In all valid cases where $v_1 \sim v_2$, $\mathbf{E}$ contains the edge (v_1, v_2). □

Lemma 6. *The algorithm terminates when all* $v_i : mD_i < D_{max}$ *are added to* $\mathbf{Q}$.

Proof. Every new vertex, v_i increments $\text{DIST}[v_i]$ by 1 from its parent (S8). For any D there are a finite number of vertices that are not m-equivalent with $mD_i < D$. Since no vertices are added to **Q** with $\text{DIST}[v_i] \geq \text{DISTMAX}$ (BM7) and each step dequeues, BUILDMANIFOLD terminates. □

By Lemmas 2-6, BUILDMANIFOLD is proven to add all the vertices on the manifold to **V** and all the edges between m-adjacent vertices to **E** prior to guaranteed termination. Therefore, the manifold generation algorithm is correct and complete.

3.6 Implementation Details

The presented algorithm computes Manhattan distance between the base and each vertex in the graph. This is a low-order approximation of physical distance. We therefore also allow diagonal moves when computing the distance value, creating an 8-connected lattice and obtaining a better approximation. The experiments in Section 5 demonstrate that it performs well in robot experiments.

In order to include diagonal moves, Line 7 of SUCCESSORS uses d instead of 1, where $d = \sqrt{2}$ for diagonally connected vertices. Furthermore, **Q** in BUILDMANIFOLD is a priority queue rather than a FIFO in order to always select vertices with the minimal distance from v_0. This approach increases computation time to $O(n \log n)$ due to priority queue operations.

4 Cable-Robot Interaction

Section 3 introduced a novel formulation of the configuration space for tethered robots and presented a complete solution to path planning for robots that are restricted by cable length. The proposed configuration space allows us to go further and consider additional constraints on robot motion. In this section, we examine *the case where the robot cannot cross the cable*. Cable-robot collisions present further algorithmic challenges that are not solved by existing methods. We evaluate two solutions and propose a novel algorithm in Section 4.3.

4.1 Preliminary Algorithm

Simple domains such as Fig. 8(a) can be solved by adding the current cable shape as an obstacle to future motion[29]. We implemented an algorithm that incrementally removed vertices from the graph that were within the robot radius of the cable through wavefront expansion. The online controller continuously updated the vector field as the cable shape changed during motion.

In our experiments, the initial path typically remained valid during robot motion because the deformation of the cable occurred behind the robot. When the plan became inaccessible, the system replanned the path. This approach required continuous tracking of cable shape. Since this is difficult in practical environments we used

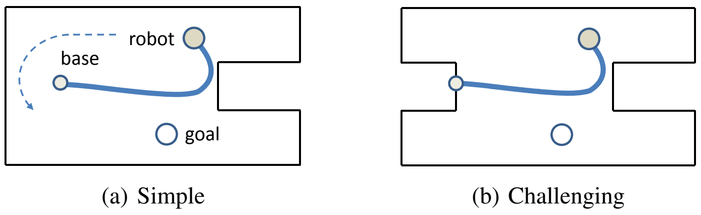


Fig. 8 Illustration of a cable blocking the path to the target.

physical simulation to predict the current shape of the cable based on the motion history of the robot. Section 5 shows that this approach works well in practice.

Notice, however, that *this simple method is not sufficient* when the cable completely blocks a path to the target as shown Fig. 8(b). Removing cable vertices from the configuration space blocks all path to the goal. We present two approaches that handle such cases. First, we consider a hardware solution in which the system retracts the cable. Second, we introduce a novel algorithm for *feasible* path planning that clear blocks through auxiliary robot motion.

4.2 Hardware Solution: Cable Retraction

First of all, the problem in Fig. 8(b) can be solved by continuously retracting the cable to make the cable as short as possible while allowing free robot motion. This approach requires additional hardware, but simplifies planning. Given cable retraction, the robot would simply need to follow the cable to the cable base until the path to the target is cleared. This can be accomplished by searching for a shortest path on the manifold to the base and directing the robot to move along that path.

The hardware implementation is not trivial because one must develop a special device than retracts the cable with appropriate force for the particular robot and cable type. The force must be simultaneously strong enough to pull a long cable and sufficiently weak to allow robot motion. Furthermore, it may be necessary to constantly adjust the force depending on the robot and cable status. We have not yet implemented this solution, however it remains an exciting topic for our future work.

4.3 Algorithmic Solution: Untangling

Given that the robot cannot retract the cable mechanically, it must perform auxiliary motions to clear the path blocked by the cable. We refer to this procedure as *untangling*. Consider Fig. 1(c). The robot must first move to the left to clear the path to the target on the right. More complex domains, such as Fig. 9, have goals that are blocked by the cable multiple times along a single path. The algorithm is required to find a sequence of untangling motions. In contrast to Section 3, evaluating all possible motions was computationally infeasible. Instead, we developed a heuristic method that efficiently computes untangling motions and performs well in practice.

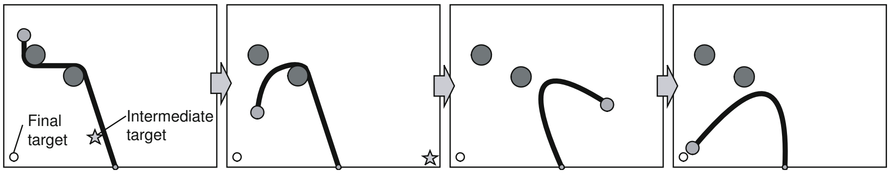


Fig. 9 Auxiliary motions open the path to the target. Each step is computed by our algorithm.

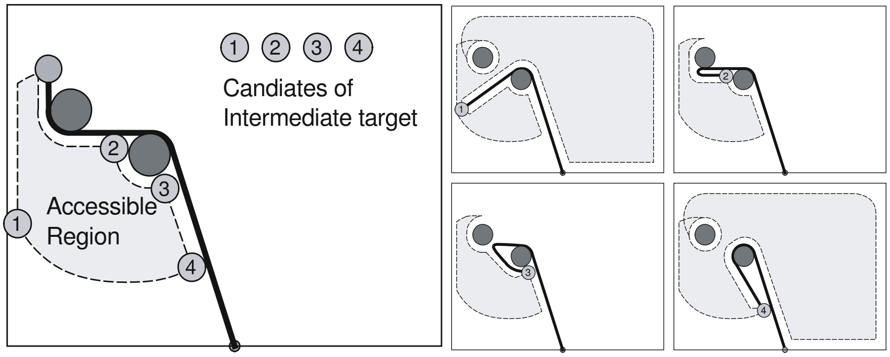


Fig. 10 A search for the most promising candidate. Candidates (left) and their accessible regions (right). Candidate 4 is selected in this case.

When our system identifies that there is no open path to the target from the current robot location, it selects an intermediate target and moves the robot towards it. Motion to the intermediate target is chosen to displace the blocking cable from the path of the robot to the goal (Fig. 9a). If the goal becomes accessible during travel to the target, the online algorithm discards the intermediate target and moves directly to the goal. If the intermediate target becomes inaccessible or if the robot reaches the target, the system computes the next target. This process repeats until the goal becomes accessible. Fig. 9 shows a complete untangling procedure.

For each step in the untangling process, we choose the intermediate target from several candidates. The most promising one is selected by internal physics-based simulation as in Section 4.1. First, we identify the region in the configuration space accessible from the current robot position (Fig. 10). We then relate each vertex in the configuration space graph to the minimum of the distance from the vertex to the region's graph center and that to the current robot position. Local maxima of the computed distances are chosen as candidates (Fig. 10 left). For each candidate, we compute a simulated robot motion where the robot moves towards the candidate pulling the cable, and test whether or not the motion clears the path. We use a simple spring-mass model to simulate the behavior of a cable. If a candidate clears the path in simulation, we select the candidate as the intermediate target. If no candidate clears the path to the goal, we choose the candidate that is expected to maximize the accessible region after the robot arrives at the target (Fig. 10 right).

The proposed algorithm is heuristic and is not guaranteed to find a solution if there is a solution. An alternative, systematic approach is to construct a search tree by recursively sampling candidates for the intermediate targets and search for a successful sequence of intermediate targets as in many path planning algorithms [1]. We did not implement such systematic approaches because our simple heuristic successfully found a path to the target via multiple intermediate targets in our experiments (Section 5) when there was a solution. When there is no solution, neither our heuristic method nor systematic approach can find one.

4.4 Deadlock Prevention

The algorithm described in the previous subsection cannot find a path when the robot is already trapped in a deadlock configuration as shown in Fig. 11 (left). The robot is trapped in the closed region and none of the auxiliary motions described above are able to open the way to the goal. To prevent this problem from occurring, we augmented the algorithm with a preprocessing step that removes configurations that can cause deadlocks from **G**. We then use the previously described runtime algorithms to find a deadlock free path to the target. This algorithm preforms well (Section 5) but does not guarantee deadlock avoidance. It is our future work to develop a complete run-time algorithm for deadlock prevention.

Starting from each graph node, the algorithm follows the path to the cable base by picking each adjacent vertex with minimum distance to the base. It identifies graph vertices that are in contact with an obstacle, yet their parents are not adjacent to an obstacle. These contact vertices are potential locations where a deadlock can occur (stars in Fig. 11 right).

Having identified contact vertices, the system examines whether or not the contacts are resolvable as follows. First, we compute a region in the configuration space separated by the path to the cable base and accessible from the contact vertex (gray area in Fig. 11). We then compare the maximum distance to any vertex in the region from the contact vertex and the *remaining cable length*. This is computed by subtracting the distance from the cable base to the contact node from the overall cable length. If the maximum distance is longer than the remaining cable length, then the

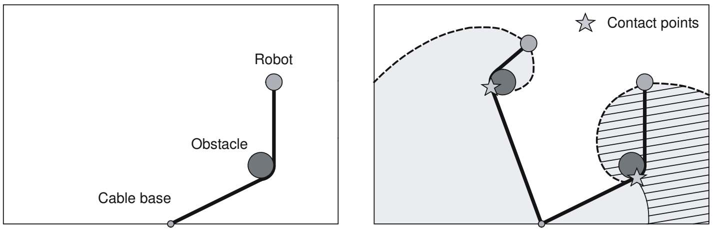


Fig. 11 Deadlock configuration (left) and the detection of potential deadlocks (right). Left contact (star) is resolvable, but the right contact (star) is not resolvable. When a potential deadlock is detected, we remove the affected area from the configuration space (shaded area).

contact is resolvable by moving the robot to the most distant position. Otherwise, the contact causes a deadlock. In this case, our system prevents the robot from causing the deadlock by removing all the configuration space vertices in the accessible area for which the distance longer than the one to the contact points (shaded in Fig. 11).

5 Experiments

In order to validate the practical effectiveness of the proposed algorithms, we conducted a series of experiments on a physical robot. We examined the basic case involving overhead tracking for robot position and a robot with no hardware for cable retraction. The cable configuration was not tracked but predicted by means of internal physics-based simulations for the algorithms in Section 4. Hence, our robot was not guaranteed to avoid cable-robot collision 100%. However, the experiments demonstrate that our algorithm significantly reduced the occurrence of collisions.

We evaluated the proposed algorithm using a cabled robot on a flat floor. An iRobot CREATE robot was connected to a cable that provided power and control signal for a total of 5 bundled wires. The location of the robot was tracked by a vision-based motion capture system (Motion Analysis). The system consisted of 8 infra-read high speed cameras that observe the motion of retro-reflective markers attached to the robot. The control PC (Dell Latitude) received the robot location from motion capture and sent control commands to the robot via the cable.

Fig. 12 gives an overview of the physical environment. It is a standard office floor covered by carpet. The layout mimics an open office or home environment with obstacles such as columns and furniture. The size is $5m \times 3m$. Our algorithm represented this space with 50×30 grid. Fig. 13 shows the layouts used in the experiment. In each trial, the robot was placed near the cable base with a compactly assembled cable. It visited six given targets in a given order. The system judged that a target visit was complete when the distance between the robot center and the target center was less than the robot diameter. We ran 10 trials for every combination of a given algorithm and layout. We prepared a set of 10 random permutations of 6 targets and used the same set for all combinations.

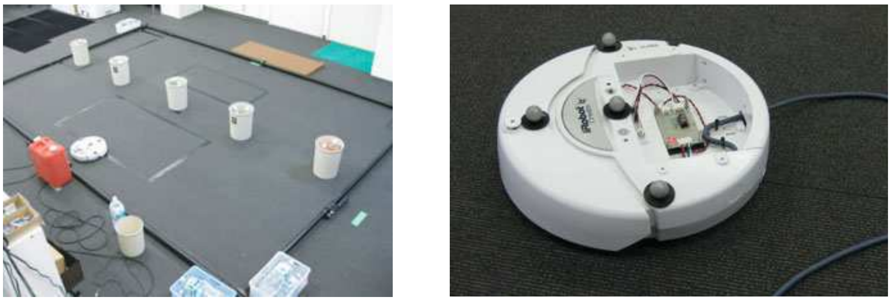


Fig. 12 The physical environment and the cabled robot used in our experiments.

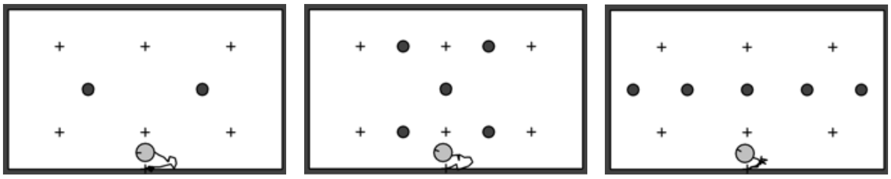


Fig. 13 Experimental layouts: dark gray circles are obstacles and plus marks are targets.

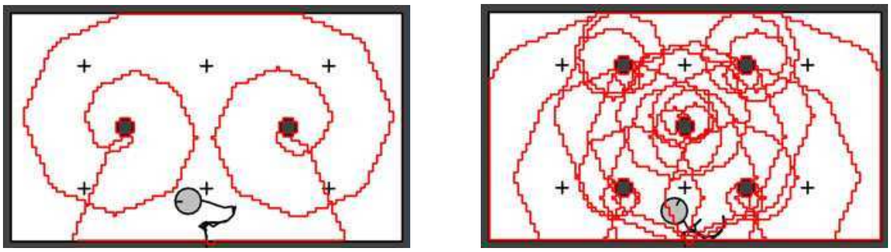


Fig. 14 Configuration space boundaries for two experimental layouts.

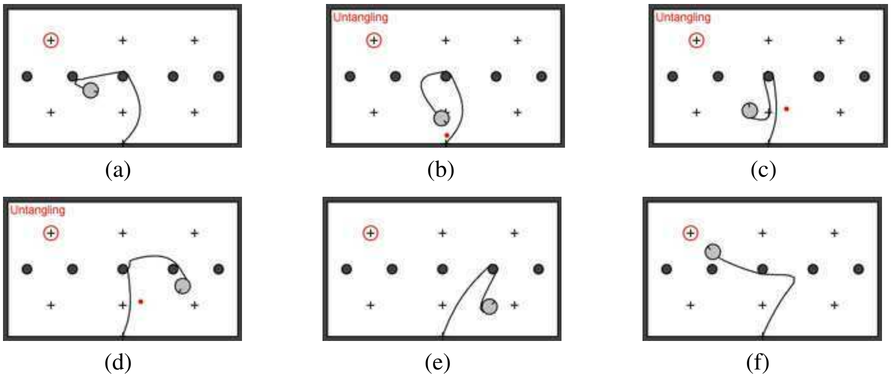


Fig. 15 Sample robot experiment with untangling: a) Initial configuration and goal (red circle) b) Approaching the first intermediate target (red dot) c) Approaching the second intermediate target. d) Arriving at the second intermediate target. e-f) Approaching and arriving at the goal.

5.1 Experimental Results

Table 1 shows the statistics of our results. The basic algorithm completed the tasks with 100% success. The extended algorithm without deadlock prevention failed in some cases (50 – 80% success). However, adding deadlock prevention achieved 100% success. Collisions between the robot and the cable did occur even when we used the extended algorithm. However the number of collisions was significantly reduced compared with the basic one. Fig. 14 shows the configuration space for

Table 1 Results from the experiments. We ran 10 trials for each combination of algorithm × task.

	Basic Algorithm			Extended Algorithm			Deadlock Prevention		
	Task1	Task2	Task3	Task1	Task2	Task3	Task1	Task2	Task3
Success Ratio	100%	100%	100%	80%	50%	80%	100%	100%	100%
Average Time (s)	83.9	100.4	88.5	100.9	103.2	93.0	89.4	110.4	92.5
Avg. Cable-Robot Collisions	1.8	1.7	2	0.25	0.4	0.125	0.4	0.3	0.2

the first two layouts. Fig. 15 shows an example of untangling observed during the experiments. It demonstrates that our algorithm successfully identified an appropriate sequence of intermediate targets. Video of the experiments and demonstration software are available at: http://www.designinterface.jp/en/projects/cable.

6 Conclusion

Our work shows that path planning for cabled robots yields significant insight into homotopic path planning. We developed a configuration space formulation that distinguishes between robot positions with distinct cable configurations. We proposed complete algorithms that compute the configuration space manifold and plan optimal paths given cable length constraints. Furthermore, we studied a practical extension of our algorithm given that the robot is not permitted to cross its cable. These algorithms were validated on a real robot platform in a series of experiments.

This paper opens the door to numerous variations of planning homotopic paths and cabled robotics. Immediate future work is the development of runtime lookahead detection of deadlocks. An interesting variant is path planning for robots that grasp or push the cable [31]. Another interesting problem is optimal placement of the cable base for a given environment to minimize deadlocks. Similar analysis would identify problematic obstacles that can cause deadlocks and warn the user.

Acknowledgements. We thank Daisuke Sakamoto, Takanori Yoshikawa, and Yoichi Kamiyama for their help in the experiments. We are grateful to Prof John F. Hughes for valuable discussions and to the anonymous reviewers for their insightful feedback.

References

[1] Latombe, J.C.: Robot motion planning. Springer, Heidelberg (1990)
[2] LaValle, S.M.: Planning algorithms. Cambridge Univ. Pr., Cambridge (2006)
[3] Hert, S., Lumelsky, V.: The ties that bind: Motion planning for multiple tethered robots. Robotics and Autonomous Systems 17(3), 187–215 (1996)
[4] Kavraki, L.E., Svestka, P., Latombe, J.C., Overmars, M.H.: Probabilistic roadmaps for path planning in high-dimensionalconfiguration spaces. IEEE Transactions on Robotics and Automation 12(4), 566–580 (1996)

[5] LaValle, S., Kuffner, J.: Rapidly-exploring random trees: Progress and prospects. In: Algorithmic and computational robotics: new directions: Workshop on the Algorithmic Foundations of Robotics, p. 293. AK Peters, Ltd. (2001)
[6] Kavraki, L., Lamiraux, F., Holleman, C.: Towards planning for elastic objects. In: 1998 Workshop on the Algorithmic Foundations of Robotics, pp. 313–325 (1998)
[7] Bayazit, O.B., Lien, J.M., Amato, N.M.: Probabilistic roadmap motion planning for deformable objects. In: IEEE Int. Conf. on Robotics and Automation, vol. 2, pp. 2126–2133 (2002)
[8] Kabul, I., Gayle, R., Lin, M.C.: Cable route planning in complex environments using constrained sampling. In: Proceedings of the 2007 ACM symposium on Solid and physical modeling, p. 402. ACM, New York (2007)
[9] Anshelevich, E., Owens, S., Lamiraux, F., Kavraki, L.: Deformable volumes in path planning applications. In: Proc. IEEE Int. Conf. Robot. Autom. (ICRA), pp. 2290–2295 (2000)
[10] Lamiraux, F., Kavraki, L.E.: Planning paths for elastic objects under manipulation constraints. The International Journal of Robotics Research 20(3), 188 (2001)
[11] Alterovitz, R., Pouliot, J., Taschereau, R., Hsu, I.C., Goldberg, K.: Sensorless planning for medical needle insertion procedures. In: IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, vol. 4, pp. 3337–3343 (2003) (Citeseer)
[12] Duindam, V., Alterovitz, R., Sastry, S., Goldberg, K.: Screw-based motion planning for bevel-tip flexible needles in 3d environments with obstacles. In: IEEE Intl. Conf. on Robot. and Autom, Citeseer, pp. 2483–2488 (2008)
[13] Sira-Ramirez, H.J.: On the control of the underactuated ship: A trajectory planning approach. In: IEEE Conference on Decision and Control, vol. 3, pp. 2192–2197 (1999)
[14] De Luca, A., Oriolo, G.: Motion planning and trajectory control of an underactuated three-link robot via dynamic feedback linearization. In: IEEE Int. Conf. on Robotics and Automation, vol. 3, pp. 2789–2795 (2000)
[15] Bergerman, M., Xu, Y.: Planning collision-free motions for underactuated manipulators in constrained configuration space. In: IEEE Int. Conf. on Robotics and Automation, pp. 549–555 (1997)
[16] Saha, M., Isto, P., Latombe, J.C.: Motion planning for robotic manipulation of deformable linear objects. In: Experimental Robotics, pp. 23–32. Springer, Heidelberg (2008)
[17] Brock, O., Khatib, O.: Real-time re-planning in high-dimensional configuration spacesusing sets of homotopic paths. In: Proceedings of IEEE Int. Conf. on Robotics and Automation, ICRA 2000, vol. 1 (2000)
[18] Schmitzberger, E., Bouchet, J.L., Dufaut, M., Wolf, D., Husson, R.: Capture of homotopy classes with probabilistic road map. In: IEEE/RSJ Int. Conf. on Intelligent Robots and System, vol. 3 (2002)
[19] Jaillet, L., Siméon, T.: Path deformation roadmaps. In: Algorithmic Foundation of Robotics VII, pp. 19–34
[20] Hershberger, J., Snoeyink, J.: Computing minimum length paths of a given homotopy class. Computational geometry 4(2), 63–97 (1994)
[21] Grigoriev, D., Slissenko, A.: Polytime algorithm for the shortest path in a homotopy class amidst semi-algebraic obstacles in the plane. In: ISSAC 1998: Proceedings of the 1998 Int. Symp. on Symbolic and algebraic computation, pp. 17–24 (1998)
[22] Bhattacharya, S., Kumar, V., Likhachev, M.: Search-based Path Planning with Homotopy Class Constraints. In: Proceedings of the Conference on Artificial Intelligence, AAAI (2010)

[23] Tenenbaum, J.B., Silva, V., Langford, J.C.: A global geometric framework for nonlinear dimensionality reduction. Science 290(5500), 2319 (2000)

[24] Jenkins, O.C., Matarić, M.J.: A spatio-temporal extension to isomap nonlinear dimension reduction. In: Int. Conf. on Machine learning, p. 56. ACM, New York (2004)

[25] Barraquand, J.: Automatic motion planning for complex articulated bodies. In: Paris Research Laboratory, Tech. Rep. (1991)

[26] Connolly, C.I., Grupen, R.A.: Harmonic control. In: Proceedings of the IEEE International Symposium on Intelligent Control, pp. 503–506

[27] Brown, J., Latombe, J.C., Montgomery, K.: Real-time knot-tying simulation. The Visual Computer 20(2), 165–179 (2004)

[28] Moll, M., Kavraki, L.E.: Path planning for deformable linear objects. IEEE Transactions on Robotics 22(4), 625–636 (2006)

[29] Yoshioka, T., Goto, K., Sato, T., Oki, H., Tsukune, H.: A Path-Planning for Mobile Robot Leading Cable Around. Nippon Kikai Gakkai Robotikusu, Mekatoronikusu Koenkai Koen Ronbunshu, 2002(Pt 1):1A1 (2002)

[30] Korf, R.E., Reid, M.: Complexity analysis admissible heuristic search. In: Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence. p. 310, American Association for Artificial Intelligence (1998)

[31] Stilman, M., Kuffner, J.: Navigation among movable obstacles: Real-time reasoning in complex environments. In: IEEE/RAS Int. Conf. on Humanoid Robots, pp. 322–341 (2004)

An Equivalence Relation for Local Path Sets

Ross A. Knepper, Siddhartha S. Srinivasa, and Matthew T. Mason

Abstract. We propose a novel enhancement to the task of collision-testing a set of local paths. Our approach circumvents expensive collision-tests, yet it declares a continuum of paths collision-free by exploiting both the structure of paths and the outcome of previous tests. We define a homotopy-like equivalence relation among local paths and provide algorithms to (1) classify paths based on equivalence, and (2) implicitly collision-test up to 90% of them. We then prove both correctness and completeness of these algorithms before providing experimental results showing a performance increase up to 300%.

1 Introduction

Planning bounded-curvature paths for mobile robots is an NP-hard problem [22]. Many nonholonomic mobile robots thus rely on hierarchical planning architectures [1, 13, 19], which split responsibility between at least two layers (Fig. 1): a slow global planner and fast local planner. We focus here on the local planner (Alg. 1 and Alg. 2), which iterates in a tight loop: searching through a set of paths and selecting the best path for execution. During each loop, the planner tests many paths before making an informed decision. The bottleneck in path testing is collision-testing [24]. In this paper, we introduce a novel approach that delivers a significant increase in path set collision-testing performance by exploiting the fundamental geometric structure of paths.

We introduce an equivalence relation intuitively resembling the topological notion of homotopy. Two paths are *path homotopic* if a continuous, collision-free deformation with fixed start and end points exists between them [20]. Like any path

Ross A. Knepper · Matthew T. Mason
Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh PA
e-mail: {rak,mason}@ri.cmu.edu

Siddhartha S. Srinivasa
Intel Labs Pittsburgh, 4720 Forbes Ave, Pittsburgh, PA
e-mail: siddhartha.srinivasa@intel.com

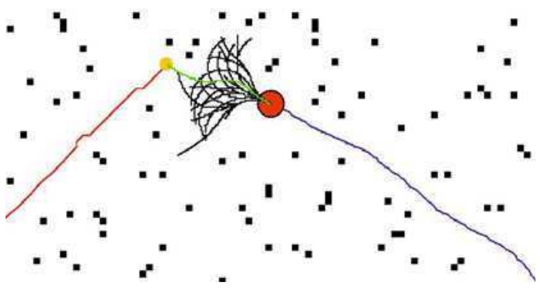


Fig. 1 An example hierarchical planning scenario. The local planner's path set expands from the robot, at center, and feeds commands to the robot based on the best path that avoids obstacles (black squares). The chosen local path (green) and global path (red) combine to form a proposed path to the goal.

equivalence relation, homotopy partitions paths into equivalence classes. Different homotopy classes make fundamentally different choices about their route amongst obstacles. However, two mobile robot concepts translate poorly into homotopy theory: limited sensing and constrained action.

The robot may lack a complete workspace map, which must instead be constructed from sensor data. Since robot perception is limited by range and occlusion, a robot's understanding of obstacles blocking its movement evolves with its vantage point. A variety of sensor-based planning algorithms have been developed to handle such partial information. Obstacle avoidance methods, such as potential fields [12], are purely reactive. The bug algorithm [18], which generates a path to the goal using only a contact sensor, is complete in 2D. Choset and Burdick [5] present the hierarchical generalized Voronoi graph, a roadmap with global line-of-sight accessibility that achieves completeness in higher dimensions using range readings of the environment.

If a robot is tasked to perform long-range navigation, then it must plan a path through unsensed regions. A low-fidelity global planner generates this path because we prefer to avoid significant investment in this plan, which will likely be invalidated later. Path homotopy, in the strictest sense, requires global knowledge of obstacles because homotopy equivalent paths must connect fixed start and goal points.

Relaxing the endpoint requirement avoids reasoning about the existence of faraway, unsensed obstacles. Naively relaxing a fixed endpoint, our paths might be permitted to freely deform around obstacles, making all paths equivalent. To restore meaningful equivalence classes, we propose an alternate constraint based on path shape. This is in keeping with the nonholonomic constraints that limit mobile robots' action. Laumond [15] first highlighted the importance of nonholonomic constraints and showed that feasible paths exist for a mobile robot with such constraints. Barraquand and Latombe [2] created a grid-based planner that innately captures these constraints. LaValle and Kuffner [17] proposed the first planner to incorporate both kinodynamic constraints and random sampling. In contrast to nonholonomic constraints, true homotopy forbids restrictions on path shape; two paths are equivalent if any path deformation—however baroque—exists between them. By restricting our paths to bounded curvature, we represent only feasible motions while limiting paths' ability to deform around obstacles. The resulting set of path

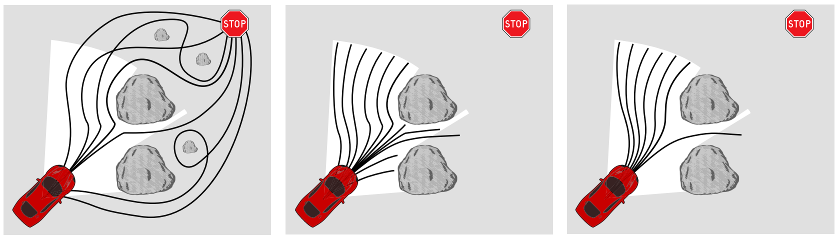


Fig. 2 **left:** Paths from a few distinct homotopy classes between the robot and the goal. The distinctions between some classes require information that the robot has not yet sensed (the dark area is out of range or occluded). **middle:** With paths restricted to the sensed area, they may freely deform around visible obstacles. **right:** After restricting path shape to conform to motion constraints, we get a handful of equivalence classes that are immediately applicable to the robot.

equivalence classes is of immediate importance to the planner (Fig. 2). The number of choices represented by these local equivalence classes relates to Farber's topological complexity of motion planning [6].

Equivalence classes have been employed in various planners. In task planning, recent work has shown that equivalence classes of actions can be used to eliminate redundant search [7]. In motion planning, path equivalence often employs homotopy. A recent paper by Bhattacharya, Kumar, and Likhachev [3] provides a technique based on complex analysis for detecting homotopic equivalence among paths in 2D. Two papers employing equivalence classes to build probabilistic roadmaps [11] are by Schmitzberger, et al. [25] and Jaillet and Simeon [10]. The latter paper departs from true homotopy by proposing the visibility deformation, a simplified alternative to homotopic equivalence based on line-of-sight visibility between paths.

Our key insight is that local path equivalence is an expressive and powerful tool that reveals shared outcomes in collision-testing. Specifically, two equivalent neighboring paths cover some common ground in the workspace, and between them lies a continuum of covered paths. We develop the mathematical foundations to detect equivalence relations among all local paths based on a finite precomputed path set. We then utilize these tools to devise efficient algorithms for detecting equivalence and implicitly collision-testing local paths.

The remainder of the paper is organized as follows. We provide an implementation of the basic algorithm in Section 2 and present the fast collision-testing technique. Section 3 then explores the theoretical foundations of our path equivalence relation. Section 4 provides some experimental results.

2 Algorithms

In this section, we present three algorithms: path set generation, path classification, and implicit path collision-testing. All of the algorithms presented here run in polynomial time. Throughout this paper, we use lowercase p to refer to a path in the workspace, while $\mathcal{P}$ is a set of paths (each one a point in path space).

Algorithm 1. Test_All_Paths(w, $\mathcal{P}$)

```
Input: $w$ – a costmap object; $\mathcal{P}$ – a fixed set of paths
Output: $\mathcal{P}_{free}$, the set of paths that passed collision test
1: $\mathcal{P}_{free} \leftarrow \emptyset$
2: while time not expired and untested paths remain do          // test paths for 0.1 seconds
3:     $p \leftarrow Get\_Next\_Path(\mathcal{P})$
4:     $collision \leftarrow w.Test\_Path(p)$                        // $collision$ is boolean
5:     if not $collision$ then
6:         $\mathcal{P}_{free} \leftarrow \mathcal{P}_{free} \cup \{p\}$               // non-colliding path set
7: return $\mathcal{P}_{free}$
```

Algorithm 2. Local_Planner_Algorithm(w, x, h, $\mathcal{P}$)

```
Input: $w$ – a costmap object; $x$ – initial state; $h$ – a heuristic function for selecting a path to
    execute;
    $\mathcal{P}$ – a fixed set of paths
Output: Moves the robot to the goal if possible
1: while not at goal and time not expired do
2:     $\mathcal{P}_{free} \leftarrow Test\_All\_Paths(w, x, \mathcal{P})$
3:     $j \leftarrow h.Best\_Path(x, \mathcal{P}_{free})$
4:     $Execute\_Path\_On\_Robot(j)$
5:     $x \leftarrow Predict\_Next\_State(x, j)$
```

Definition 1. *Path space* is a metric space $(\mathcal{P}, \mu)$ in which the distance between a pair of paths in $\mathcal{P}$ is defined by metric μ. Paths can vary in shape and length. □

2.1 Path Set Generation

We use the greedy path set construction technique of Green and Kelly [8], outlined in Alg. 3. The algorithm iteratively builds a path set $\mathcal{P}_N$ by drawing paths from a densely-sampled source path set, $\mathcal{X}$. During step i, it selects the path $p \in \mathcal{X}$ that minimizes the dispersion of $\mathcal{P}_i = \mathcal{P}_{i-1} \cup \{p\}$. Borrowing from Niederreiter [21]:

Definition 2. Given a bounded metric space $(\mathcal{X}, \mu)$ and point set $\mathcal{P} = \{x_1, \ldots, x_N\} \in \mathcal{X}$, the *dispersion* of $\mathcal{P}$ in $\mathcal{X}$ is defined by

$$\delta(\mathcal{P}, \mathcal{X}) = \sup_{x \in \mathcal{X}} \min_{p \in \mathcal{P}} \mu(x, p) \tag{1}$$

□

The dispersion of $\mathcal{P}$ in $\mathcal{X}$ equals the radius of the biggest open ball in $\mathcal{X}$ containing no points in $\mathcal{P}$. By minimizing dispersion, we ensure that there are no large voids in path space. Thus, dispersion reveals the quality of $\mathcal{P}$ as an "approximation" of $\mathcal{X}$ because it guarantees that for any $x \in \mathcal{X}$, there is some point $p \in \mathcal{P}$ such that $\mu(x, p) \leq \delta(\mathcal{P}, \mathcal{X})$.

The Green-Kelly algorithm generates a sequence of path sets $\mathcal{P}_i$, for $i \in \{1, \ldots, N\}$, that has monotonically decreasing dispersion. Alg. 1 searches paths in this order at

Algorithm 3. Green_Kelly($\mathcal{X}$, N)

```
Input: $\mathcal{X}$ – a densely-sampled, low-dispersion path set; $N \leq |\mathcal{X}|$ – the target path set size
Output: path sequence $\mathcal{P}_N$ of size $N$
1: $\mathcal{P}_0 \leftarrow \emptyset$
2: $n \leftarrow 0$
3: while $n < N$ do
4:     $n \leftarrow n+1$
5:     $p \leftarrow \underset{x \in \mathcal{X}}{\operatorname{argmin}}\ \delta(\mathcal{P}_{n-1} \cup \{x\}, \mathcal{X})$
6:     $\mathcal{P}_n \leftarrow \mathcal{P}_{n-1} \cup \{p\}$
7: return $\mathcal{P}_N$
```

runtime, thus permitting early termination while retaining near-optimal results. Note that while the source set $\mathcal{X}$ is of finite size—providing a lower bound on dispersion at runtime—it can be chosen with arbitrarily low dispersion a priori.

2.2 *Path Classification*

We next present Alg. 4, which classifies collision-free members of a path set. The Hausdorff metric is central to the algorithm. Intuitively, this metric returns the greatest amount of separation between two paths in the workspace. From Munkres [20]:

$$\mu_H(p_i, p_j) = \inf\{\varepsilon \colon p_i \subset (p_j)_\varepsilon \text{ and } p_j \subset (p_i)_\varepsilon\}, \tag{2}$$

where $(p)_r$ denotes dilation of p by r: $\{t \in \mathbb{R}^2 : \|t_p - t\|_{L2} \leq r \text{ for some } t_p \in p\}$. Note that μ_H satisfies all properties of a metric [9]. For our fixed path set generated by Green-Kelly, we precomputed each pairwise path metric value of (2) and stored them in a lookup table for rapid online access.

Alg. 4 performs path classification on a set of paths that have already tested collision-free at runtime. We form a graph $G = (V, E)$ in which node $v_i \in V$ corresponds to path p_i. Edge $e_{ij} \in E$ joins nodes v_i and v_j when this relation holds:

$$\mu_H(p_i, p_j) \leq d, \tag{3}$$

where d is the diameter of the robot. Taking the transitive closure of this relation, two paths p_a and p_b are equivalent if nodes v_a and v_b are in the same connected component of G (Fig. 3).

In effect, this algorithm constructs a probabilistic roadmap (PRM) in the path space instead of the conventional configuration space. A query into this PRM tells whether two paths are equivalent. As with any PRM, a query is performed by adding two new graph nodes v_s and v_g corresponding to the two paths. We attempt to join these nodes to other nodes in the graph based on (3). The existence of a path connecting v_s to v_g indicates path equivalence.

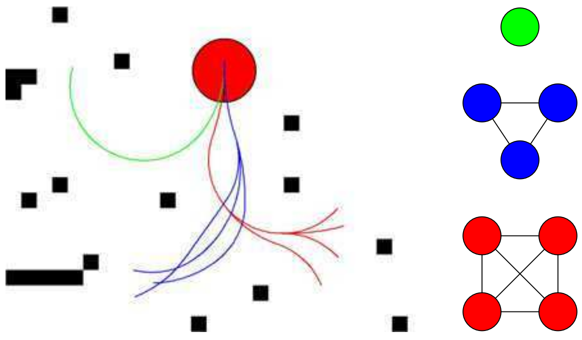


Fig. 3 A simple path set, in which obstacles (black) eliminate colliding paths. The collision-free path set has three equivalence classes (red, green, and blue). In the corresponding graph representation, at right, adjacent nodes represent proximal paths. Connected components indicate equivalence classes of paths.

Algorithm 4. Equivalence_Classes($\mathcal{P}_{free}$, d)

```
Input: $\mathcal{P}_{free}$ – a set of safe, appropriate paths; $d$ – the diameter of the robot
Output: $D$ – a partition of $\mathcal{P}_{free}$ into equivalence classes (a set of path sets)
 1: Let $G = (V,E) \leftarrow (\emptyset,\emptyset)$
 2: $D \leftarrow \emptyset$
 3: for all $p_i \in \mathcal{P}_{free}$ do                      // This loop discovers adjacency
 4:     $V.add(p_i)$                                // Add a graph node corresponding to path $p_i$
 5:     for all $p_j \in V \setminus p_i$ do
 6:         if $\mu_H(p_i, p_j) < d$ then
 7:             $E.add(i, j)$                 // Connect nodes $i$ and $j$ with an unweighted edge
 8: $\mathcal{S} \leftarrow \mathcal{P}_{free}$
 9: while $\mathcal{S} \neq \emptyset$ do                          // This loop finds the connected components
10:     $\mathcal{C} \leftarrow \emptyset$
11:     $p \leftarrow$ a member of $\mathcal{S}$
12:     $\mathcal{L} \leftarrow \{p\}$                              // List of nodes to be expanded in this class
13:     while $\mathcal{L} \neq \emptyset$ do
14:         $p \leftarrow$ a member of $\mathcal{L}$
15:         $\mathcal{C} \leftarrow \mathcal{C} \cup \{p\}$                                          // Commit $p$ to class
16:         $\mathcal{S} \leftarrow \mathcal{S} - \{p\}$
17:         $\mathcal{L} \leftarrow (\mathcal{L} \cup V.neighbors(p)) \cap \mathcal{S}$
18:     $D \leftarrow D \cup \{\mathcal{C}\}$
19: return $D$
```

2.3 Implicit Path Safety Test

There is an incessant need in motion planning to accelerate collision-testing, which may take 99% of total CPU time [24]. During collision-testing, the planner must verify that a given swath is free of obstacles.

Definition 3. A *swath* is the workspace area of ground or volume of space swept out as the robot traverses a path. □

Definition 4. We say a path is *safe* if its swath contains no obstacles. □

Algorithm 5. Test_Path_Implicit(p, w, $\mathcal{S}$, d)

```
Input: $p$ is a path to be tested
Input: $w$ is a costmap object        // used as a backup when path cannot be implicitly tested
Input: $\mathcal{S}$ is the set of safe paths found so far
Input: $d$ is the diameter of the robot
1: for all $p_i, p_j \in \mathcal{S}$ such that $\mu_H(p_i, p_j) \leq d$ do
2:     if $p.Is\_Between(p_i, p_j)$ then        // $p$'s swath has been tested previously
3:         $s_f \leftarrow p.Get\_End\_Point()$
4:         $collision \leftarrow w.Test\_Point(s_f)$        // endpoint may not be covered by swaths
5:         return $collision$
6: return $w.Test\_Path(p)$        // Fall back to explicit path test
```

In testing many swaths of a robot passing through space, most planners effectively test the free workspace many times by testing overlapping swaths. We may test a path implicitly at significant computational savings by recalling recent collision-testing outcomes. We formalize the idea in Alg. 5, which is designed to be invoked from Alg. 1, line 4 in lieu of the standard path test routine.

The implicit collision-test condition resembles the neighbor condition (3) used by Alg. 4, but it has an additional "Is_Between" check, which indicates that the swath of the path under test is covered by two collision-free neighboring swaths. The betweenness trait can be precomputed and stored in a lookup table. Given a set of safe paths, we can quickly discover whether any pair covers the path under test. Experimental results show that this algorithm allows us to test up to 90% of paths implicitly, thus increasing the path evaluation rate by up to 300% in experiments.

3 Foundations

In this section, we establish the foundations of an equivalence relation on path space based on continuous deformations between paths. We then provide correctness proofs for our algorithms for classification and implicit collision-testing.

We assume a kinematic description of paths. All paths are parametrized by a shared initial pose, shared fixed length, and individual curvature function. Let $\kappa_i(s)$ describe the curvature control of path i as a function of arc length, with $\max_{0 \leq s \leq s_f} |\kappa_i(s)| \leq \kappa_{max}$. Typical expressions for κ_i include polynomials, piecewise constant functions, and piecewise linear functions. The robot motion produced by control i is a *feasible* path given by

$$\begin{bmatrix} \dot{\theta}_i \\ \dot{x}_i \\ \dot{y}_i \end{bmatrix} = \begin{bmatrix} \kappa_i \\ \cos\theta_i \\ \sin\theta_i \end{bmatrix}. \tag{4}$$

Definition 5. A *feasible* path has bounded curvature (implying C^1 continuity) and fixed length. The set $\mathcal{F}(s_f, \kappa_{max})$ contains all feasible paths of length s_f and curvature $|\kappa(s)| \leq \kappa_{max}$. □

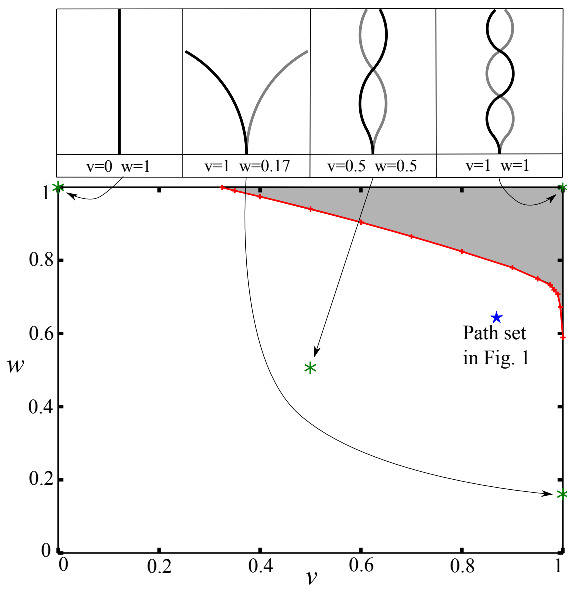


Fig. 4 **At top:** several example paths combining different values of v and w. Each path pair obeys (3). The value of v affects the "curviness" allowed in paths, while w affects their length. **At bottom:** this plot, generated numerically, approximates the set of appropriate choices for v and w. The gray region at top right must be avoided, as we show in Lemma 2. Such choices would permit an obstacle to occur between two safe paths that obey (3). A path whose values fall in the white region is called an *appropriate path*.

3.1 Properties of Paths

In this section, we establish a small set of conditions under which we can quickly determine that two paths are equivalent. We constrain path shape through two dimensionless ratios relating three physical parameters. We may then detect equivalence through a simple test on pairs of paths using the Hausdorff metric.

These constraints ensure a continuous deformation between neighboring paths while permitting a range of useful actions. Many important classes of action sets obey these general constraints, including the line segments common in RRT [17] and PRM planners, as well as constant curvature arcs. Fig. 1 illustrates a more expressive action set [13] that adheres to our constraints.

The three physical parameters are: d, the diameter of the robot; s_f, the length of each path; and r_{min}, the minimum radius of curvature allowed for any path. Note that $1/r_{min} = \kappa_{max}$, the upper bound on curvature. For non-circular robots, d reflects the minimal cross-section of the robot's swath sweeping along a path. We express relationships among the three physical quantities by two dimensionless parameters:

$$v = \frac{d}{r_{min}} \qquad\qquad w = \frac{s_f}{2\pi r_{min}}.$$

We only compare paths with like values of v and w. Fig. 4(top) provides some intuition on the effect of these parameters on path shape. Due to the geometry of paths, only certain choices of v and w are appropriate.

Definition 6. An *appropriate path* is a feasible path conforming to appropriate values of v and w from the proof of Lemma 2. Fig. 4 previews the permissible values.

When the condition in (3) is met, the two paths' swaths overlap, resulting in a continuum of coverage between the paths. This coverage, in turn, ensures the existence of a continuous deformation, as we show in Theorem 1, but first we formally define a continuous deformation between paths.

Definition 7. A *continuous deformation* between two safe, feasible paths p_i and p_j in $\mathcal{F}(s_f, \kappa_{max})$ is a continuous function $f\colon\ [0,1] \to \mathcal{F}(s_f^-, \kappa_{max}^+)$, with s_f^- slightly less than s_f and κ_{max}^+ slightly more than κ_{max}. $f(0)$ is the initial interval of p_i, and $f(1)$ is the initial interval of p_j, both of length s_f^-. We write $p_i \sim p_j$ to indicate that a continuous deformation exists between paths p_i and p_j, and they are therefore equivalent. □

The length s_f^- depends on v and w, but for typical values, s_f^- is fully 95–98% of s_f. For many applications, this is sufficient, but an application can quickly test the remaining path length if necessary. Nearly all paths $f(c)$ are bounded by curvature κ_{max}, but it will turn out that in certain geometric circumstances, the maximum curvature through a continuous deformation is up to $\kappa_{max}^+ = \frac{4}{3}\kappa_{max}$.

Definition 8. Two safe, feasible paths that define a continuous deformation are called *guard paths* because they protect the intermediate paths. □

In the presence of obstacles, it is not trivial to determine whether a continuous deformation is safe, thus maintaining equivalency. Rather than trying to find a deformation between arbitrary paths, we propose a particular condition under which we show that a bounded-curvature, fixed-length, continuous path deformation exists,

$$\mu_H(p_1, p_2) \le d \implies p_1 \sim p_2. \tag{5}$$

This statement, which we prove in the next section, is the basis for Alg. 4 and Alg. 5. The overlapping swaths of appropriate paths p_1 and p_2 cover a continuum of intermediate swaths between the two paths. Eqn. (5) is a proper equivalence relation because it possesses each of three properties:

- **reflexivity.** $\mu_H(p,p) = 0$; p is trivially deformable to itself.
- **symmetry.** The Hausdorff metric is symmetric.
- **transitivity.** Given $\mu_H(p_1,p_2) \le d$ and $\mu_H(p_2,p_3) \le d$, a continuous deformation from p_1 to p_3 passes through p_2.

3.2 Equivalence Relation

Having presented the set of conditions under which (5) holds, we now prove that they are sufficient to ensure the existence of a continuous deformation between two neighboring paths. Our approach to the proof will be to first describe a feasible continuous deformation, then show that paths along this deformation are safe.

Given appropriate guard paths p_i and p_j with common origin, let p_e be the shortest curve in the workspace connecting their endpoints without crossing either path (p_e may pass through obstacles). The closed path $B = p_i \cup p_j \cup p_e$ creates one or

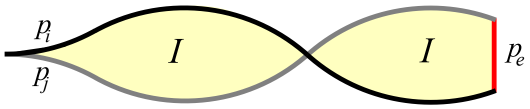


Fig. 5 Paths p_i, p_j, and p_e form boundary B. Its interior, I, contains all paths in the continuous deformation from p_i to p_j.

more closed loops (the paths may cross each other). By the Jordan curve theorem, each loop partitions $\mathbb{R}^2$ into two sets, only one of which is compact. Let I, the interior, be the union of these compact regions with B, as in Fig. 5.

Definition 9. A path p_c is *between* paths p_i and p_j if $p_c \subset I$. □

Lemma 1. *Given appropriate paths $p_i, p_j \subset \mathcal{F}(s_f, \kappa_{max})$ with $\mu_H(p_i, p_j) \leq d$, a path sequence exists in the form of a feasible continuous deformation between p_i and p_j.*

Proof. We provide the form of a continuous deformation from p_i to p_j such that each intermediate path is between them. With t a workspace point and p a path, let

$$\gamma(t,p) = \inf\{\varepsilon : t \in (p)_\varepsilon\} \tag{6}$$

$$g(t) = \begin{cases} [0,1] & \text{if } \gamma(t,p_i) = \gamma(t,p_j) = 0 \\ \left\{ \frac{\gamma(t,p_i)}{\gamma(t,p_i)+\gamma(t,p_j)} \right\} & \text{otherwise,} \end{cases} \tag{7}$$

where $g(t)$ is a set-valued function to accommodate intersecting paths. Each level set $g(t) = c$ for $c \in [0,1]$ defines a weighted generalized Voronoi diagram (GVD) forming a path as in Fig. 6. We give the form of a continuous deformation using level sets $g^{-1}(c)$; each path is parametrized starting at the origin and extending for a length s_f^- in the direction of p_e. Let us now pin down the value of s_f^-. Every point t_i on p_i forms a line segment projecting it to its nearest neighbor t_j on p_j (and vice versa). Their collective area is shown in Fig. 7. Eqn. (3) bounds each segment's length at d. s_f^- is the greatest value such that no intermediate path of length s_f^- departs from the region covered by these projections.

For general shapes in $\mathbb{R}^2$, the GVD forms a set of curves meeting at branching points [23]. In this case, no GVD cusps or branching points occur in any intermediate path. Since $d < r_{min}$, no center of curvature along either guard path can fall in I [4]. Therefore, each level set defines a path through the origin.

Each path's curvature function is piecewise continuous and everywhere bounded. A small neighborhood of either guard path approximates constant curvature. A GVD curve generated by two constant-curvature sets forms a conic section [27]. Table 1 reflects that the curvature of p_c is everywhere bounded with the maximum possible curvature being bounded by $\frac{4}{3}\kappa_{max}$. For the full proofs, see [14]. □

Lemma 2. *Given safe, appropriate guard paths $p_i, p_j \in \mathcal{F}(s_f, \kappa_{max})$ separated by $\mu_H(p_i, p_j) \leq d$, any path $p_c \subset \mathcal{F}(s_f^-, \frac{4}{3}\kappa_{max})$ between them is safe.*

Proof. We prove this lemma by contradiction. Assume an obstacle lies between p_i and p_j. We show that this assumption imposes lower bounds on v and w. We then conclude that for lesser values of v and w, no such obstacle can exist.

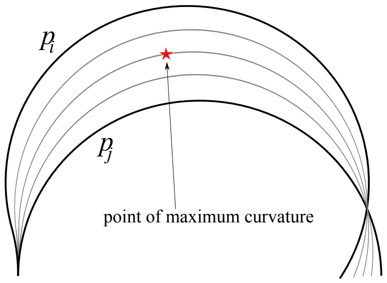


Fig. 6 In a continuous deformation between paths p_i and p_j, as defined by the level sets of (7), each path takes the form of a weighted GVD. Upper bounds on curvature vary along the deformation, with the maximum bound of $\frac{4}{3}\kappa_{max}$ occurring at the medial axis of the two paths.

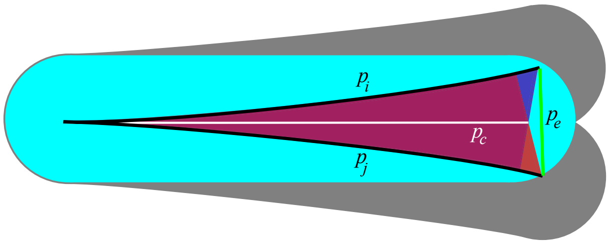


Fig. 7 Hausdorff coverage (overlapping red and blue shapes in center) is a conservative approximation of swath coverage (gray). The Hausdorff distance between paths p_i and p_j is equal to the maximum-length projection from any point on either path to the closest point on the opposite path. Each projection implies a line segment. The set of projections from the top line (blue) and bottom line (red) each cover a solid region between the paths. These areas, in turn, cover a slightly shorter intermediate path p_c, in white, with its swath in cyan. This path's length, s_f^- is as great as possible while remaining safe, with its swath inside the gray area.

Table 1 Conic sections form the weighted Voronoi diagram. κ_1 and κ_2 represent the curvatures of the two guard paths, with κ_1 the lesser magnitude. Let $\kappa_m = \max(|\kappa_1|, |\kappa_2|)$. For details, see [14].

Type	Occurrence	Curvature bounds of intermediate paths
line	$\kappa_1 = -\kappa_2$	$\lvert\kappa\rvert \leq \kappa_m$
parabola	$\kappa_1 = 0, \kappa_2 \neq 0$	$\lvert\kappa\rvert \leq \kappa_m$
hyperbola	$\kappa_1\kappa_2 < 0, \kappa_1 \neq -\kappa_2$	$\lvert\kappa\rvert \leq \kappa_m$
ellipse	$\kappa_1\kappa_2 > 0$	$\lvert\kappa\rvert < \frac{4}{3}\kappa_m$

Let $sl(p,d) = \{t \in \mathbb{R}^2, t_p = nn(t,p) \colon \overline{t_p t} \perp p \text{ and } \|t - t_p\|_{L2} \leq \frac{d}{2}\}$ define a conservative approximation of a swath, obtained by sweeping a line segment of length d with its center along the path. $\overline{t_p t}$ is the line segment joining t_p to t and $nn(t,p)$ is the nearest neighbor of point t on path p. The two swaths form a safe region, $U = sl(p_i,d) \cup sl(p_j,d)$.

Suppose that U contains a hole, denoted by the set h, which might contain an obstacle. Now, consider the shape of the paths that could produce such a hole. Beginning with equal position and heading, they must diverge widely enough to

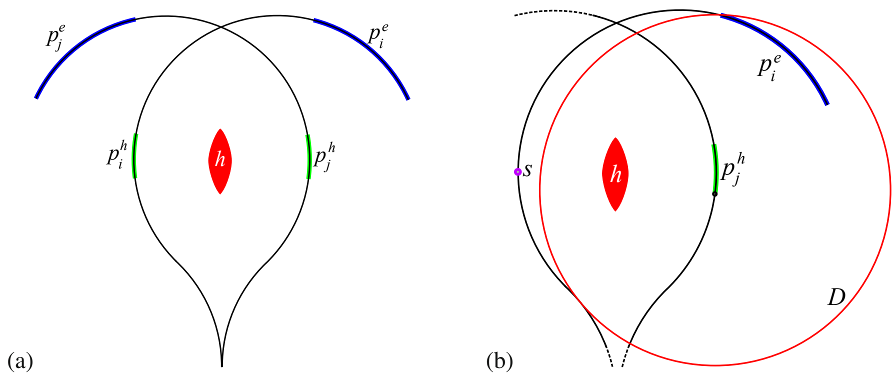


Fig. 8 (a) With bounded curvature, there is a lower bound on path lengths that permit a hole, h, while satisfying (3). Shorter path lengths ensure the existence of a safe continuous deformation between paths. (b) We compute the maximal path length that prevents a hole using Vendittelli's solution to the shortest path for a Dubins car. Starting from the dot marked s, we find the shortest path intersecting the circle D. The interval p_i^e illustrates path lengths permitting a hole to exist.

separate by more than d. To close the loop in U, the paths must then bend back towards each other. Since the paths separate by more than d, there exist two open intervals $p_i^h \subset p_i$ and $p_j^h \subset p_j$ surrounding the hole on each path such that (at this point) $p_i^h \not\subset (p_j)_d$ and $p_j^h \not\subset (p_i)_d$. To satisfy (3), there must exist later intervals $p_i^e \subset p_i$ such that $p_j^h \subset (p_i^e)_d$ and likewise $p_j^e \subset p_j$ such that $p_i^h \subset (p_j^e)_d$, as in Fig. 8a.

How long must a path be to satisfy this condition? Consider the minimum length solution to this problem under bounded curvature. For each point $t \in p_j^h$, the interval p_i^e must intersect the open disc $D = \text{int}((t)_d)$, as in Fig. 8b. Since p_j^h grows with the width of h, and p_i^e must intersect all of these open neighborhoods, the path becomes longer with larger holes. We will therefore consider the minimal small-hole case.

Vendittelli [26] solves the shortest path problem for a Dubins car to reach a line segment. We may approximate the circular boundary of D by a set of arbitrarily small line segments. One may show from this work that given the position and slope of points along any such circle, the shortest path to reach its boundary (and thus its interior) is a constant-curvature arc of radius r_{min}. In the limit, as v approaches one and the size of h approaches zero, the length of arc needed to satisfy (3) approaches $\pi/2$ from above, resulting in the condition that $w > 0.48$. Thus, for $w \leq 0.48$ and $v \in [0,1)$, p_c is safe. For smaller values of v, D shrinks relative to r_{min}, requiring longer paths to reach, thus allowing larger values of w as shown in the plot in Fig. 4.

We have shown that there exist appropriate choices for v and w such that (3) implies that U contains no holes. Since U contains the origin, any path $p_c \in I$ emanating from the origin passes through U and is safe. □

Theorem 1. *Given safe, appropriate guard paths $p_i, p_j \in \mathcal{F}(s_f, \kappa_{max})$, and given $\mu_H(p_i, p_j) \leq d$, a safe continuous deformation exists between p_i and p_j.*

Proof. Lemma 1 shows that (7) gives a continuous deformation between paths p_i and p_j such that each intermediate path $p_c \subset I$ is feasible. Lemma 2 shows that any such path is safe. Therefore, a continuous deformation exists between p_i and p_j. This proves the validity of the Hausdorff metric as a test for path equivalence. □

3.3 Resolution Completeness of Path Classifier

In this section, we show that Alg. 4 is resolution complete. Resolution completeness commonly shows that for a sufficiently high discretization of each dimension of the search space, the planner finds a path exactly when one exists in the continuum space. We instead show that for a sufficiently low dispersion in the infinite-dimensional path space, the approximation given by Alg. 4 has the same connectivity as the continuum safe, feasible path space.

Let $\mathcal{F}$ be the continuum feasible path space and $\mathcal{F}_{free} \subset \mathcal{F}$ be the set of safe, feasible paths. Using the Green-Kelly algorithm, we sample offline from $\mathcal{F}$ a path sequence $\mathcal{P}$ of size N. At runtime, using Alg. 1, we test members of $\mathcal{P}$ in order to discover a set $\mathcal{P}_{free} \subset \mathcal{P}$ of safe paths.

The following lemma is based on the work of LaValle, Branicky, and Lindemann [16], who prove resolution completeness of deterministic roadmap (DRM) planners, which are PRM planners that draw samples from a low-dispersion, deterministic source. Since we use a deterministic sequence provided by Green-Kelly, the combination of Alg. 1 and 4 generates a DRM in path space.

Lemma 3. *For any given configuration of obstacles and any path set $\mathcal{P}_N$ generated by the Green-Kelly algorithm, there exists a sufficiently large N such that any two paths $p_i, p_j \in \mathcal{P}_{free}$ are in the same connected component of $\mathcal{F}_{free}$ if and only if Alg. 4 reports that $p_i \sim p_j$.*

Proof. LaValle, et al. [16], show that by increasing N, a sufficiently low dispersion can be achieved to make a DRM complete in any given C-Space. By an identical argument, given a continuum connected component $\mathcal{C} \subset \mathcal{F}_{free}$, all sampled paths in $\mathcal{C} \cap \mathcal{P}_N$ are in a single partition of D. If q is the radius of the narrowest corridor in $\mathcal{C}$, then for dispersion $\delta_N < q$, our discrete approximation exactly replicates the connectivity of the continuum freespace. □

Lemma 4. *Under the same conditions as in Lemma 3, there exists a sufficiently large N such that for any continuum connected component $\mathcal{C} \subset \mathcal{F}_{free}$, Alg. 1 returns a $\mathcal{P}_{free}$ such that $\mathcal{P}_{free} \cap \mathcal{C} \neq \emptyset$. That is, every component in $\mathcal{F}_{free}$ has a corresponding partition returned by Alg. 4.*

Proof. Let B_r be the largest open ball of radius r in $\mathcal{C}$. When $\delta_N < r$, B_r must contain some sample $p \in \mathcal{P}$. Since $\mathcal{C}$ is entirely collision-free, $p \in \mathcal{P}_{free}$. Thus, for dispersion less than r, $\mathcal{P}_{free}$ contains a path in $\mathcal{C}$. □

There exists a sufficiently large N such that after N samples, $\mathcal{P}$ has achieved dispersion $\delta_N < \min(q, r)$, where q and r are the dispersion required by Lemmas 3 and 4, respectively. Under such conditions, a bijection exists between the connected components of $\mathcal{P}_{free}$ and $\mathcal{F}_{free}$.

Theorem 2. *Let $D = \{\mathcal{D}_1|\ldots|\mathcal{D}_m\}$ be a partition of $\mathcal{P}_{free}$ as defined by Alg. 4. Let $C = \{\mathcal{C}_1|\ldots|\mathcal{C}_m\}$ be a finite partition of the continuum safe, feasible path space into connected components. A bijection $f : D \to C$ exists such that $\mathcal{D}_i \subset f(\mathcal{D}_i)$.*

Proof. Lemma 3 establishes that f is one-to-one, while Lemma 4 establishes that f is onto. Therefore, f is bijective. This shows that by sampling at sufficiently high density, we can achieve an arbitrarily good approximation of the connectedness of the continuum set of collision-free paths in any environment. □

Theorem 3. *A path interval p may be implicitly tested safe if it is between paths p_i and p_j such that $\mu_H(p_i, p_j) \leq d$ and a small region at the end of p_c has been explicitly tested.*

Proof. By Lemma 2, the initial interval of p_c is safe because its swath is covered by the swaths of the guard paths. Since the small interval at the end of p_c has been explicitly tested, the whole of p_c is collision-free. □

4 Results

We briefly summarize some experimental results involving equivalence class detection and implicit path collision-testing. All tests were performed in simulation on planning problems of the type described in [13].

Path classification imposes a computational overhead due to the cost of searching collision-free paths. Collision rate in turn relates to the density of obstacles in the environment. The computational overhead of our classification implementation is

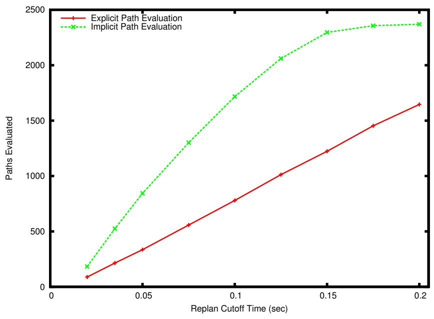


Fig. 9 Paths tested per time-limited replan step in an obstacle-free environment. Path testing performance improves by up to 3x with the algorithms we present here. Note that an artificial ceiling curtails performance at the high end due to a maximum path set of size 2,401.

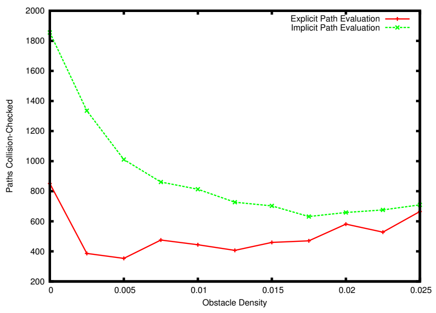


Fig. 10 Paths tested per 0.1 second time step at varying obstacle densities. Implicit collision-testing allows significantly more paths to be tested per unit time. Even in extremely dense clutter, implicit path testing considers an extra six paths on average.

nearly 20% in an empty environment but drops to 0.3% in dense clutter. However, implicit collision-testing more than compensates for this overhead.

Fig. 9 shows the effect of implicit path testing on total paths tested in the absence of obstacles. As the time limit increases, the number of paths collision-tested under the traditional algorithm increases linearly at a rate of 8,300 paths per second. With implicit testing, the initial test rate over small time limits (thus small path set sizes) is over 22,500 paths per second. The marginal rate declines over time due to the aforementioned overhead, but implicit path testing still maintains its speed advantage until the entire 2,401-member path set is collision-tested.

Fig. 10 presents implicit collision-testing performance in the presence of clutter. We compare the implicit collision-tester in Alg. 5 to traditional explicit collision-testing. When fixing the replan rate at 10 Hz, implicit path evaluation maintains an advantage, despite the overhead, across all navigable obstacle densities.

5 Discussion and Future Work

In this paper, we propose an equivalence relation on local paths based on the following constraints: fixed start position and heading, fixed length, and bounded curvature. We describe an algorithm for easily classifying paths using the Hausdorff distance between them. Path classification is a tool that permits collective reasoning about paths, leading to more efficient collision-testing.

There are many other applications for path equivalence. One example uses path class knowledge in obstacle avoidance to improve visibility and safety around obstacles. Another avenue of future work involves generalizing path equivalence to higher dimensions. For instance, an implicit path test for a robot floating in 3D requires three neighboring paths, while a manipulator arm needs only two.

Acknowledgements. This work is sponsored by the Defense Advanced Research Projects Agency. This work does not necessarily reflect the position or the policy of the Government. No official endorsement should be inferred. Thank you to Matthew Tesch, Laura Lindzey, and Alberto Rodriguez for valuable comments and discussions.

References

1. Allen, T., Underwood, J., Scheding, S.: A path planning system for autonomous ground vehicles operating in unstructured dynamic environments. In: Proc. Australasian Conference on Robotics and Automation (2007)
2. Barraquand, J., Latombe, J.-C.: Nonholonomic multibody mobile robots: Controllability and motion planning in the presence of obstacles. Algorithmica 10(2-3-4), 121–155 (1993)
3. Bhattacharya, S., Kumar, V., Likhachev, M.: Search-based path planning with homotopy class constraints. In: Proc. National Conference on Artificial Intelligence (2010)
4. Blum, H.: A transformation for extracting new descriptors of shape. In: Whaters-Dunn, W. (ed.) Proc. Symposium on Models for the Perception of Speech and Visual Form, pp. 362–380. MIT Press, Cambridge (1967)

5. Choset, H., Burdick, J.: Sensor based planning, part I: The generalized Voronoi graph. In: Proc. International Conference on Robotics and Automation, pp. 1649–1655 (1995)
6. Farber, M.: Topological complexity of motion planning. Discrete & Computational Geometry 29(2), 211–221 (2003)
7. Gardiol, N.H., Kaelbling, L.P.: Action-space partitioning for planning. In: National Conference on Artificial Intelligence, Vancouver, Canada (2007)
8. Green, C., Kelly, A.: Toward optimal sampling in the space of paths. In: Proc. International Symposium of Robotics Research, Hiroshima, Japan (November 2007)
9. Henrikson, J.: Completeness and total boundedness of the Hausdorff metric. The MIT Undergraduate Journal of Mathematics 1 (1999)
10. Jaillet, L., Simeon, T.: Path deformation roadmaps: Compact graphs with useful cycles for motion planning. International Journal of Robotics Research 27(11-12), 1175–1188 (2008)
11. Kavraki, L., Svestka, P., Latombe, J.-C., Overmars, M.: Probabilistic roadmaps for path planning in high-dimensional configuration spaces. In: Proc. International Conference on Robotics and Automation, pp. 566–580 (1996)
12. Khatib, O.: Real-time obstacle avoidance for manipulators and mobile robots. In: Proc. International Conference on Robotics and Automation, St. Louis, USA (March 1985)
13. Knepper, R.A., Mason, M.T.: Empirical sampling of path sets for local area motion planning. In: Proc. International Symposium of Experimental Robotics, Athens, Greece (July 2008)
14. Knepper, R.A., Srinivasa, S.S., Mason, M.T.: Curvature bounds on the weighted Voronoi diagram of two proximal paths with shape constraints. Technical Report CMU-RI-TR-10-25, Robotics Institute, Carnegie Mellon University (2010)
15. Laumond, J.P.: Feasible trajectories for mobile robots with kinematic and environment constraints. In: Intelligent Autonomous Systems, An International Conference, Amsterdam, The Netherlands (December 1986)
16. LaValle, S.M., Branicky, M.S., Lindemann, S.R.: On the relationship between classical grid search and probabilistic roadmaps. International Journal of Robotics Research 23(7/8), 673–692 (2004)
17. LaValle, S.M., Kuffner, J.J.: Randomized kinodynamic planning. International Journal of Robotics Research 20(5), 378–400 (2001)
18. Lumelsky, V., Stepanov, A.: Automaton moving admist unknown obstacles of arbitrary shape. Algorithmica 2, 403–430 (1987)
19. Marder-Eppstein, E., Berger, E., Foote, T., Gerkey, B., Konolige, K.: The office marathon: Robust navigation in an indoor office environment. In: Proc. International Conference on Robotics and Automation (May 2010)
20. Munkres, J.R.: Topology. Prentice Hall, Upper Saddle River (2000)
21. Niederreiter, H.: Random Number Generation and Quasi-Monte-Carlo Methods. Society for Industrial Mathematics, Philadelphia (1992)
22. Reif, J., Wang, H.: The complexity of the two dimensional curvature-constrained shortest-path problem. In: Third International Workshop on Algorithmic Foundations of Robotics, pp. 49–57 (June 1998)
23. Sampl, P.: Medial axis construction in three dimensions and its application to mesh generation. Engineering with Computers 17(3), 234–248 (2001)
24. Sánchez, G., Latombe, J.-C.: On delaying collision checking in PRM planning: Application to multi-robot coordination. International Journal of Robotics Research 21(1), 5–26 (2002)

25. Schmitzberger, E., Bouchet, J.L., Dufaut, M., Wolf, D., Husson, R.: Capture of homotopy classes with probabilistic road map. In: Proc. International Conference on Intelligent Robots and Systems (October 2002)
26. Vendittelli, M., Laumond, J.P., Nissoux, C.: Obstacle distance for car-like robots. IEEE Transactions on Robotics and Automation 15, 678–691 (1999)
27. Yap, C.K.: An $O(n \log n)$ algorithm for the Voronoi diagram of a set of simple curve segments. Discrete & Computational Geometry 2, 365–393 (1987)

Using Lie Group Symmetries for Fast Corrective Motion Planning

Konstantin Seiler, Surya P.N. Singh, and Hugh Durrant-Whyte

Abstract. For a mechanical system it often arises that its planned motion will need to be corrected either to refine an approximate plan or to deal with disturbances. This paper develops an algorithmic framework allowing for fast and elegant path correction for nonholonomic underactuated systems with Lie group symmetries, which operates without the explicit need for control strategies. These systems occur frequently in robotics, particularly in locomotion, be it ground, underwater, airborne, or surgical domains. Instead of reintegrating an entire trajectory, the method alters small segments of an initial trajectory in a consistent way so as to transform it via symmetry operations. This approach is demonstrated for the cases of a kinematic car and for flexible bevel tip needle steering, showing a prudent and simple, yet computationally tractable, trajectory correction.

1 Introduction

In practice, mechanical systems drift. Be it due to unexpected disturbances or in order to refine a coarse plan, corrective motion planning seeks to efficiently adapt a given trajectory in an elegant way. This is of particular interest in the agile control of underactuated nonholonomic systems. The nature of these systems is that certain degrees of freedom can only be controlled in a coupled manner (if at all). This makes it computationally hard to determine simple and valid trajectories [3, 11], thus it is preferable to efficiently adapt a given trajectory in an elegant way without having to start anew. Even in cases where explicit control laws are available, pure pursuit tracking is likely to produce unwanted artefacts due to its myopic nature [5,13]. Taking a larger horizon into account increases algorithmic and computational complexity, but enables alterations to the path in an elegant way. An example of such corrections is illustrated in Fig. 1.

Konstantin Seiler · Surya P.N. Singh · Hugh Durrant-Whyte
Australian Centre for Field Robotics, University of Sydney
e-mail: {k.seiler,spns,hugh}@acfr.usyd.edu.au

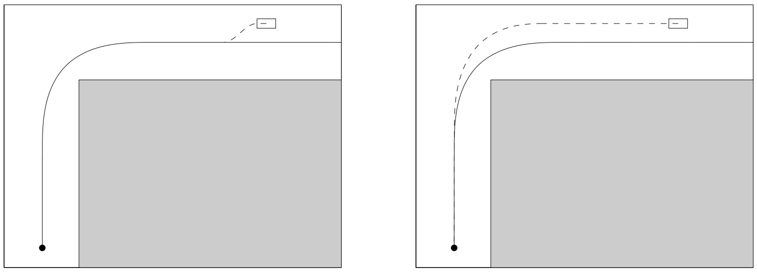


Fig. 1 A car (*small rectangle*) is following a previously planned path (*solid line*) to the goal (*dot*), but got off track due to disturbances. The left image shows a pure pursuit controller trying to get back on track as quickly as possible, resulting in unnecessary turns (*dashed line*). A more natural solution is shown in the second picture where the available space is used to elegantly correct the path during the upcoming turn.

Mechanical systems frequently exhibit symmetries that can be represented as Lie groups of translation or rotation [8, 14, 17]. Exploiting this can allow for elegant trajectory corrections in a computationally tractable way. This is valuable as the degrees of freedom represented within this symmetry group are often the ones that are only indirectly modifiable, and thus hard to control. For example, for most vehicles (be it submersible, ground, or airborne) properties such as thrust, speed and turning rate can be easily influenced; whereas, the position and heading are hard to control. As the latter often exhibits aforementioned symmetries, these methods allow for efficient planning and control of this subset.

Towards this, an algorithmic framework is introduced that allows for elegant planning and control systems that exhibit symmetries but are hard to control due to nonholonomic constraints. The method works without prior knowledge of control strategies specific to the system at hand. Further, it can be used either as an aid within an existing planning technique such as *rapidly exploring random tree* (RRT) or *probabilistic roadmap* (PRM) algorithms [11]; or, as presented here, on its own in order to adapt an existing trajectory and partly replace a classical controller.

This approach generalises on the use of Lie group actions for gap reduction during RRT planning. Cheng [4], for example, introduced a method to insert *coasting trajectories* into an existing trajectory in order to reduce gaps that arise during sampling based planning. That approach is likely to perform well for twisted paths but it comes short for less twisted ones as there is no possibility to shorten any part of an initial trajectory to recover from overshooting. The algorithm presented overcomes this problem by actually altering existing segments of the initial trajectory in a consistent manner.

The following discussion is framed on the assumption that an initial path has been obtained, but needs to be corrected as it does not reach the desired goal. Such corrections might be necessary due to gaps arising from sampling based planners, dynamic changes in the environment, or due to disturbances. Furthermore the algorithm is designed under the assumption that the initial path is up to some degree surrounded by free space, but may still contain narrow doorway situations. For ease of presentation, this work concentrates on altering degrees of freedom represented by aforementioned symmetry groups. It is understood that the remaining degrees are dealt with via a classical planning or control methods [11].

The remainder of this paper is structured as follows. Section 2 develops the mathematical model and introduces the basic concepts for trajectory alteration. Section 3 presents the algorithmic framework for situations without obstacles and shows results of its application to the (kinematic) car tracking and flexible needle steering problems. This is extended to the cases with obstacles in Section 4. Finally, Section 5 summarises the ideas presented and discusses future applications.

2 Mathematical Model

2.1 Basic Definitions

Kinodynamics can be defined on a state space X, which itself is a differentiable manifold with a metric [11]. The subset $X_{\text{obs}} \subseteq X$ denotes the states that have obstacles, and its complement $X_{\text{free}} := X \setminus X_{\text{obs}}$ is the viable free space. For clarity of presentation, an obstacle free setting ($X_{\text{obs}} = \emptyset$) is assumed (cases with obstacles will be tackled in Section 4). The space $U \subseteq \mathbb{R}^n$ represents the the system's control inputs. System progress is modelled via a set of ordinary differential equations (ODE)

$$\dot{x} = F(x,u) \tag{1}$$

for $x \in X$ and $u \in U$.

It is assumed that the reader is familiar with the notion of Lie groups. An introduction to the topic can be found, for example, in [12,1]. Let G be a Lie group acting on X such that $F(\,.\,,u)$ are left-invariant vector fields under the action of G. That is, there exists a multiplication law for elements $g \in G$ and $x \in X$, such that $gx \in X$, and for every trajectory $x(t) : I \subseteq \mathbb{R} \to X$ and control input $u(t) : I \to U$ fulfilling Eq. (1), the product $gx(t)$ also fulfils (1) for the same $u(t)$. This setting often allows for a decomposition of the state space X in the form

$$X = Z \times G$$

where the manifold Z is the *base space* and the Lie group G is denoted the *fibre component*.[1] The projections from X onto its components Z and G are denoted π_Z and π_G respectively. Common examples of such invariantly acting Lie groups arising from the system's symmetry group, are translations ($\mathbb{R}^n$), rotations (SO(2), SO(3)) or combinations thereof (SE(2), SE(3), $\mathbb{R}^3 \times$ SO(2), ...).

[1] For the decomposition to exist, the Lie group's action has to be *free*. That is, for all $x \in X$ and $g,h \in G$ it has to be true that $gx = hx$ implies $g = h$. If G is a symmetry group of the system, this is usually the case.

Applying this framework to the example of a kinematic car yields a state space X containing five dimensions, denoted speed v, turning rate ω, position x and y, and heading θ. The control inputs U contain two dimensions, acceleration a and change in turning rate ρ. The equations of motion are

$$\begin{aligned}\dot{v} &= a\\ \dot{\omega} &= \rho\\ \dot{x} &= \cos(\theta)v\\ \dot{y} &= \sin(\theta)v\\ \dot{\theta} &= \omega v\,.\end{aligned}$$

Since the car's behaviour is independent of position and heading in the sense that, if a valid path is translated or rotated, the resulting path is equally valid, as illustrated in Fig. 2, these dimensions form a symmetry Lie group to the system. Thus G should be set to be the group of Euler transformations, SE(2), representing x, y and θ. The remaining base space Z is spanned by v and ω. Thus

$$X = \mathbb{R}^2 \times \mathrm{SE}(2)\,.$$

Introducing some notation simplifies matters. Let $I \subseteq \mathbb{R}$ be a closed finite interval. Then I^- and I^+ denote the lower and upper boundary values respectively, such that

$$I = [I^-, I^+]\,.$$

A *time dependent control input* is considered to be a function $u : I_u \to U$ that maps from a closed finite interval $I_u \subseteq \mathbb{R}$ into the control space U. Integrating such a control input over time via the ODE (1) gives rise to a path in state space X that is dependent on an initial state x_0 and time t. Such integrated paths can be written as functions $\Phi_u(x_0,t) : X \times I_u \to X$ with the properties

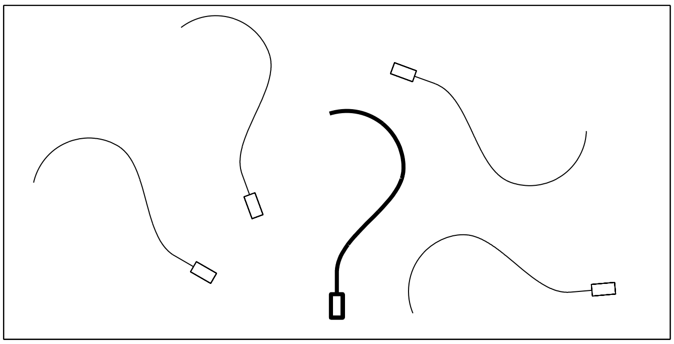


Fig. 2 The car (*rectangle*) has a valid initial state and path depicted in *bold*. It follows that the translated and rotated initial states and paths are equally valid.

$$\dot{\Phi}_u(x_0,t) = F(\Phi_u(x_0,t),u(t))$$

and

$$\Phi(x_0,I_u^-) = x_0 \ .$$

Given two time dependent control inputs $u : I_u \rightarrow U$ and $v : I_v \rightarrow U$ with $I_u^+ = I_v^-$, $u * v : [I_u^-, I_v^+] \rightarrow X$ is defined as the concatenation of the two functions u and v such that

$$(u*v)(t) = \begin{cases} u(t), & \text{if } t \in [I_u^-, I_u^+) \\ v(t), & \text{otherwise.} \end{cases}$$

This notation may also be used in cases where $I_u^+ \neq I_v^-$. In these cases a suitable reparameterisation of I_v is performed implicitly. Note that when using this notation for two integrated paths in state space X, the concatenation results in a single continuous path if and only if the final state of the first path coincides with the initial state of the second path. When this is the case, the resulting path is equivalent to integrating the concatenated control inputs directly, thus

$$\Phi_{u*v}(x_0,t) = \Phi_u(x_0,t) * \Phi_v(\Phi_u(x_0,I_u^+),t) \ .$$

2.2 Trajectory Transformations

It is hard to find a solution for the planning problem of connecting two predefined points x_{start} and x_{goal} in X [3, 10]. In the general case, this leads to running a search over all time varying control inputs. As the space of all possible control inputs can be too big to search exhaustively, many algorithms focus on relatively small subsets and either run a search over a discrete path set [6] or run a non-linear optimisation algorithm or search over a continuous path set [7,9]. The former, by its very nature, can only reach a discrete subset of X, where as the latter typically involves reintegrating the whole trajectory $\Phi_u(x_{\text{start}},t)$ in each step of the optimisation process.

Using operations given by a Lie group to transform a valid trajectory allows for the reuse of large parts of a previously calculated $\Phi_u(x_{\text{start}},t)$ as long as changes to the trajectory happen in a compatible way. Thus searching a continuum can be done without complete reintegration.

Let u and v be two time dependent control inputs that differ in some region, but coincide otherwise. They can be split up as

$$u = u_1 * u_2 * u_3$$

and

$$v = u_1 * v_2 * u_3$$

where u_1 and u_3 represent the parts that are common to both. Note that the lengths of the middle segments I_{u_2} and I_{v_2} do not necessarily have to be equal. Starting both trajectories at a common initial state $x_0 \in X$ yields

$$\Phi_u(x_0,t) = \Phi_v(x_0,t) \quad \text{for } t \in I_{u_1}.$$

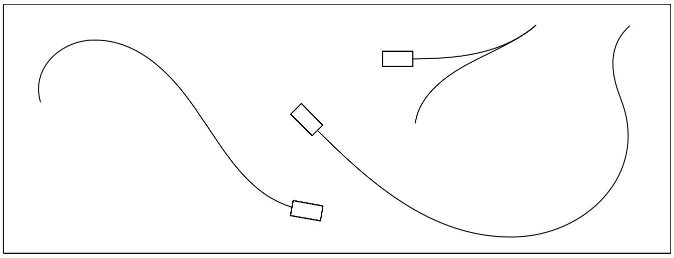


Fig. 3 Three trajectories for a car (*rectangle*), all resulting from the same control input. The behaviour is sensitive to initial conditions (speed and turning rate), causing different trajectories.

In general, equality of the third part of the control inputs, u_3, can not be used, as the final states of the middle segments, $\Phi_u(x_0, I_{u_2}^+)$ and $\Phi_u(x_0, I_{v_2}^+)$, need not coincide. Using different states as initial states for the third part of the path can result in a variety of different trajectories as illustrated in Fig. 3. If however it is assumed that the final states of the middle segments u_2 and v_2 only differ on the fibre component G but coincide on the base space Z, the similarity of the third parts of the trajectory can be exploited. Having equality on the base space as in

$$\pi_Z(\Phi_u(x_0, I_{u_2}^+)) = \pi_Z(\Phi_v(x_0, I_{v_2}^+)) \tag{2}$$

implies there exists a transformation $g \in G$ such that

$$\Phi_v(x_0, I_{v_2}^+) = g\Phi_u(x_0, I_{u_2}^+) \; . \tag{3}$$

In the case of the kinematic car, Eq. (2) can be interpreted as having identical speed v and turning rate ω. Then Eq. (3) yields the translation and rotation necessary to transform one state into the other. Because the equations of motion are invariant under translation and rotation, the resulting third parts of the paths will be translated and rotated versions of each other as illustrated in Fig. 4. In the general case, the same line of reasoning on invariance yields

$$\Phi_{u_3}(\Phi_v(x_0, I_{v_2}^+), t) = \Phi_{u_3}(g\Phi_u(x_0, I_{u_2}^+), t) = g\Phi_{u_3}(\Phi_u(x_0, I_{u_2}^+), t) \; . \tag{4}$$

Looking at this result from a viewpoint of computational complexity, Eq. (4) can save calculation time. Given Φ_u, the computational cost of Φ_v is mainly the cost of integrating the second segment given by v_2. The third segment defined by u_3 can be calculated directly by the use of group operations. In particular, during non-linear optimisation, the final state $\Phi_v(x_0, I_v^+)$ is typically the only one of interest. Thus, there is no need to actually transform the whole third segment of the path. Instead one can determine the trajectory's final state directly. As a result, the cost for $\Phi_v(x_0, I_v^+)$ is linear in the size of I_{v_2}.

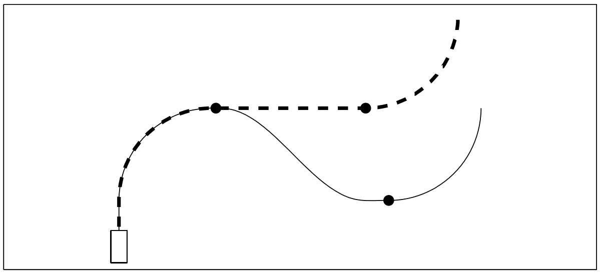


Fig. 4 A car (*rectangle*) follows two different paths resulting from control inputs that only differ on a region in the middle but are identical otherwise. The paths coincide up to the first marker (*dot*). After that, the paths differ. However at the respective second markers, speed and turning rate are identical for both paths and thus the remaining parts of the path are the same, just transformed.

2.3 Optimising a Trajectory

Given a time dependent control input u and a corresponding trajectory $\Phi_u(x_0,t)$, one might be able to find an alteration u^c that stretches (or compresses) the trajectory's behaviour on the base space Z over time. That is

$$\pi_Z(\Phi_{u^c}(x_0,t)) = \pi_Z(\Phi_u(x_0,ct)) \tag{5}$$

for a stretch factor $0 < c \in \mathbb{R}$. In particular, this yields identical final states on Z,

$$\pi_Z(\Phi_{u^c}(x_0,I^+_{u^c})) = \pi_Z(\Phi_u(x_0,I^+_u)) \; .$$

In the case of the car, for instance, this could map to reduced accelerator commands resulting in a longer distance travelled by the time the target speed is reached. While the stretching operation does not change the end result on the base space Z, it does alter the fibre G, thus emphasising or weakening features of the trajectory. For the car, the stretching operation can be calculated by dividing the control inputs a and ρ by c while multiplying the time they are applied by c.

Combining the results obtained so far, an efficient tool for altering a trajectory during a non-linear optimisation process can be built. Let $\Phi_u(x_0,t)$ be a trajectory given by a split control input

$$u = u_1 * \ldots * u_n$$

and starting point $x_0 \in X$. Changing a single u_i to $u_i^{c_i} =: v_i$ results in a Lie group operation $g_i \in G$ as of Eq. (3). Repeating this, one is able to alter several or even all segments of the path at once in order to get a new control input

$$v := v_1 * \ldots * v_n$$

where all v_i result from some $u_i^{c_i}$. In cases where $c_i = 1$, and thus the segment is unaltered, the corresponding g_i is set to the identity element $\mathbf{1} \in G$ without further calculation. Assuming $\Phi_u(x_0,t)$ is given and the changed segments $\Phi_{v_i}(\Phi_u(x_0,I^+_{u_{i-1}}),t)$ and transformations g_i have been calculated, the new trajectory $\Phi_v(x_0,t)$ is computed efficiently using group operations only. Iteratively applying (4) yields

$$\Phi_{v_i}(\Phi_v(x_0,I^+_{v_{i-1}}),t) = g_{i-1}\dots g_1 \Phi_{v_i}(\Phi_u(x_0,I^+_{u_{i-1}}),t)$$

and thus

$$\Phi_v(x_0,t) = \Phi_{v_1}(x_0,t) * g_1 \Phi_{v_2}(\Phi_{u_1}(x_0,I^+_{u_1}),t) * \dots * g_{n-1}\dots g_1 \Phi_{v_n}(\Phi_{u_{n-1}}(x_0,I^+_{u_{n-1}}),t)\,.$$

In particular, one can write the final state of the trajectory as

$$\Phi_v(x_0,I^+_v) = g_n \dots g_1 \Phi_u(x_0,I^+_u)\,. \tag{6}$$

Clearly not much is saved in cases where all segments of the trajectory have been changed (i.e., all $c_i \neq 1$). However, if only a small fraction of the control input has been altered, then it is only necessary to reintegrate the fibre component of those altered segments. Thus the computational cost for calculating the new trajectory, or directly its end point, is linear in the length of the changed segments plus the cost of a few additional group operations.

Note that it is possible to perform the calculations of g_i and $\Phi_{v_i}(\Phi_u(x_0,I^+_{u_{i-1}}),t)$ separately for each segment, independent of what is done to other segments. Thus, for another transformation using some c'_i, all results where $c'_i = c_i$ can be reused. This speeds up things significantly for gradient calculations as will be detailed later and also allows for parallel computation.

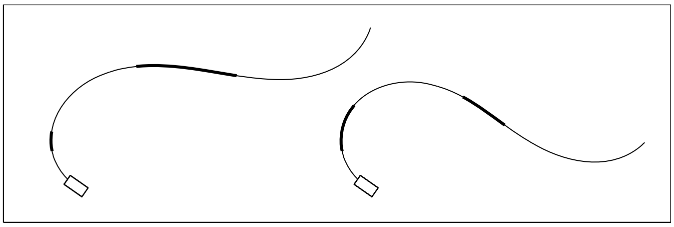


Fig. 5 A scaling operation has been applied to the *bold* segments of the *left hand* path to derive the *right hand* trajectory. Only the *bold* segments had to be reintegrated, the remainder is identical.

3 Path Correction Algorithm without Obstacles

For path correction, it will be assumed that an initial path $\Phi_u(x_0,t)$ as well as its control input u and initial state x_0 have been given. Furthermore, the path's final state $\Phi_u(x_0,I^+_u)$ does not coincide with the goal x_{goal}, but is somewhat in the vicinity of it. The objective is to alter the trajectory Φ_u in such a way that its final state matches x_{goal}. It will be assumed that the correction needs to be done in the fibre component only and that there are no obstacles present. This will be achieved in two

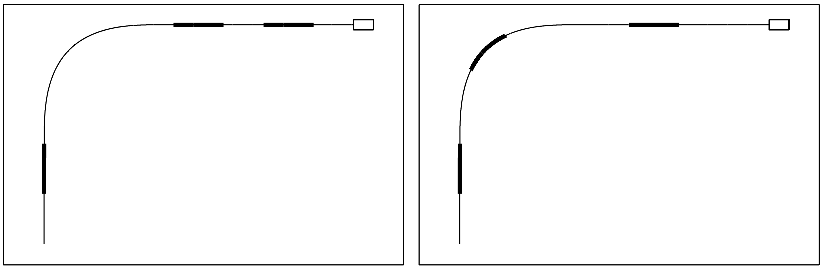


Fig. 6 The segments (*bold*) chosen for the trajectory on the *left* are unable to span the space well as they enable moving the final state horizontally and vertically, but prohibit alteration to the car's heading. The selection shown on the *right* is superior because changes in all directions including heading are possible.

steps: (1) a small and suitable set of path segments will be selected for stretching operations; (2) matching stretching factors c_i will be determined for said segments.

When selecting path segments, it is advantageous to select exactly as many segments as there are dimensions in the Lie group G. Using less segments results in too few degrees of freedom when altering the trajectory and thus failure to span a whole neighbourhood of the final state $\Phi_u(x_0, I^+)$. Using more segments than $\dim G$ leads to undesired behaviour as the solution is no longer unique. Furthermore, segments are chosen in such a way that the directions they move the trajectory's final state into have the potential to span the space well as illustrated in Fig. 6. This can be formalised by considering the derivatives

$$\frac{\partial \Phi_v(x_0, I_v^+)}{\partial c_i} = \frac{\partial g_i \Phi_u(x_0, I_u^+)}{\partial c_i}$$

evaluated at $c_i = 1$. As above, v represents the control input u with some segments u_i replaced by their scaled versions $u_i^{c_i}$ and, again, $g_i \in G$ denotes the resulting Lie group transformation. The quality of a selection of $\dim G$ segments can then be measured by analysing the condition of the resulting Jacobian

$$J = \frac{\partial \Phi_v(x_0, I_v^+)}{\partial (c_1, \ldots, c_{\dim G})} = \left(\frac{\partial g_1 \Phi_u(x_0, I_u^+)}{\partial c_1}, \ldots, \frac{\partial g_{\dim G} \Phi_u(x_0, I_u^+)}{\partial c_{\dim G}} \right) \tag{7}$$

evaluated at $c_i = 1$ for all i. The derivatives in the matrix on the right hand side are written as column vectors. If the matrix's condition is small, it has the potential to span the space well.

Since each column of J in Eq. (7) is independent of the remaining segments, the derivative has to be calculated only once. Thus, in practise, a solution is to select a larger set of non-overlapping segments and out of that then randomly draw selections of $\dim G$ elements for further testing. The selection with the smallest condition of the resulting Jacobian is then chosen. An exhaustive search for the optimal selection is not necessary since it is sufficient to remove poor candidates. Taking a few random samples is often enough.

As optimisation algorithms typically work by minimising a target function [2], here the distance of the path's final state to the goal, it might seem tempting not to use the condition of the final state's Jacobian as presented here, but instead estimate the convergence rate of that target function directly via its second order approximation and the eigenvalues of the Hessian [16, 4]. In tests however this proved to perform poorly.

Once a set of segments is chosen, the values for the c_i need to be determined in order to actually improve the trajectory. Therefore a target function $f(c_1,\dots,c_{\dim G})$ is defined as the distance between $\Phi_v(x_0, I_v^+)$ and x_{goal}. It is then minimised using a Conjugate Gradient method [16, 2]. Estimating the gradient of f at $(c_1,\dots,c_{\dim G})$ is done by taking into account the function value $f(c_1,\dots,c_{\dim G})$, as well as those resulting from going a small step into each direction, $f(c_1,\dots,c_i+\varepsilon,\dots,c_{\dim G})$, naively resulting in $\dim G + 1$ integrations for each segment. However, since only two distinct values, c_i and $c_i+\varepsilon$, are used for each dimension of G, the calculated g_i can be recombined to obtain all function evaluations necessary. Thus, the cost to estimate a gradient is two integrations per segment plus some group operations.

Pseudocode for this algorithm is presented in Algorithm 1. It was used for the path depicted on the right hand side of Fig. 1 as well as the example presented in Fig. 7. Implementing this algorithm for more complex 3D cases, such as bevel tip needle steering, the state space X consists of eight dimensions: Insertion speed v, turning rate ω as well as six degrees of freedom representing position and orientation in three space. Thus the base space Z represents v and ω whereas G equals the group of Euler transformations SE(3). Following previous notation in this domain [15, 17], SE(3) is represented using homogeneous 4×4 matrices $g = \left(\begin{smallmatrix} R & \mathbf{t} \\ \mathbf{0} & 1\end{smallmatrix}\right)$ where $R \in \mathrm{SO}(3)$ is a rotation matrix and $\mathbf{t} \in \mathbb{R}^3$ represents translation. The control

Algorithm 1. Path correction algorithm without obstacles

```
$u \leftarrow$ current plan
$M \leftarrow$ select set of at least $\dim G$ non overlapping segments of $I_u$
$C_{\min} \leftarrow \infty$
for $i = 1$ to $\min(maxSelections,$ number of selections possible$)$ do
    $S \leftarrow$ draw new selection of $\dim G$ random elements of $M$
    $J \leftarrow \left(\frac{\partial g_1 \Phi_u(x_0, I_u^+)}{\partial c_1}, \dots, \frac{\partial g_{\dim G} \Phi_u(x_0, I_u^+)}{\partial c_{\dim G}}\right)$   {calculated for the segments stored in $S$}
    if $\mathrm{cond}(J) < C_{\min}$ then
        $C_{\min} \leftarrow \mathrm{cond}(J)$
        $S_{\min} \leftarrow S$
    end if
end for
optimise $c_1,\dots,c_{\dim G}$
    $v \leftarrow$ scale the segments of $u$ stored in $S_{\min}$ according to values of $c_1,\dots,c_{\dim G}$
until $\mathrm{dist}\left(x_{\text{goal}}, \Phi_v(x_0, I_v^+)\right)$ minimal
return $v$
```

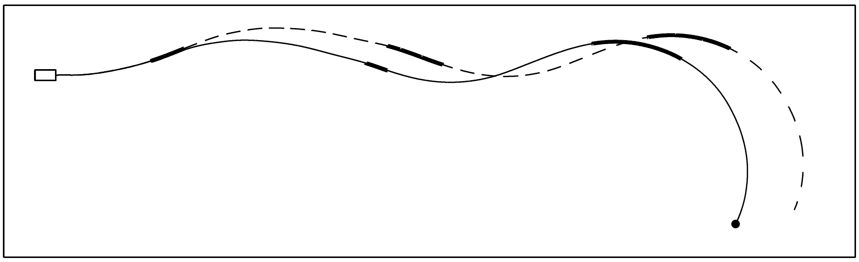


Fig. 7 A car (*rectangle*) is trying to reach the goal (*dot*). The *dashed* line shows the initial path that fails to reach the goal. By altering the segments depicted in *bold*, the *solid* path is created.

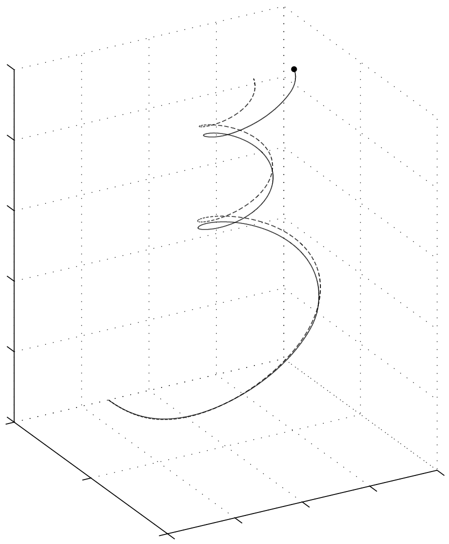


Fig. 8 Path correction for a needle steering case. The needle needs to reach the goal (*dot*), but the initial plan, depicted by the *dashed* line, misses it. The solid line is the correction made by the path correction algorithm.

space U has two dimensions, acceleration $a = \dot{v}$ and change in turning rate $\rho = \dot{\omega}$. The remaining equations of motion are given by

$$g^{-1}\dot{g} = \begin{pmatrix} 0 & -\omega & 0 & 0 \\ \omega & 0 & -\kappa v & 0 \\ 0 & \kappa v & 0 & v \\ 0 & 0 & 0 & 0 \end{pmatrix} \in \mathfrak{se}(3)$$

where the constant κ is the curvature of the needle's trajectory and $\mathfrak{se}(3)$ is the Lie algebra of SE(3). An example for path correction using this system is presented in Fig. 8.

Since the algorithm works by enlarging or shrinking certain sections of the trajectory, it can not perform well in cases where the trajectory has too few features. Especially in cases where the path consists only of a straight line or a section of a circle, it is impossible to find segments that span the space well in a way discussed previously and illustrated in Fig. 6.

4 Path Correction Algorithm with Obstacles

When dealing with obstacles, an inversion of perspective is helpful. Up to now the initial path was considered to start at the robot's current state and the final state was then optimised. However, it is equally valid to anchor the initial path at the goal state x_{goal} and consider the robot to be at a state x_{curr} that does not coincide with the path's initial state $x_{\text{start}} = \Phi_u(x_{\text{start}}, I_u^-)$.

It will be assumed that the initial path is a collision free trajectory $\Phi_u(x_{\text{start}}, t)$ with $\Phi_u(x_{\text{start}}, I_u^+) = x_{\text{goal}}$. X_{obs} does not have to be empty, but it is assumed that there is

Algorithm 2. Path correction algorithm with obstacles

```
$u \leftarrow$ original plan
$v \leftarrow$ current plan
$x_{\text{curr}} \leftarrow$ system's current state
$S \leftarrow$ empty stack
**repeat**
    **if** $\Phi_v(x_{\text{curr}}, t)$ in collision **then**
        $v_1, v_2 \leftarrow v$ split at point of first collision
    **else**
        $v_1, v_2 \leftarrow v, \emptyset$
    **end if**
    $x_{\text{new}} \leftarrow$ find intermediate goal using $u$
    $v_1 \leftarrow$ run *algorithm without obstacles* (Alg. 1) for $\Phi_{v_1}(x_{\text{curr}}, t)$ in order to reach $x_{\text{new}}$
    **if** $\Phi_{v_1}(x_{\text{curr}}, t)$ collision free **then**
        $S.push \leftarrow v_1, x_{\text{curr}}$
        $v, x_{\text{curr}} \leftarrow v_2, \Phi_{v_1}(x_{\text{curr}}, I_{v_1}^+)$
    **else if** $\Phi_{v_1}(x_{\text{curr}}, I_{v_1}^+)$ collision free **then**
        $v \leftarrow v_1 * v_2$
    **else if** $S$ not empty **then**
        $v_0, x_{\text{curr}} \leftarrow S.pop$
        $v \leftarrow v_0 * v_1 * v_2$
    **else**
        **return** FAIL
    **end if**
**until** $v = \emptyset$
**while** $S$ not empty **do**
    $v_0, x_{\text{curr}} \leftarrow S.pop$
    $v \leftarrow v_0 * v$
**end while**
**return** $v$
```

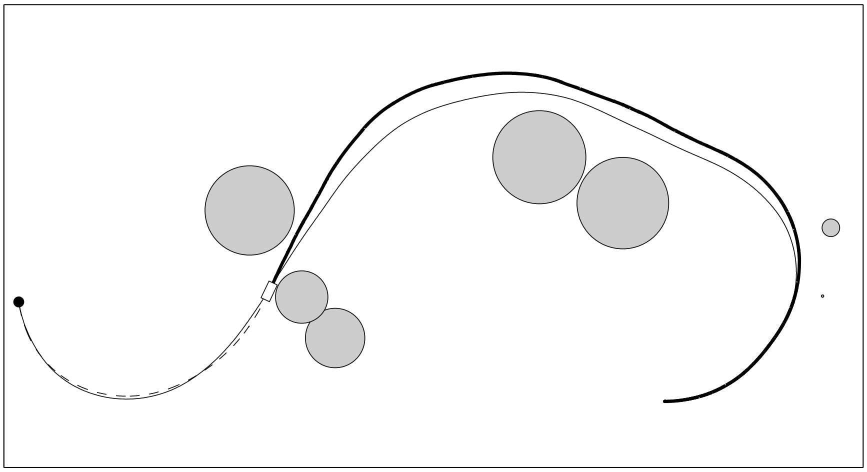


Fig. 9 A car tracking a path (*thin solid line*) through terrain with obstacles to reach the goal (*dot*). While driving, base space and control inputs (v, ω, a and ρ) are disturbed by random errors in form of a Wiener process. Errors on the base space are corrected by use of a feedforward controller with a saturation function, while resulting errors in position and heading are corrected repeatedly using the path correcting algorithm. The actual path taken is depicted by the *bold solid line* and the currently planned path that is to be followed is shown by the *dashed line*.

a certain amount of free space surrounding the trajectory most of the time that can be used for corrective actions. The robot is currently at x_{curr} and is following a path defined by v that is derived from, but that might not coincide with u. So $\Phi_v(x_{\mathrm{curr}}, t)$ does not necessarily reach the goal x_{goal}. Allowing a discrepancy between u and v is advantageous when making multiple corrections; for example, when repeated online calculations are performed while executing a path under disturbances. In this setting, the initial path u will be kept constant during the whole process, while alterations are made to v only.

If no collisions occur in $\Phi_v(x_{\mathrm{curr}}, t)$, alterations to v can be made directly using the *path correction algorithm without obstacles* (Algorithm 1). Otherwise, in case of collisions, the first point of collision is found as

$$t_{\mathrm{col}} := \min\{t \in I_v \mid \Phi_v(x_{\mathrm{curr}}, t) \in X_{\mathrm{obs}}\}$$

and the path can be split such that $v = v_1 * v_2$ and $t_{\mathrm{col}} = I_{v_1}^{+}$. To get around the obstacle, it is necessary to correct the path in two steps, first $\Phi_{v_1}(x_{\mathrm{curr}}, t)$ using a new intermediate goal $x_{\mathrm{new}} \in X_{\mathrm{free}}$ and then v_2.

To define x_{new}, the colliding point $\Phi_{v_1}(x_{\mathrm{curr}}, I_{v_1}^{+})$ is pulled towards the corresponding point x_{orig} of $\Phi_u(x_{\mathrm{start}}, t)$ where, due to scaling operations performed on v, the

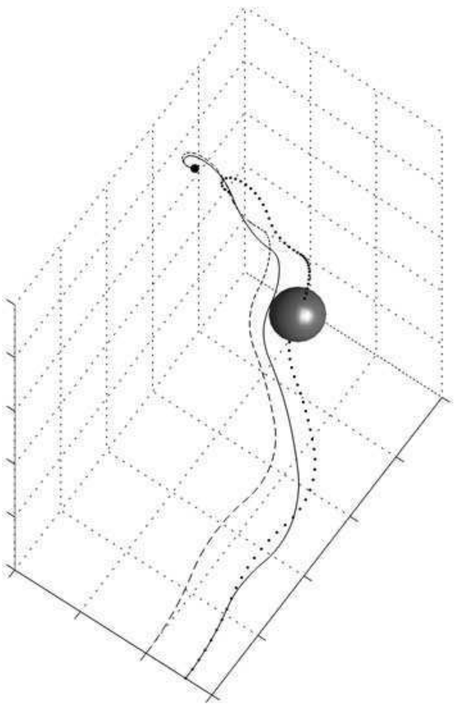


Fig. 10 A needle has to reach a goal (*dot*) and has an initial path depicted by the *dashed line*. Due to an offset in the initial position, the planned trajectory would result in the *dotted line*, colliding with an obstacle and missing the goal. The path is corrected and a valid path depicted by the *solid line* is created.

time t is not necessarily identical for u and v any more. This is done by parameterising a *straight line* connecting x_{orig} with $\Phi_{v_1}(x_{\text{curr}}, I_{v_1}^+)$ by $s : [0,1] \to X$ such that $s(0) = x_{\text{orig}}$ and $s(1) = \Phi_{v_1}(x_{\text{curr}}, I_{v_1}^+)$.[2] Using s, the intermediate goal is defined as

$$x_{\text{new}} := s(\alpha^n) ,$$

where n is the smallest $n \in \mathbb{N}$ such that $s(\alpha^n) \in X_{\text{free}}$ and $\alpha \in [0,1)$. The convergence rate α determines how fast the trajectory should be pulled back towards the original path and away from the obstacle. A large α results in staying closer to the obstacle

[2] Note that, as states already coincide on the base, the line s only has to be defined in G and is constant in Z. In most cases, it is intuitive what a suitable choice for a straight line within G should be and how it can be implemented easily. In less obvious cases, the exponential map $\exp : \mathfrak{g} \to G$ can be used, where $\mathfrak{g}$ is the Lie algebra of G. Let $d = \pi_G(x_{\text{orig}})^{-1}\pi_G(\Phi_{v_1}(x_{\text{curr}}, I_{v_1}^+))$ be the difference between the two states to connect. The points have to be close enough such that d lies within the identity component of G (i.e. the image of exp), as otherwise an easy connection is not possible. Then the line can be defined as $s(t) := \pi_Z(x_{\text{orig}}) \exp(t \exp^{-1}(d))$ for a suitable pre-image $\exp^{-1}(d)$.

and thus in a higher change that future corrections are necessary; a smaller α on the other hand pulls the trajectory more aggressively towards the original trajectory, preventing effective use of the available free space. The presented implementation uses $\alpha = 1/2$.

Using x_{new}, the *path correction algorithm without obstacles* is run over v_1 to get a new plan w_1 and thus $w = w_1 * v_2$. Note that it is acceptable if the new trajectory does not actually reach x_{new} itself as the main purpose of the operation is to pull the path away from the obstacle. What has to be considered tough are collisions of $\Phi_{w_1}(x_{\text{curr}}, t)$. If $\Phi_{w_1}(x_{\text{curr}}, t)$ is collision free, the process continues by recursively applying the *path correction algorithm with obstacles* on v_2. If $\Phi_{w_1}(x_{\text{curr}}, t)$ is in collision, two cases have to be considered. If the final state $\Phi_{w_1}(x_{\text{curr}}, I^+_{w_1})$ is in collision, the optimisation run did not get close enough to x_{new} because the path given by v_1 was too short or too featureless for the algorithm to perform well. In cases where v_1 is not the first part of the path due to a recursive call, w (and thus v_1) can be extended and a new attempt can be made. Otherwise the system is too close to an obstacle for suitable correction, and the algorithm is considered failed. If the final state $\Phi_{w_1}(x_{\text{curr}}, I^+_{w_1})$ is in X_{free}, the optimisation run was successful but the alteration introduced a new collision. Then a recursive call on the altered plan w and the starting point x_{curr} is necessary to get rid of the newly introduced collision. Pseudocode for the complete framework is given in Algorithm 2.

The algorithm can be used either offline within another planning framework (e.g. PRM, RRT) [4,11] or online while tracking a previously planned path. When applied in the latter approach, it is important that the initial trajectory u is kept unaltered during the whole process. Examples of how the algorithm performs are presented in Figs. 9 and 10.

5 Conclusion

An algorithmic framework was presented that allows for elegant and fast path correction while preserving the character of the initial trajectory, thus eliminating the need for expensive re-planning from scratch. The algorithm has been implemented for a kinematic car as well as for needle steering and simulations for the system's behaviour under disturbances have been performed. Future work will include implementing the system on experimental field systems currently under development.

Acknowledgements. This work is supported by the Rio Tinto Centre for Mine Automation and the ARC Centre of Excellence program funded by the Australian Research Council (ARC) and the New South Wales State Government.

References

1. Bloch, A.M.: Nonholonomic Mechanics and Control. In: Interdisciplinary Applied Mathematics, vol. 24. Springer, New York (2003)
2. Byrd, R.H., Nocedal, J., Waltz, R.A.: Knitro: An integrated package for nonlinear optimization. In: Large Scale Nonlinear Optimization, pp. 35–59. Springer, Heidelberg (2006)

3. Canny, J.F.: The complexity of robot motion planning. MIT Press, Cambridge (1988)
4. Cheng, P., Frazzoli, E., LaValle, S.M.: Improving the performance of sampling-based motion planning with symmetry-based gap reduction. IEEE Transactions on Robotics 24(2), 488–494 (2008)
5. Kanayama, Y., Kimura, Y., Miyazaki, F., Noguchi, T.: A stable tracking control method for an autonomous robot. In: Proceedings of the IEEE International Conference on Robotics and Automation, vol. 1, pp. 384–389 (1990)
6. Kelly, A., Nagy, B.: Reactive nonholonomic trajectory generation via parametric optimal control. The International Journal of Robotics Research 22(7-8), 583–601 (2003)
7. Kobilarov, M., Desbrun, M., Marsden, J.E., Sukhatme, G.S.: A discrete geometric optimal control framework for systems with symmetries. In: Proceedings of Robotics: Science and Systems, Atlanta, GA, USA (2007)
8. Koon, W.S., Marsden, J.E.: Optimal control for holonomic and nonholonomic mechanical systems with symmetry and Lagrangian reduction. SIAM Journal on Control and Optimization 35(3), 901–929 (1995)
9. Lamiraux, F., Bonnafous, D., Lefebvre, O.: Reactive path deformation for nonholonomic mobile robots. IEEE Transactions on Robotics 20(6), 967–977 (2004)
10. Latombe, J.C.: Robot Motion Planning. Kluwer, Boston (1991)
11. LaValle, S.M.: Planning Algorithms. Cambridge University Press, Cambridge (2006)
12. Murray, R.M., Li, Z., Sastry, S.S.: A Mathematical Introduction to Robotic Manipulation. CRC Press, Boca Raton (1994)
13. Ollero, A., Heredia, G.: Stability analysis of mobile robot path tracking. In: Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 461–466 (1995)
14. Ostrowski, J.P.: Computing reduced equations for robotic systems with constraintsand symmetries. IEEE Transactions on Robotics and Automation 15(1), 111–123 (1999)
15. Park, W., Reed, K.B., Chirikjian, G.S.: Estimation of model parameters for steerable needles. In: IEEE International Conference on Robotics and Automation, pp. 3703–3708 (2010)
16. Shewchuk, J.R.: An introduction to the conjugate gradient method without the agonizing pain. Tech. rep., Carnegie Mellon University Pittsburgh (1994)
17. Webster III, R.J., Cowan, N.J., Chirikjian, G.S., Okamura, A.M.: Nonholonomic modeling of needle steering. In: Proc. 9th International Symposium on Experimental Robotics (2004)

Asynchronous Distributed Motion Planning with Safety Guarantees under Second-Order Dynamics

Devin K. Grady, Kostas E. Bekris, and Lydia E. Kavraki

Abstract. As robots become more versatile, they are increasingly found to operate together in the same environment where they must coordinate their motion in a distributed manner. Such operation does not present problems if the motion is quasi-static and collisions can be easily avoided. However, when the robots follow second-order dynamics, the problem becomes challenging even for a known environment. The setup in this work considers that each robot replans its own trajectory for the next replanning cycle. The planning process must guarantee the robot's safety by ensuring collision-free paths for the considered period and by not bringing the robot to states where collisions cannot be avoided in the future. This problem can be addressed through communication among the robots, but it becomes complicated when the replanning cycles of the different robots are not synchronized and the robots make planning decisions at different time instants. This paper shows how to guarantee the safe operation of multiple communicating second-order vehicles, whose replanning cycles do not coincide, through an asynchronous, distributed motion planning framework. The method is evaluated through simulations, where each robot is simulated on a different processor and communicates with its neighbors through message passing. The simulations confirm that the approach provides safety in scenarios with up to 48 robots with second-order dynamics in environments with obstacles, where collisions occur often without a safety framework.

1 Introduction

This paper considers multiple autonomous robots with non-trivial dynamics operating in a static environment. The robots try to reach their individual goals without

Devin K. Grady · Lydia E. Kavraki
Computer Science, Rice Univ., Houston, TX
e-mail: `{devin.grady,kavraki}@rice.edu`

Kostas E. Bekris
Computer Science and Engineering, Univ. of Nevada Reno, Reno, NV
e-mail: `bekris@cse.unr.edu`

collisions. Such scenarios are becoming increasingly interesting. For instance, consider the case of vehicles moving in a parking lot or going through a busy intersection, or unmanned aerial vehicles that carry out complex maneuvers. These examples involve second-order systems, which cannot stop instantaneously and must respect limits in the second-order derivatives of their state parameters. For such systems, collisions with other robots or obstacles cannot be easily avoided.

Real applications also require the solution of such problems in a decentralized manner. This work imposes a requirement for a decentralized solution and considers robots that replan their trajectories on the fly. Replanning allows robots to consider multiple alternative trajectories during each cycle and provides flexibility in changing environments. To coordinate the robots, this work utilizes communication. A planning algorithm makes use of information collected through communication to avoid collisions for the next cycle and ensure that robots reach states from where collisions can be avoided in the future. The duration of the cycle is the same for all robots, but the robots are not synchronized. Hence communication of plans can happen at any point and the robots need to operate safely in the presence of partial information about the plans of their neighbors. An asynchronous, distributed framework is developed that guarantees the safety of all robots in this setup.

Background. Safety issues for dynamical systems were first studied many years ago. Collision-free states that inevitably lead to collisions have been referred as Obstacle Shadows [24], Regions of Inevitable Collision [20] or Inevitable Collision States (`ICS`) [13]. A study on `ICS` resulted in conservative approximations [13] and generic `ICS` checkers [21]. It also provided *3 criteria for motion safety*: a robot must (i) consider its dynamics, (ii) the environment's future behavior, and (iii) reason over infinite-time horizon [12]. This line of research, however, did not deal with coordinating robots as the current paper does.

Reactive methods, such as the Dynamic Window Approach [11] and Velocity Obstacles [10], can enable a robot to avoid collisions for unknown on dynamic environments. Many existing reactive planners, however, do not satisfy the criteria for motion safety [12, 21]. Path deformation techniques compute a flexible path, adapted on the fly to avoid moving obstacles [18, 27], but do not deal with `ICS`. Reciprocal Velocity Obstacles (`RVOs`) [4] involve multiple agents which simultaneously avoid one another without communication but do not deal yet with `ICS`.

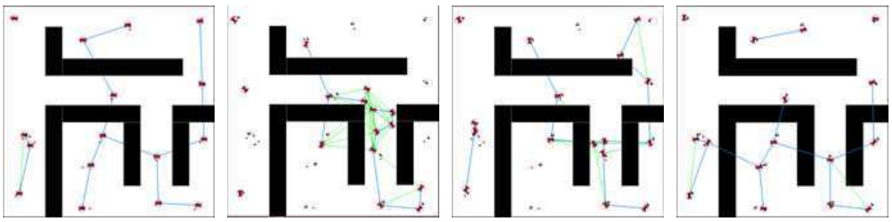


Fig. 1 A sample run in the office environment (left to right). Links show communicating robots.

A related control-based approach [17] deals with second-order models of a planar unicycle but does not provide guarantees in the presence of obstacles.

In contrast to reactive approaches, this paper focuses on *planning* safe paths. Planning has a longer horizon so it does not get stuck in minima as easily and extends to high degrees-of-freedom systems. Reasoning about safety during planning focuses the search on the safe part of the state space. In this work planning is achieved using a sampling-based tree planner [20, 15, 2]. Alternatives include, among others, navigation functions [8] and lattice-based approaches [23].

Braking maneuvers have been shown sufficient in providing safety in static environments [26] and have been combined with sampling-based replanning [5, 2]. For dynamic environments, relaxations of ICS are typically considered, such as τ-safety [14]. This notion guarantees no collision for τ seconds in the future for each node of a sampling-based tree. A sampling-based planner was tested on air-cushioned robots moving in dynamic environments, where an escape maneuver was computed when the planner failed to find a solution [15]. Learning-based approximations of ICS can also be found [16], as well as approximations of state$\times$time space obstacles [6]. Other works focus on the interaction between planning and sensing, and point out that planning must be limited within the robot's visibility region [1, 25]. The current paper extends the authors' earlier work [3], which integrated a sampling-based planner with ICS avoidance [2] to safely plan for multiple robots that formed a network and explored an unknown workspace. The previous work required a synchronous planning operation, which simplified coordination.

Planning for dynamic networks of robots has been approached by a combination of centralized and decoupled planning [7], without considering, however, the ICS challenge. Centralized planning does not scale and decoupled approaches, which may involve prioritization [9] or velocity tuning [22], are incomplete. The existing work follows a decoupled approach for performance purposes. In contrast to velocity tuning, it weakly constraints the robots' motion before considering interactions since it allows multiple alternative paths for each robot at each cycle. At the same time, it does not impose priorities but instead robots respect their neighbors in a way that emerges naturally from their asynchronous operation.

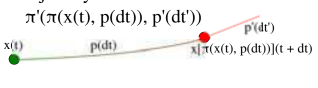
Contributions. This work extends the range of problems that can be solved efficiently with guarantees for ICS avoidance. The paper presents a general framework for independent but communicating second-order robots to reach their destinations in an otherwise known environment. The framework is fully distributed and relies on asynchronous interaction among the robots, where the robots' replanning cycles are not synchronized, the robots have no knowledge about their clock differences and no access to a global clock. It is based on the exchange of contingency plans between neighboring robots that are guaranteed to be collision-free. While contingency plans have been used in the past, this work emphasizes the importance of communicating contingencies in multi-robot scenarios and studies the asynchronous case. A proof that the proposed scheme guarantees ICS avoidance is provided. The framework has been implemented on a distributed simulator, where each robot is assigned to a different processor and message passing is used to convey plans. The experiments consider various scenarios involving 2 to 48 robots and demonstrate that safety is indeed achieved in scenarios where collisions are frequent if the `ICS` issue is ignored. The experiments also evaluate the efficiency and the scalability of the approach.

2 Problem Statement

Consider robots operating in the same known workspace with static obstacles. Each **robot** R^i exhibits drift and must satisfy non-holonomic constraints expressed by differential equations of the form: $\dot{x}^i = f^i(x^i, u^i)$, $g^i(x^i, \dot{x}^i) \leq 0$, where $x^i \in \mathcal{X}^i$ represents a **state**, u^i is a **control** and f^i, g^i are smooth. The subset of the **state space** $\mathcal{X}^i$ that does not cause a collision with static obstacles is denoted as $\mathcal{X}^i_f$. The robot model used in this paper can be found in Section 5 and involves acceleration controlled car-like systems, including versions with minimum positive velocity.

Each R^i is located at an initial state $x^i(0)$ and must compute plans that will bring it to its individual goal $x^i_g(t_{max})$ without collisions and within finite time t_{max}. Then:

- A **plan** is a sequence of controls $p(dt) = \{(u_1, dt_1), \ldots, (u_n, dt_n)\}$ $(dt = \sum_i dt_i)$.
- A plan $p(dt)$ executed at state $x(t)$ defines a **trajectory**: $\pi(x(t), p(dt))$, which is a sequence of states.
- A trajectory is **feasible** as long as it satisfies functions f^i and g^i for robot R^i.
- A plan $p(dt)$ is **valid** at state $x(t)$, if it defines a feasible trajectory $\pi(x(t), p(dt))$.
- A state along $\pi(x(t), p(dt))$ at time $t \ \in [t : t+dt]$ is denoted as $x[\pi(x(t), p(dt))](t \ )$.
- A feasible trajectory $\pi(x(t), p(dt))$ is **collision-free** with respect to the static obstacles if: $\forall\ t \ \in [t : t+dt] : \quad x[\pi(x(t), p(dt))](t \ ) \ \in \ \mathcal{X}_f$.
- For a **trajectory concatenation** (figure below) $\pi \ (\pi(x(t), p(dt)), p \ (dt \ ))$, plan $p(dt)$ is executed at $x(t)$ and then $p \ (dt \ )$ is executed at state: $x[\pi(x(t), p(dt))](t+dt)$.
- Two trajectories for R^i and R^j are **compatible:** $\pi^i(x^i(t^i), p(dt^i)) \asymp \pi^j(x^j(t^j), p^j(dt^j))$ as long as:
$$x[\pi^i](t) \asymp x[\pi^j](t) \qquad \forall\ \ t \ \in [max(t^i, t^j) : min(t^i + dt^i, t^j + dt^j)]$$
where $x^i \asymp x^j$ means that R^i in state x^i does not collide with R^j at state x^j. The corresponding plans $p(dt^i)$, $p(dt^j)$ are also called compatible at states $x^i(t^i)$, $x^j(t^j)$.


trajectory concatenation:
π'(π(x(t), p(dt)), p'(dt'))
x(t)
p(dt)
p'(dt')
x[π(x(t), p(dt))](t + dt)

The robots are equipped with an omnidirectional, range-limited communication tool, which is reliable and used for coordination and collision avoidance. The robots within range of R^i define the neighborhood N^i. A robot has information about other robots only if they communicate.

Given the above notation, the problem of **distributed motion planning with dynamics** (`DMPD`) can be defined as follows: Consider m robots with range-limited communication capabilities operating in the same workspace with obstacles. Each robot's motion is governed by second-order dynamics specified by f^i and g^i. Initially, robot R^i is located at state $x^i(0)$, where $x^i(0) \in \mathcal{X}^i_f$ and $\forall i, j : \ x^i(0) \asymp x^j(0)$. Each R^i must compute a valid plan $p^i(t_{max})$ so that:

- $x[\pi^i(x^i(0), p^i(t_{max}))](t_{max}) = x^i_g(t_{max})$ (i.e., the plans bring the robots to their individual goals within time t_{max}),
- $\forall\ i,\ \forall t \in [0 : t_{max}] :\ x[\pi^i(x^i(0), p^i(t_{max}))](t) \in \mathcal{X}_f$ (i.e., the resulting trajectories are collision-free with static obstacles)
- and $\forall\ i, j :\quad \pi^i(x^i(0), p^i(t_{max})) \asymp \pi^j(x^j(0), p^j(t_{max}))$ (i.e., the trajectories are pairwise compatible from the beginning and until all the robots reach their goals).

3 A Simple Framework without Safety Guarantees

This paper adopts a decentralized framework for scalability purposes. Each robot's operation is broken into intervals $([t^i_0 : t^i_1], [t^i_1, t^i_2],\ \ldots, [t^i_n : t^i_{n\ 1}], \ldots)$, called cycles. During $[t^i_{n\ 1} : t^i_n]$, robot R^i considers different plans Π^i for cycle $[t^i_n : t^i_{n\ 1}]$, given the future initial state $x^i(t^i_n)$. Through coordination, R^i selects plan $p^i\ ([t^i_n : t^i_{n\ 1}])$.

It is assumed that the duration of each cycle is constant and the same for all robots: $\forall i,\ \forall n : t^i_{n\ 1} - t^i_n = dt$. Nevertheless, the robots do not have a synchronous operation: the cycles among different robots do not coincide and t^i_0 is typically different than t^j_0. Synchronicity is a restrictive assumption, as it requires all the robots to initiate their operation at exactly the same time although they may be located in different parts of the world and may not communicate their initial states.

Given this setup, Algorithm 3.1 outlines a straightforward approach for the single cycle operation of each robot that tries to find compatible plans. During $[t^i_{n\ 1} : t^i_n]$, R^i computes alternative partial plans Π^i for the consecutive planning cycle. In parallel, R^i listens for messages from robots in neighborhood N^i. The messages contain the selected trajectories for each robot. When time approaches $t^i_n - \epsilon$, R^i selects among all trajectories that are collision-free and compatible with the neighbors' messages, the one that brings the robot closer to its goal. If such a trajectory is indeed found at each iteration, then the DMPD problem is eventually solved by this algorithm.

Algorithm 3.1. Simple but Unsafe Operation of R^i During Cycle $[t^i_{n\ 1} : t^i_n]$

```
$\Pi^i \leftarrow \emptyset$ and $\Pi^{N^i} \leftarrow \emptyset$
**while** $t < t^i_n - \epsilon$ **do**
    $\pi^i(x^i(t^i_n), p^i(t^i_n : t^i_{n+1})) \leftarrow$ collision-free trajectory from a single-robot planner
    $\Pi^i \leftarrow \Pi^i \cup \pi^i(x^i(t^i_n), p^i(t^i_n : t^i_{n+1}))$
    **if** $R^j \in N^i$ is transmitting a trajectory $\pi^j$ **then** $\Pi^{N^i} \leftarrow \Pi^{N^i} \cup \pi^j$
**for all** $\pi^i \in \Pi^i$ **do**
    **for all** $\pi^j \in \Pi^{N^i}$ **do**
        **if** $\pi^i \not\asymp \pi^j$ (incompatible trajectories) **then** $\Pi^i \leftarrow \Pi^i - \pi^i$
$\pi^i_* \leftarrow$ trajectory in $\Pi^i$ which brings $R^i$ closer to the goal given a metric
Transmit $\pi^i_*$ to all neighbors in $N^i$ and execute $\pi^i_*$ during next cycle
```

4 A Safe Solution to Distributed Motion Planning with Dynamics

A robot following the above approach might fail to find a trajectory π^i . This section describes a distributed algorithm that guarantees the existence of a collision-free, compatible trajectory for all robots at every cycle.

A. Safety Considerations - Inevitable Collision States: One reason for failure is when the single-robot planner fails to find collision-free paths. This is guaranteed to happen when $x^i(t_n^i)$ is an ICS. State $x(t)$ is ICS with regards to static obstacles if: $\forall\ p(\infty) :\ \exists\ dt \in [t,\infty)$ so that $x[\pi(x(t), p(\infty))] \notin \mathcal{X}_f$.

Computing whether a state is ICS is intractable, since it requires reasoning over an infinite horizon for all possible plans. It is sufficient, however, to consider conservative methods that identify states that are *not* ICS [13, 2]. The approximation reasons over a subset of predefined maneuvers $\Gamma(\infty)$, called here **contingency plans**. If R^i can avoid collisions in the future with static obstacles at $x^i(t_n)$ by guaranteeing that a contingency plan $\gamma^i(\infty) \in \Gamma^i(\infty)$ avoids collisions over an infinite horizon, then $x^i(t_n)$ is not ICS with regards to static obstacles. For cars, braking maneuvers are sufficient since it is possible to reason over an infinite time horizon whether these plans will collide with static obstacles. Circling maneuvers can be used for systems with minimum velocity limits, such as airplanes.

Multiple moving robots pose new challenges for ICS. Trajectories π^i and π^j may be compatible for the next cycle, but the corresponding robots may reach states that will inevitably lead them in a future collision. Thus, safety notions have to be extended into the multi-robot case. It is still necessary for computational reasons to be conservative and focus only on a set of contingency plans. For m robots $\{R^1, R^2, \ldots, R^m\}$ executing plans $\{p^1(dt^1), p^2(dt^2), \ldots, p^m(dt^m)\}$ at states $\{x^1(t), x^2(t), \ldots, x^m(t)\}$, state $x^i(t)$ is considered a **safe state** if:

$$\exists\ \gamma^i(\infty) \in \Gamma^i(\infty) \text{ so that } \forall\ t\ \ \in [t,\infty):\quad x[\pi^i(x^i(t), \gamma^i(\infty))](t\ ) \in \mathcal{X}_f \text{ and}$$
$$\forall\ j \in [1,m],\ j \neq i,\ \exists\ \gamma^j(\infty) \in \Gamma^j(\infty):\ \pi^i(x^i(t), \gamma^i(\infty)) \asymp \pi^j(\ \pi^j(x^j(t), p^j(dt^j)),\ \gamma^j(\infty)).$$

In the above definition, dt^j is the remaining of robot R^j's cycle past time t. Note that a trajectory concatenation is used for R^j's trajectory. In this trajectory concatenation, $p^j(dt^j)$ is executed for time dt^j and then the contingency $\gamma^j(\infty)$ is applied. The reason is that as robots decide asynchronously, it may happen that at t, robot R^j has already committed to plan $p^j(dt^j)$. Extending the assumption in the problem statement about compatible starting states, the following discussion will assume that the initial states of all the robots are safe states. Then an algorithm for the DMPD problem must maintain the following invariant for each robot and planning cycle:

Safety Invariant: The selected trajectory $\pi^i\ (x^i(t_n^i), p^i(t_n^i : t_{n\ \ 1}^i))$:

a) Must be collision-free with obstacles.
b) Must be compatible with all other robots, during the cycle $(t_n^i : t_{n\ \ 1}^i)$:
$\pi^i\ (x^i(t_n^i), p^i(t_n^i : t_{n\ \ 1}^i)) \asymp \pi^j\ (x^j(t_n^i), p^j(t_n^i : t_{n\ \ 1}^i)),\ \ \forall j \neq i.$

c) The resulting state $x[\pi^i](t^i_{n\ 1})$ is safe for all possible future plans $p^j(t^i_{n\ 1} : \infty)$ selected by other robots ($j \neq i$). In other words, the concatenation of π^i with $\gamma^i(\infty)$ must be compatible with the concatenations of other vehicles, i.e., $\forall j \neq i$: $\pi^i(\pi^i(x^i(t^i_n), p^i(t^i_n : t^i_{n\ 1})), \gamma^i(\infty)) \asymp \pi^j(\pi^j(x^j(t^i_n), p^j(t^i_n : t^i_{n\ 1})), \gamma^j(\infty))$.

Point c) above means that R^i has a contingency plan at $x[\pi^i](t^i_{n\ 1})$, which can be safely followed for the other robots' choices given the algorithm. If the invariant holds for all the robots, then they will always be safe. If for any reason a robot cannot find a plan that satisfies these requirements, then it can revert to its contingency that guarantees its safety.

Algorithm 4.1. Safe and Asynchronous Operation of R^i During Cycle $[t^i_{n\ 1} : t^i_n]$

```
1: $\Pi^i \leftarrow \emptyset$, $\Pi^{N^i}_{prev} \leftarrow \emptyset$, $\Pi^{N^i}_{new} \leftarrow \emptyset$
2: **for all** $R^j \in N^i$ **do**
3:     $\Pi^{N^i}_{prev} \leftarrow \Pi^{N^i}_{prev} \cup \pi^j(\,\pi^j(x^j(t^j_{n-1}), p^j(t^j_{n-1} : t^j_n)),\ \gamma(t^j_n : \infty)\,)$
       (i.e., include all past trajectories and attached contingencies of neighbors)
4: **while** $t < t^i_n - \epsilon$ **do**
5:     $\pi^i(x^i(t^i_n), p^i(t^i_n : t^i_{n+1})) \leftarrow$ collision-free trajectory from a single-robot planner
6:     $\pi^i_\gamma \leftarrow \pi^i(\,\pi^i(x^i(t^i_n), p^i(t^i_n : t^i_{n+1})),\ \gamma(t^i_{n+1} : \infty))$ (i.e., contingency concatenation)
7:     **if** $\forall\ t \in [t^i_{n+1} : \infty)$ : $x[\pi^i_\gamma](t) \in \mathcal{X}_f$ **then**
8:         $\Pi^i \leftarrow \Pi^i \cup \pi^i_\gamma$
9:         **for all** $\pi^j_\gamma \in \Pi^{N^i}_{prev}$ **do**
10:            **if** $\pi^i_\gamma \not\asymp \pi^j_\gamma$ **then**
11:                $\Pi^i \leftarrow \Pi^i - \pi^i_\gamma$
12:    **if** $R^j \in N^i$ is transmitting a trajectory and an attached contingency **then**
13:        $\Pi^{N^i}_{new} \leftarrow \Pi^{N^i}_{new} \cup \pi^j_\gamma(\,\pi^j(x^j(t^j_n), p^j(t^j_n : t^j_{n+1})),\ \gamma(t^j_{n+1} : \infty)\,)$
14: **for all** $\pi^i_\gamma \in \Pi^i$ **do**
15:     **for all** $\pi^j_\gamma \in \Pi^{N^i}_{new}$ **do**
16:         **if** $\pi^i_\gamma \not\asymp \pi^j_\gamma$ **then**
17:             $\Pi^i \leftarrow \Pi^i - \pi^i_\gamma$
18: **if** $\Pi^i$ empty or if a message was received during compatibility check **then**
19:     $\pi^i_* \leftarrow \pi^i(x^i(t^i_n), \gamma(t^i_n : \infty))$ (i.e., follow the available contingency for next cycle)
20: **else**
21:     $\pi^i_* \leftarrow$ trajectory in $\Pi^i$ which brings $R^i$ closer to the goal given a metric
22: Transmit $\pi^i_*$ to all neighbors in $N^i$ and execute $\pi^i_*$ during next cycle
```

B. Safe and Asynchronous Distributed Solution: Algorithm 4.1, in contrast to Algorithm 3.1, maintains the safety invariant. The protocol follows the same high-level framework and still allows a variety of planning techniques to be used for producing trajectories. The differences with the original algorithm can be summarized as follows:

- The algorithm stores the messages received from neighbors during the previous cycle in the set $\Pi^{N^i}_{prev}$ *(lines 1-3)*. Note that the robots transmit the selected trajectory together with the corresponding contingency *(lines 12-13 and 22)*.
- A contingency plan $\gamma(t^i_{n\ 1} : \infty)$ is attached to every collision-free trajectory $\pi^i(x^i(t^i_n), p^i(t^i_n : t^i_{n\ 1}))$ and the trajectory concatenation π^i is generated *(line 5-6)*. Note that potentially multiple different contingencies can be attached to the trajectory $\pi^i(x^i(t^i_n), p^i(t^i_n : t^i_{n\ 1}))$. Each resulting trajectory concatenation is treated individually by the algorithm.
- The trajectory π^i is added to Π^i only if it is collision-free with static obstacles for an infinite time horizon *(lines 7-8)*, thus guaranteeing that $x[\pi^i](t^i_{n\ 1})$ is not ICS.
- π^i is rejected, however, if it is not compatible with all the trajectories and contingencies of neighbors from the compatibility check *(lines 14-17)*. R^i checks not just trajectories for the next cycle but its trajectory concatenations with contingencies π^i against its neighbors' trajectory concatenations π^j.
- The final change *(lines 18-21)* addresses the possibility that Π^i is empty or when a message arrives while R^i executes its compatibility check. If any of the two is true, then R^i selects to follow the contingency $\gamma(t^i_n : \infty)$, which was used in the previous cycle to prove that $x(t^i_n)$ was safe. Otherwise, R^i selects among the set Π^i the trajectory that brings it closer to the goal according to a desired metric. previous cycle, stored in $\Pi^{N^i}_{prev}$ *(lines 9-11)*.
- The while loop *(lines 4-13)* is executed as long as time t is less than the end of the planning cycle (t^i_n) minus an ϵ time period. Time ϵ should be sufficient for the robot to complete the compatibility check *(lines 14-17)* and the selection process *(lines 18-22)*. If the robot is running out of time, the robot should immediately select a contingency in order to guarantee safety. In a real robot implementation, this can be achieved through an interrupt or a signal that stops execution and enforces the contingency. In a serial implementation ϵ has to be sufficiently large.

Overall, each robot selects a plan $p^i(t^i_n : t^i_{n\ 1})$ and a contingency $\gamma^i(t^i_{n\ 1} : \infty)$ that respect the plans and contingencies of other robots that have been selected before time t^i_n. If no such plan is found or there is no time to check against newly incoming messages, then the contingency $\gamma^i(t^i_n : \infty)$ is selected.

Computational Complexity: The algorithm's complexity depends on the number of neighbors N^i, which in the worst case is the total number of robots N. In order to evaluate the cost of operations involving trajectories, it is important to consider a trajectory representation. A discrete sequence of states can be sampled along a trajectory, given a predefined resolution in time Q (i.e., the technique becomes resolution-safe in this case). Then, let S be the upper limit in the number of states used to represent each trajectory conceternation. P denotes the upper limit in the number of plans considered during each planning cycle for the current agent.

Given the above notation, the complexity of the algorithm's various operations is as follows: (a) *Lines 2-3 :* $S \times N$, (b) *Lines 7 - 8:* $P \times S$, (c) *Lines 9 -11:* $P \times N \times S^2$ (if the states in a trajectory are not accompanied by a global timestamp) or $P \times N \times S$ (if the states are tagged with a global timestamp), (d) *Lines 14-17:* Same as above,

(e) *Lines 20-21: P*, assuming constant time for computing a cost-to-go metric for each state, (f) *Line 22:* $N \times S$.

Overall, the worst-case complexity is: $P \times N \times S^2$. Note that for robots with limited communication, the parameter N is reduced. Furthermore, coarser resolution in the representation of trajectories improves efficiency but introduces the probability of collision due to resolution issues. Similarly, considering fewer plans reduces computational complexity but reduces the diversity of solutions considered at each time step. Finally, lower maximum velocity or higher maximum deacceleration also assist computationally in the case of braking maneuvers.

C. Guaranteeing Maintenance of the Safety Invariant: This section provides a proof that Algorithm 4.1 maintains the safety invariant given some simplyfying assymptions that will be waived later.

Theorem 1: Algorithm 4.1 guarantees the maintenance of the safety invariant in every planning cycle given it holds during the cycle $(t^i_0 : t^i_1)$ and that:

i) all robots can communicate one with another,

ii) plans are transmitted instantaneously between robots.

Proof: The proof is obtained by induction. The *base case* holds for R^i because of the Theorem's assumption that the Invariant holds during cycle $(t^i_0 : t^i_1)$. The *inductive step* will show that if the Invariant holds during the cycle $(t^i_n : t^i_{n\;1})$ then it will also hold during the cycle $(t^i_{n\;1} : t^i_{n\;2})$ for Algorithm 4.1. Without loss of generality consider Figure 2 and focus on robot R^i. To prove the inductive step, it is necessary to show that each one of the three points of the Invariant will be satisfied during $(t^i_{n\;1} : t^i_{n\;2})$. For cycle $(t^i_{n\;1} : t^i_{n\;2})$ there are two cases: (1) A compatible trajectory $\pi^i \;= \pi^i \;\in \Pi^i$ is selected, or (2) the current contingency is returned.

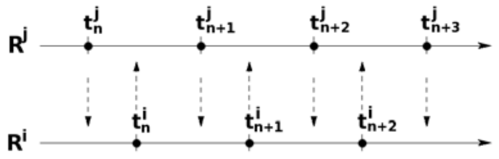


Fig. 2 The replanning cycles of two neighboring robots R^i and R^j. The times denote transitions between planning cycles for each robot. The vertical arrows denote the transmission of information, e.g., at t^i_n, R^i transmits $\pi^i(\ \pi^i(x^i(t^i_n)), p^i(t^i_n : t^i_{n+1})),\ \gamma(t^i_{n+1} : \infty)\ )$.

Case 1: A trajectory $\pi^i \;\in \Pi^i$ *is selected.*

a) Trajectory π^i has to be collision-free as part of Π^i.

b) Assuming instantaneous plan transmission and by time $t^i_{n\;1}$, R^i has been sent and has available the choices of other robots for cycles that start before $t^i_{n\;1}$. Since $\pi^i \in \Pi^i$ is selected, none of these messages arrived during the compatibility check. This means that R^j's trajectory $\pi^j(\ \pi^j(x^j(t^j_{n\;1}), p^j(t^j_{n\;1} : t^j_{n\;2})),\ \gamma(t^j_{n\;2} : \infty)\ )$ is available to R^i during the compatibility check. Then the cycle $(t^i_{n\;1} : t^i_{n\;2})$ can be broken into two parts:

i) During part $(t^i_{n\ 1} : t^j_{n\ 2})$, the selected plan $p^i(t^i_{n\ 1} : t^i_{n\ 2})$ is compatible with $p^j(t^j_{n\ 1} : t^j_{n\ 2})$ because the second plan was known to R^i when selecting π^i .

ii) For part $(t^j_{n\ 2} : t^i_{n\ 2})$ there are two cases for R^j at time $t^j_{n\ 2}$:

- R^j will either select a plan $p^j(t^j_{n\ 2} : t^j_{n\ 3})$ that is compatible with $p^i(t^i_{n\ 1} : t^i_{n\ 2})$,
- or it will resort to a contingency $\gamma^j(t^j_{n\ 2} : \infty)$, which, however, is already compatible with trajectory π^i .

In both cases, R^j will follow a plan that is compatible with $p^i(t^i_{n\ 1} : t^i_{n\ 2})$.

Thus, the second point b) of the Invariant is also satisfied for robots R^i and R^j.

c) For the third point of the Invariant, the contingency $\gamma^i(t^i_{n\ 2} : \infty)$ has to be compatible with the future choices of the other robots. Focus again on the interaction between R^i and R^j. There are again two cases for R^j at time $t^j_{n\ 2}$:

i) R^j will select a plan $p^j(t^j_{n\ 2} : t^j_{n\ 3})$ and a corresponding contingency $\gamma^j(t^j_{n\ 3} : \infty)$. This plan and contingency respect by construction R^i's contingency $\gamma^i(t^i_{n\ 2} : \infty)$, since it was known to R^j at time $t^j_{n\ 2}$.

ii) Or R^j will resort to its contingency $\gamma^j(t^j_{n\ 2} : \infty)$, which, however, the contingency $\gamma^i(t^i_{n\ 2} : \infty)$ respected upon its selection.

In any case, whatever R^j chooses at time $t^j_{n\ 2}$, it is going to follow plans in the future that are compatible with $\gamma^i(t^i_{n\ 2} : \infty)$. Thus, point c) is also satisfied.

Case 2: A contingency $\gamma^i(t^i_{n\ 1} : \infty)$ was selected.

The *inductive hypothesis* implies that $x^i(t^i_{n\ 1})$ is a *s*afe state. Thus:

a) $\gamma^i(t^i_{n\ 1} : t^i_{n\ 2})$ is collision-free with static obstacles

b) The current plans of all robots will be compatible with $\gamma^i(t^i_{n\ 1} : t^i_{n\ 2})$, which was known to them at time t^i_n. Furthermore, $\gamma^i(t^i_{n\ 1} : t^i_{n\ 2})$ already respects the contingencies of other robots that might be executed before $t^i_{n\ 1}$.

c) The state $x^i[\gamma^i(t^i_{n\ 1} : \infty)](t^i_{n\ 2})$ is trivially safe, because R^i can keep executing the same contingency for ever and this contingency will have to be respected by its neighbors, as it will always be known ahead of time.

In both cases, all three points of the Invariant are satisfied for R^i and the inductive step is proved. Thus, if the Invariant holds, the algorithm maintains its validity. □

D. Addressing the Assumptions: Theorem 1 assumed that messages are transmitted instantaneously and that all the robots communicate one with another. The assumption that plans are transmitted instantaneously will not hold in real-world experiments with wireless communication. Similarly, it is more realistic to assume that robots can communicate only if their distance is below a certain threshold. In the latter case, the proposed approach can be invoked using only point to neighborhood communication and thus achieve higher scalability. The following theorem shows that the safety guarantees can be provided without these restrictive assumptions.

Theorem 2: Algorithm 4.1 guarantees the maintenance of the safety invariant in every planning cycle given it holds during cycle $(t^i_0 : t^i_1)$ and that:

i) two robots with limited communication ranges can communicate before they enter into ICS given a predefined set of contingencies $\Gamma(\infty)$,
ii) robots utilize acknowledgments upon the receipt of a trajectory by a neighbor.

Sketch of Proof: Theorem 1 showed that the invariant holds as long as it was valid during the first cycle ($t_0^i : t_1^i$) and that two vehicles can communicate continuously since t_0^i. For two robots with limited communication range, denote as time t_{comm} the beginning of the first planning cycle of either robot after they are able to communicate. If at t_{comm}, both robots have available a contingency $\gamma(\infty) \in \Gamma(\infty)$, that can be used to prove the safety of their corresponding states, then all the requirements of Theorem 1 are satisfied for $t_0^i = t_{comm}$. Thus the invariant will be maintained.

Regarding the issue of delayed messages, consider the case that R^j's cycle ends at time t_n^j, which is before the end of the neighboring R^i's cycle at time t_n^i. Figure 3 provides an example. If the transmission of the trajectory π^j to R^i is delayed, it might arrive after time t_n^i and R^i cannot detect that it did not take into account the choice of R^j during its compatibility check given Algorithm 4.1. Thus, R^i's choice might end up being incompatible with π^j. This problem becomes more frequent for robots that have synchronized cycles. Nevertheless, if an acknowledgment that signals the reception of a trajectory by a neighbor is used R^i can acknowledge the message's reception, whether it arrives before or after t_n^i. If the acknowledgment arrives at R^j before t_n^j, it knows that it is safe to execute π^j. If the acknowledgment is not received on time, R^j can revert to its contingency which is by construction respected by the future plan of R^i, whatever this is. Thus, the introduction of an acknowledgment resolves the issue of possible delays in the transmission of trajectories. □

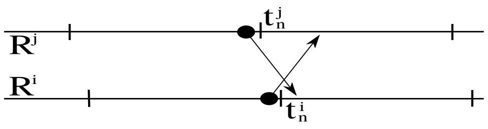


Fig. 3 If messages arrive after the start of a neighbor's future cycle, as with the message from R^j to R^i above, this is problematic.

5 Experimental Results

To validate the theoretical discussion, simulated experiments were conducted. Our first experiments revealed performance deficits, however, practical modifications in the implementation of the algorithm were made. These resulted in significant speed ups and quick convergence to a solution.

Implementation Specifics: This section describes some steps to make the implementation of Algorithm 4.1 more efficient computationally. In particular:

- Instead of checking all the candidate plans Π^i with the trajectories of the neighbors $\Pi_{new}^{N^i}$, only the best plan in Π^i according to a metric is checked. If this plan fails the check, then the previous contingency is selected.

- At each step of the “while” loop in Algorithm 4.1 (lines 4-13), the implementation propagates an edge along a tree of trajectories using a sampling-based planner, instead of generating an entire trajectory. If the edge intersects $t^i_{n\ 1}$, a contingency $\gamma(t^i_{n\ 1} : \infty)$ is extended from $x(t^i_{n\ 1})$. If the contingency is collision-free and compatible with the available trajectories of neighbors in $\Pi^{N^i}_{prev}$, $x(t^i_{n\ 1})$ is assumed safe. Otherwise, it is unsafe and no future expansion of an edge is allowed past $x(t^i_{n\ 1})$.
- The sampling-based expansion of the tree structure of trajectories is biased given a potential field in the workspace that promotes the expansion of the tree towards the goal [2]. The tree expansion is also biased away from other vehicles. Different algorithms can be considered for the actual planning process [20, 15, 8, 23].
- Each robot maintains a common buffer for the sets $P^{N^i}_{prev}$ and $P^{N^i}_{new}$ from each neighbor. As new trajectories are transmitted, they replace the part of old trajectories that has already been executed by a neighbor along the buffer.
- The latency in the experimental setup was relatively low. Thus, the situation in Figure 3 did not arise. Thus, the acknowledgement step was not included for the experiments presented below, which reduced the number of peer-to-peer messages.

Modeled System: The experiments presented in this paper are using the model of a second-order car like vehicle [19] shown on the right side, where (x,y) are the car’s reference point in Cartesian coordinates, θ is the car’s orientation, w its velocity and ζ the steering angle. The controls are α, the acceleration, and ϕ the rate of change of the steering angle. There are limits both for state and control parameters ($||w|| < w_{max}$, $||\zeta|| < \zeta_{max}$, $||\alpha|| < \alpha_{max}$, $||\phi|| < \phi_{max}$). All robots have range-limited communication out to 30% of the total environment width, and brake to zero speed for contingency.

$$\begin{aligned}\dot{x} &= w{\cdot}cos\zeta{\cdot}cos\theta\\ \dot{y} &= w{\cdot}cos\zeta{\cdot}sin\theta\\ \dot{\theta} &= w{\cdot}sin\zeta\\ \dot{w} &= \alpha\\ \dot{\zeta} &= \varphi\end{aligned}$$

Environments. Four simulated environments were used for the experiments:

1. An “empty” environment (Fig. 4 (left)),
2. an “office” environment (Fig. 1),
3. a “ random” environment (Fig. 4 (right)), and
4. an “intersection” environment with two crossing corridors (Fig. 5).

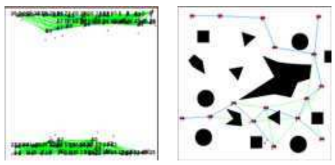


Fig. 4 Starting positions for the “empty” and “random” environments.

These environments are presented in approximate order of difficulty. The various experiments tested different numbers of vehicles: 2, 4, 8, 16, 32, 48. Because the 16 robots alone took up 6% of the entire workspace (ignoring obstacles), the size of the robots was reduced to half for the 32 robot case, and to a quarter of their size for the 48 robot case. If this was not done, then the robots would take up 12% and 18% of the workspace, respectively. Since much of the workspace is already occupied by obstacles, this reduction in size assists in reducing clutter effects that effect solution time. The empty environment was the easiest to solve. The office environment was chosen as a gauge for how hard a structured environment can be. The robots, in their original size, are about 1/5 of the size of the hallway. In the random environment, there were polygons of varying shapes and sizes. The intersection case seemed to be the hardest to solve, since the robots not only have to navigate through a relatively narrow passage together with their neighbors, but they are all forced to traverse the center, almost simultaneously.

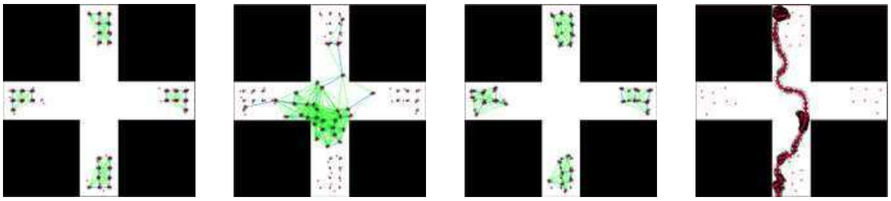
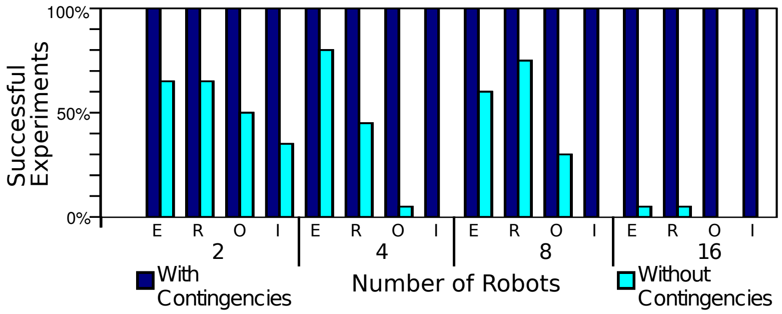


Fig. 5 Snapshots from a typical run with 32 robots; Final image is the full trajectory of robot 0.

When possible, starting/goal locations were identical across runs as more robots were added. Experiments for the same number of robots have the same start/goal locations. All experiments were repeated at least 10 times. The algorithm was run in real time such that computation time is equal to execution time.

Evaluation of Safety. To verify that the system implemented truly provides the guarantees presented in this paper, three different cases were considered for the algorithm: (i) an implementation without contingencies, (ii) with contingencies but for robots with synchronized cycles and (iii) with contingencies and robots that are *not* synchronized. For each type of experiment the following figure reports the percentage of successful experiments. 20 experiments were executed for each case, averaging across synchronous and asynchronous cases. The results presented clearly indicate that enabling contingencies results in a safe system in all cases.



Scalability and Efficiency. Once the safety of the approach was confirmed, the focus turned on evaluating the effects of contingencies. A high-selection rate of contingencies is expected to decrease the performance of the robots, as these plans are not selected to make progress towards the goal. The following table presents the average duration of experiments in seconds and the average velocity achieved by the robots both for the case without contingencies and the case with contingencies (both for synchronized and asynchronous robots). The performance data without contingencies is from the cases where none of the robots entered `ICS`, which means they often correspond to fewer than 20 experiments, and in some cases there is no successful experiment without contingencies to compare against.

Effects of Contingencies		Number of Robots							
		2		4		8		16	
Scenes	Approach	Time	Vel.	Time	Vel.	Time	Vel.	Time	Vel.
Empty	Without Cont.	85.1	6.7	84.5	4.8	82.9	3.9	87.5	3.4
	With	82.6	6.9	90.9	4.7	88.8	3.7	335.8	1.4
Office	Without Cont.	97.0	8.3	98.1	6.6	X	X	X	X
	With	99.1	8.2	111.5	5.9	206.9	2.7	553.3	1.0
Random	Without Cont.	87.2	6.5	84.4	4.8	88.3	3.6	X	X
	With	88.0	6.5	103.1	4.4	92.4	3.6	604.8	1.3
Intersection	Without Cont.	101.0	8.0	100.0	8.0	X	X	X	X
	With	108.9	7.5	272.5	4.2	469.1	2.3	1415.4	1.0

The behavior of the robots is indeed more conservative when contingencies are employed and it takes longer to complete an experiment. Although the algorithm has no progress guarantees, the randomized nature of the probabilistically complete planning algorithms helped to offset this. The simulations always eventually resulted in a solution for the tested problems even if the robots temporarily entered oscillatory motions. The local penalty for trajectories that brought an agent in close proximity to neighboring robots helped to reduce the occurrence of oscillations and resulted in significant improvements in performance.

Synchronous vs. Asynchronous. Another objective of the experimentation procedure was to evaluate the differences in the performance of the algorithm between the synchronous and the asynchronous case. In the synchronous case, all robots have a zero time offset but they are not aware of their synchronicity and they are not taking advantage of it as in previous work [3]. In the asynchronous case, the offsets are the same across 10 averaged runs. These offsets are randomly precomputed and range from 0 to a maximum of 3/4 of the planning cycle.

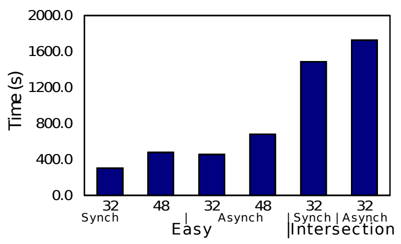
When the robots' cycles are synchronized, then it will be often the case that robots are transmitting simultaneously, and potentially during the compatibility check of their neighbors. This in certain cases results in slightly longer durations for the completion of an experiment, as well as lower average velocities, but overall there is no consistent effect as in the random and empty scenes, there is a performance boost under synchronous operation, especially as the number of robots increases. In comparison to previous work [3] where synchronicity was specifically

Sync. Vs. Async.		Number of Robots							
		2		4		8		16	
Scenes	Approach	Time	Vel.	Time	Vel.	Time	Vel.	Time	Vel.
Empty	Asynch.	81.5	7.0	85.5	4.8	87.3	3.8	400.0	1.4
	Synch.	83.8	6.8	96.3	4.5	90.3	3.6	271.5	1.4
Office	Asynch.	96.0	8.4	112.5	6.0	197.5	2.8	541.0	1.0
	Synch.	102.3	7.9	110.5	5.9	216.3	2.7	565.5	1.0
Random	Asynch.	85.5	6.7	90.8	4.5	85.8	3.8	729.6	1.4
	Synch.	90.5	6.3	115.5	4.2	99.0	3.5	480.0	1.3
Intersection	Asynch.	105.0	7.8	268.3	4.1	335.8	2.9	899.8	1.3
	Synch.	112.8	7.2	276.8	4.3	602.5	1.6	1931.0	0.8

taken advantage of, it is clear that the quality of the paths selected are worse in the current asynchronous implementation. However, it is expected that further research in asynchronous coordination algorithms can reduce this performance gap.

Scaling. Larger scale simulations for 32 and 48 robots were run to study the algorithm's scalability. For these cases, the approach without contingencies always fails. Note that as mentioned earlier, these robots are of reduced size to decrease the effects on completion time due to a cluttered environment.

Achieving safe, asynchronous operation for 48 second-order systems with the proposed setup is a challenge. The agent model is complex as are the safety guarantees address the ICS issue. The simulation environment mimics the constraints of real-world communication by running each agent on a separate processor and allowing only message-passing communication (TCP sockets). An experiment with 48 robots requires 49 separate processors (1 processor is used as a simulation server).



Parameter Evaluation. An important parameter for the proposed approach is the duration of the planning cycle. For shorter durations of cycles, there was a higher deviation between runs and it was not possible to execute the larger experiments with 32 and 48 robots for a cycle duration less than 2 seconds. This limitation is due to the single thread running the world simulation. It is expected that the limit in hardware implementation would be dependent on the communication latency. The average completion time shows a noticeable increase as the duration of a cycle increases. The experiments presented in the previous tables were executed for a cycle duration of 2.5 seconds.

Planning Cycle		Number of Robots							
		2		4		8		16	
Scene	Cycle	Time	Vel.	Time	Vel.	Time	Vel.	Time	Vel.
Empty	1.0s	53.3	10.8	52.5	7.8	59.2	5.8	96.9	3.5
	1.5s	59.3	9.7	63.8	6.4	60.0	5.3	197.1	2.0
	2.0s	71.4	8.0	74.0	5.8	75.6	4.2	116.8	2.7
	2.5s	79.5	7.2	82.8	5.2	86.5	3.7	134.0	2.2
	3.0s	98.4	5.8	98.4	4.4	99.9	3.2	135.0	2.0
	3.5s	167.7	3.8	193.6	2.5	125.5	1.7	482.7	0.7

6 Discussion

This paper presented a fully distributed algorithm that guarantees `ICS` safety for a number of second-order robots that move in the same environment. Simulations confirm that the framework indeed provides safety and is scalable and adaptable. Additional experiments not presented above were conducted for a system with positive minimum velocity, i.e., a system that cannot brake to zero velocity. Safety was achieved for this system using a different set of contingencies than braking maneuvers. In this case, the system was required to turn into the tightest circle possible without exceeding the specified limits on velocity and turning rate. Future work includes: (a) considering robots with different durations for planning cycles, (b) dealing with unreliable communication, (c) studying the effects of motion uncertainty to the protocol's performance, (d) distributed optimization for improving the quality of paths selected despite the asynchronous operation, (e) dealing with non-cooperating vehicles and (f) addressing tasks that go beyond moving from initial to final states. Experiments using physical systems with interesting dynamics would provide a real-world verification of the approach.

Acknowledgements. Work by D. Grady and L. Kavraki on this paper has been supported in part by the US Army Research Laboratory and the US Army Research Office under grant number W911NF-09-1-0383 and by NSF IIS 0713623. Work by K. Bekris has been supported by NSF CNS 0932423. Any conclusions expressed here are of the authors and do not reflect the views of the sponsors. The authors would like to thank the anonymous reviewers for their helpful comments.

References

1. Alami, R., Simeon, T., Krishna, K.M.: On the influence of sensor capacities and environment dynamics onto collision-free motion plans. In: IEEE/RSJ IROS, Lausanne, CH (2002)
2. Bekris, K.E., Kavraki, L.E.: Greedy but safe replanning under kinodynamic constraints. In: IEEE ICRA, Rome, Italy (2007)

3. Bekris, K.E., Tsianos, K., Kavraki, L.E.: Safe and distributed kinodynamic replanning for vehicular networks. Mobile Networks and Applications 14(3), 292–308 (2009)
4. Van den Berg, J., Lin, M., Manocha, D.: Reciprocal velocity obstacles for real-time multi-agent navigation. In: Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA) (2008)
5. Bruce, J., Veloso, M.: Real-time multi-robot motion planning with safe dynamics. In: Schultz, A., Parker, L., Schneider, F. (eds.) Intern. Workshop on Multi-Robot Systems (2003)
6. Chan, N., Kuffner, J.J., Zucker, M.: Improved motion planning speed and safety using regions of inevitable collision. In: 17th CISM-IFToMM RoManSy (2008)
7. Clark, C., Rock, S., Latombe, J.C.: Motion planning for multi-robot systems using dynamic robot networks. In: IEEE ICRA, Taipei, Taiwan (2003)
8. Dimarogonas, D.V., Kyriakopoulos, K.J.: Decentralized Navigation Functions for Multiple Robotic Agents. Intelligent and Robotic Systems 48(3), 411–433 (2007)
9. Erdmann, M., Lozano-Perez, T.: On multiple moving objects. In: ICRA, pp. 1419–1424 (1986)
10. Fiorini, P., Shiller, Z.: Motion planning in dynamic environments using velocity obstacles. Int. Journal of Robotics Research 17(7) (1998)
11. Fox, D., Burgard, W., Thrun, S.: The dynamic window approach to collision avoidance. IEEE Robotics and Automation Magazine 4(1) (1997)
12. Fraichard, T.: A short paper about motion safety. In: IEEE ICRA, Rome, Italy (2007)
13. Fraichard, T., Asama, H.: Inevitable collision states: A step towards safer robots? In: Advanced Robotics, pp. 1001–1024 (2004)
14. Frazzoli, E., Dahleh, M., Feron, E.: Real-time motion planning for agile autonomous vehicles. AIAA Journal of Guidance, Control and Dynamics 25(1), 116–129 (2002)
15. Hsu, D., Kindel, R., Latombe, J.C., Rock, S.: Randomized kinodynamic motion planning with moving obstacles. IJRR 21(3), 233–255 (2002)
16. Kalisiak, M., Van de Panne, M.: Faster motion planning using learned local viability models. In: Proc. of the IEEE Int. Conf. on Robotics and Automation, Roma, Italy (2007)
17. Lalish, E., Morgansen, K.A.: Decentralized reactive collision avoidance for multivehicle systems. In: IEEE Conference on Decision and Control (2008)
18. Lamiraux, F., Bonnafous, D., Lefebvre, O.: Reactive path deformation for nonholonomic mobile robots. IEEE Transactions on Robotics 20, 967–977 (2004)
19. Laumond, J.P. (ed.): Robot Motion Planning and Control. Lectures Notes in Control and Information Sciences, vol. 229. Springer, Heidelberg (1998)
20. LaValle, S.M., Kuffner, J.J.: Randomized kinodynamic planning. IJRR 20(5), 378–400 (2001)
21. Martinez-Gomez, L., Fraichard, T.: Collision avoidance in dynamic environments: An ics-based solution and its comparative evaluation. In: IEEE ICRA, Kobe, Japan (2009)
22. Peng, J., Akella, S.: Coordinating multiple robots with kinodynamic constraints along specified paths. Int. Journal of Robotics Research 24(4), 295–310 (2005)
23. Pivtoraiko, M., Knepper, R.A., Kelly, A.: Differentially constrained mobile robot motion planning in state lattices. Journal of Field Robotics 26, 308–333 (2009)
24. Reif, J., Sharir, M.: Motion planning in the presence of moving obstacles. In: Proc. of the IEEE Int. Symp. on Foundations of Computer Science, Portland, OR (1985)

25. Vatcha, R., Xiao, J.: Perceived CT-Space for motion planning in unknown and unpredictable environments. In: Workshop on Algorithmic Foundations of Robotics, Mexico (2008)
26. Wikman, M.S., Branicky, M., Newman, W.S.: Reflexive collision avoidance: A generalized approach. In: Proc. of the IEEE Int. Conf. on Robotics and Automation (1993)
27. Yang, Y., Brock, O.: Elastic roadmaps: Globally task-consistent motion for autonomous mobile manipulation in dynamic environments. In: Robotics: Science and Systems II (2006)

Incremental Sampling-Based Algorithms for a Class of Pursuit-Evasion Games

Sertac Karaman and Emilio Frazzoli

Abstract. Pursuit-evasion games have been used for modeling various forms of conflict arising between two agents modeled as dynamical systems. Although analytical solutions of some simple pursuit-evasion games are known, most interesting instances can only be solved using numerical methods requiring significant offline computation. In this paper, a novel incremental sampling-based algorithm is presented to compute optimal open-loop solutions for the evader, assuming worst-case behavior for the pursuer. It is shown that the algorithm has probabilistic completeness and soundness guarantees. As opposed to many other numerical methods tailored to solve pursuit-evasion games, incremental sampling-based algorithms offer anytime properties, which allow their real-time implementations in online settings.

1 Introduction

Pursuit-evasion games have been used for modeling various problems of conflict arising between two dynamic agents with opposing interests [1, 2]. Some examples include multiagent collision avoidance [3], air combat [4], and path planning in an adversarial environment [5]. The class of pursuit-evasion games that will be considered in this paper is summarized as follows. Consider two agents, an evader and a pursuer; the former is trying to "escape" into a goal set in minimum time, and the latter is trying to prevent the evader from doing so by "capturing" her, while both agents are required to avoid collision with obstacles. The evader is only aware of

Sertac Karaman
Laboratory for Information and Decision Systems,
Department of Electrical Engineering and Computer Science,
Massachusetts Institute of Technology
e-mail: `sertac@mit.edu`

Emilio Frazzoli
Laboratory for Information and Decision Systems,
Department of Aeronautics and Astronautics,
Massachusetts Institute of Technology
e-mail: `frazzoli@mit.edu`

the initial state of the pursuer, while the pursuer has access to full information about the evader's trajectory. This class of pursuit-evasion games is of interest when the evader can be easily detected by stealthy pursuers, who operate from known locations. Problems in this class include the generation of trajectories for an airplane to avoid threats from known Surface-to-Air Missile (SAM) sites, or for a ship to avoid attacks by pirates based at known locations. The information structure of this class of pursuit-evasion games is such that the evader discloses her (open-loop) strategy first, and the pursuer decides his strategy accordingly. In this setting, the evader's strategy should be chosen carefully, considering the worst-case (from the evader's point of view) response of the pursuer. Rational players in this game will choose a Stackelberg strategy with the evader as a leader [6].

Analytical solutions to certain classes of pursuit-evasion games, e.g., the "homicidal chauffeur" and the "lady in the lake" games, exist [1, 2]. However, for problems involving agents with more complex dynamics, or for problems involving complex environments (e.g., including obstacles), existing analytical techniques are difficult to apply. For example, the pursuit-evasion game addressed in this article can be solved in principle by determining the set of all states that can be reached by the evader before the pursuer, and then choosing the optimal trajectory for the evader, if one exists, within this set [1]. In the simple case of kinematic agents moving with bounded speed within an empty environment, such a set coincides with the evader's region in a Voronoi tesselation generated by the evader's and pursuer's initial conditions. However, analytical methods for computation of this set are not available in the general case in which non-trivial dynamics and obstacles are considered.

Standard numerical approaches for solving pursuit-evasion games are based on either direct or indirect methods [7]. The former reduce the problem to a sequence of finite dimensional optimization problems through discretization [8], whereas the latter solves the Isaacs partial differential equation with boundary conditions using, e.g., multiple shooting [9, 10], collocation [11, 12], or level-set methods [3, 13].

A number of algorithms for motion planning in the presence of dynamic, possibly adversarial obstacles, have been proposed in the context of mobile robotics. A common approach relies on planning in a 'space-time' state space, avoiding spatio-temporal regions representing possible motions of the dynamic obstacles [14, 15]. However, such regions, representing reachable sets by the dynamic obstacles, are typically hard to compute exactly in the general case, and conservative approximations are used, e.g., to estimate regions of inevitable collision [16]. Other recent contributions in this area include [17–24].

Several types of pursuit-evasion games have been studied from an algorithmic perspective. In particular, pursuit games on graphs [25–27] as well as on polygonal environments [28–30] have received significant attention during the last decade. More recently, pursuit-evasion games on timed roadmaps have also been considered [31]. All these approaches typically impose severe limitations on the allowable agents' dynamics, e.g., by considering only finite state spaces and discrete time.

Based on recent advances in incremental sampling-based motion planning algorithms, we propose a new method for solving the class of pursuit-evasion games under consideration. In fact, the game we consider is a generalization of the kinodynamic motion planning problem [15]. During the last decade, a successful algorithmic approach to this problem has been the class of sampling-based methods including, e.g., Probabilistic RoadMaps (PRMs) [32], which construct a roadmap by connecting randomly-sampled states with feasible trajectories so as to form a strong hypothesis of the connectivity of the state space, and, in particular, the initial state and the goal region.

Incremental versions of sampling-based motion planning methods were proposed to address on-line planning problems [17, 33]. In particular, the Rapidly-exploring Random Tree (RRT) algorithm proposed in [33] has been shown to be very effective in practice, and was demonstrated on various platforms in major robotics events (see, e.g., [34]). Very recently, optimality properties of incremental sampling-based planning algorithms were analyzed and it was shown that, under mild technical assumptions, the RRT algorithm converges to a non-optimal solution with probability one [35]. In [35], the authors have proposed a new algorithm, called RRT*, which converges to an optimal solution almost surely, while incurring essentially the same computational cost when compared to the RRT. The RRT* algorithm can be viewed as an anytime algorithm for the optimal motion planning problem. Loosely speaking, an anytime algorithm produces an approximate solution and gradually improves the quality of the approximation given more computation time [36, 37]. The quality measure is defined, e.g., with respect to a cost function.

In this paper, inspired by incremental sampling-based motion planning methods, in particular the RRT* algorithm, we propose an incremental sampling-based algorithm that solves the pursuit-evasion game with probabilistic guarantees. More precisely, if evader trajectories that escape to the goal set while avoiding capture exist, then the output of the algorithm will converge to the minimum-cost one with probability approaching one as the number of samples increases.To the best of authors' knowledge, this algorithm constitutes the first algorithmic approach to numerically solve, with both asymptotic and anytime guarantees, the class of pursuit-evasion games under consideration.

The paper is organized as follows. Section 2 formulates the problem. The proposed solution algorithms are introduced in Section 3. The algorithm is shown to be probabilistically sound and probabilistically complete in Section 4. Simulation examples are provided in Section 5. Conclusions and remarks on future work can be found in Section 6.

2 Problem Definition

We consider a two-player zero-sum differential game in which one of the players, called the evader, tries to escape in minimum time to a goal set, while the second player, called the pursuer, tries the capture the evader before it reaches the goal set.

More formally, consider a time-invariant dynamical system described by the differential equation $\dot{x}(t) = f(x(t), u_{\mathrm{e}}(t), u_{\mathrm{p}}(t))$, where $x : t \mapsto x(t) \in X \subset \mathbb{R}^d$ is the state trajectory, $u_{\mathrm{e}} : t \mapsto u_{\mathrm{e}}(t) \in U_{\mathrm{e}} \subset \mathbb{R}^{m_{\mathrm{e}}}$ is the evader's control input, $u_{\mathrm{p}} : t \mapsto$

$u_{\mathrm{p}}(t) \in U_{\mathrm{p}} \subset \mathbb{R}^{m_{\mathrm{p}}}$ is the pursuer's control input. The sets X, U_{e}, and U_{p} are assumed to be compact, the control signals u_{e} and u_{p} are essentially bounded measurable, and $f(z, w_{\mathrm{e}}, w_{\mathrm{p}})$ is locally Lipschitz in z and piecewise continuous in both w_{e} and w_{p}. Consider also an obstacle set X_{obs}, a goal set X_{goal}, and a capture set X_{capt}; these sets are assumed to be open subsets of X, and X_{goal} and X_{capt} are disjoint.

Given an initial condition $x(0) \in X \setminus X_{\mathrm{obs}}$ and the control inputs of the evader and the pursuer, a unique state trajectory can be computed. The final time of the game is given by $T = \inf\{t \in \mathbb{R}_{\geq 0} : x(t) \in \mathrm{cl}\left(X_{\mathrm{goal}} \cup X_{\mathrm{capt}}\right)\}$. Since this is a zero-sum game, only one objective function will be considered, defined as follows: $L(u_{\mathrm{e}}, u_{\mathrm{p}}) = T$, if $x(T) \in \mathrm{cl}(X_{\mathrm{goal}})$; and $L(u_{\mathrm{e}}, u_{\mathrm{p}}) = +\infty$, otherwise. The evader tries to minimize this objective function by escaping to the goal region in minimum time, while the pursuer tries to maximize it by capturing the evader before she reaches the goal.

Let $\mathrm{BR} : U_{\mathrm{e}} \to U_{\mathrm{p}}$ denote a transformation that maps each evader trajectory to the best response of the pursuer, i.e., $\mathrm{BR}(u_{\mathrm{e}}) := \arg\max_{u_{\mathrm{p}}} L(u_{\mathrm{e}}, u_{\mathrm{p}})$. In the game described above, the evader picks her strategy so that $L^* = L(u_{\mathrm{e}}^*, \mathrm{BR}(u_{\mathrm{e}}^*)) \leq L(u_{\mathrm{e}}, e_{\mathrm{p}})$ for all u_{e} and all u_{p}. Let $u_{\mathrm{p}}^* := \mathrm{BR}(u_{\mathrm{e}}^*)$. Then, ($u_{\mathrm{e}}^*$ and u_{p}^*) are called the (open-loop) *Stackelberg strategies* of this differential game [2].

Note that open-loop Stackelberg strategies computed for the evader in this way would be conservative when compared to the saddle-point equilibrium of a pursuit-evasion game with feedback information pattern (see [2]). Open-loop Stackelberg strategies correspond to trajectories that would allow escape without any additional information on the pursuer other than the initial condition. Should other information become available, or should the pursuer not play optimally, the time needed to reach the goal set may be further reduced. In addition, even in the case in which escape is unfeasible (i.e., $L^* = +\infty$) under the open-loop information structure for the evader, there may exist feedback strategies that would allow the evader to escape while avoiding capture.

As common in pursuit-evasion games, the problem considered in this paper further possesses a separable structure, in the following sense. It is assumed that the state can be partitioned as $x = (x_{\mathrm{e}}, x_{\mathrm{p}}) \in X_{\mathrm{e}} \times X_{\mathrm{p}} = X$, the obstacle set can be similarly partitioned as $X_{\mathrm{obs}} = (X_{\mathrm{obs,e}} \times X_{\mathrm{p}}) \cup (X_{\mathrm{e}} \times X_{\mathrm{obs,p}})$, where $X_{\mathrm{obs,e}} \subset X_{\mathrm{e}}$ and $X_{\mathrm{obs,p}} \subset X_{\mathrm{p}}$, the goal set is such that $X_{\mathrm{goal}} = (X_{e,\mathrm{goal}} \times X_{\mathrm{p}}) \setminus X_{\mathrm{capt}}$, where $X_{e,\mathrm{goal}} \subset X_{\mathrm{e}}$, and the dynamics are decoupled as follows:

$$\frac{d}{dt}x(t) = \frac{d}{dt}\begin{bmatrix} x_{\mathrm{e}}(t) \\ x_{\mathrm{p}}(t) \end{bmatrix} = f(x(t), u(t)) = \begin{bmatrix} f_{\mathrm{e}}\left(x_{\mathrm{e}}(t), u_{\mathrm{e}}(t)\right) \\ f_{\mathrm{p}}\left(x_{\mathrm{p}}(t), u_{\mathrm{p}}(t)\right) \end{bmatrix}, \quad \text{for all } t \in \mathbb{R}_{\geq 0},$$

It is also assumed that the initial condition is an equilibrium state for the pursuer, i.e., there exists $u_{\mathrm{p}}' \in U_{\mathrm{p}}$ such that $f_{\mathrm{p}}(x_{\mathrm{init,p}}, u_{\mathrm{p}}') = 0$.

Assume that there exist a Stackelberg strategy enabling the evader to escape (i.e., $L^* < +\infty$). An algorithm for the solution of the pursuit-evasion game defined in this section is said to be *sound* if it returns a control input u_{e}' such that $\max_{u_{\mathrm{p}}} L(u_{\mathrm{e}}', u_{\mathrm{p}})$, is finite. An algorithm is said to be *complete* if it terminates in finite time returning a solution u_{e}' as above if one exists, and returns failure otherwise.

The pursuer dynamics can be used to naturally model one or more pursuing agents, as well as moving obstacles whose trajectories are a priori unknown. It is known that even when the number of degrees of freedom of the robot is fixed, the motion planning problem with moving obstacles is NP-hard, whenever the robot has bounds on its velocities. In fact, a simple version of this problem, called the *2-d asteroid avoidance problem*, is NP-hard [38].

The discussion above also suggests that complete algorithms aimed to solve the proposed pursuit-evasion game will be computationally intensive. To overcome this difficulty, we propose a sampling-based algorithm, which is both *probabilistically sound*, i.e., such that the probability that the returned trajectory avoids capture converges to one, and *probabilistically complete*, i.e., such that the probability that it returns a solution, if one exists, converges to one, as the number of samples approaches infinity. Finally, the proposed algorithm is *asymptotically optimal* in the sense that the cost of the returned trajectory converges to the value of the game L^*, almost surely, if $L^* < +\infty$.

3 Algorithm

In this section, an algorithm that solves the proposed pursuit-evasion game with probabilistic soundness and completeness guarantees is introduced. This algorithm is closely related to the RRT* algorithm recently introduced in [35], which will be discussed first. RRT* is an incremental sampling-based motion planning algorithm with the asymptotic optimality property, i.e., almost-sure convergence to optimal trajectories, which the RRT algorithm lacks [35]. In fact, it is precisely this property of the RRT* that allows us to cope with the game introduced in the previous section.

Before formalizing the algorithm, some primitive procedures are presented below. Let $\alpha \in \{\mathrm{e}, \mathrm{p}\}$ denote either the evader or the pursuer.

Sampling: The sampling procedure $\texttt{Sample}_\alpha : \mathbb{N} \to X_\alpha$ returns independent and identically distributed samples from X_α. The sampling distribution is assumed to be absolutely continuous with density bounded away from zero on X_α.

Distance Function: Given two states z_1 and z_2, let $\mathrm{dist}_\alpha(z_1, z_2)$ be a function that returns the minimum time to reach z_2 starting from z_1, assuming no obstacles. Clearly, the distance function evaluates to the Euclidean distance between z_1 and z_2 when $f_\alpha(x_\alpha, u_\alpha) = u_\alpha$ and $\|u_\alpha\| \leq 1$.

Nearest Neighbor: Given a tree $G = (V, E)$, where $V \subset X_\alpha$, and a state $z \in X_\alpha$, $\texttt{Nearest}_\alpha(G, z)$ returns the vertex $v \in V$ that is closest to z. This procedure is defined according to the distance function as $\texttt{Nearest}_\alpha(G, z) = \arg\min_{v \in V} \mathrm{dist}(v, z)$.

Near-by Vertices: Given a tree $G = (V, E)$, where $V \subset X_\alpha$, a state $z \in X_\alpha$, and a number $n \in \mathbb{N}$, $\texttt{Near}_\alpha(G, z, n)$ procedure returns all the vertices in V that are sufficiently close to z, where closeness is parameterized by n. More precisely, for any $z \in X_\alpha$, let $\texttt{Reach}_\alpha(z, l) := \{z' \in X \,|\, \mathrm{dist}(z, z') \leq l \vee \mathrm{dist}(z', z) \leq l\}$. Given, z and n, the distance threshold is chosen such that the set $\texttt{Reach}_\alpha(z, l(n))$ contains a ball of volume $\gamma_\alpha \frac{\log n}{n}$, where γ_α is an appropriate constant. (This particular scaling rate is chosen since it ensures both computational efficiency and asymptotic optimality of the RRT* algorithm [35, 39].) Finally, we define $\texttt{Near}_\alpha(G,z,n) := V \cap \texttt{Reach}_\alpha(z,l(n))$.

Collision Check: Given a state trajectory $x : [0,t] \to X_\alpha$, the $\texttt{ObstacleFree}_\alpha(x)$ procedure returns true if and only if x lies entirely in the obstacle-free space, i.e., if and only if $x(t') \notin X_{\mathrm{obs},\alpha}$ for all $t' \in [0,t]$.

Local Steering: Given two states $z_1, z_2 \in X_\alpha$, the $\texttt{Steer}_\alpha(z_1,z_2)$ function returns a trajectory that starts from z_1 and ends at z_2, ignoring obstacles. We assume that the $\texttt{Steer}$ procedure returns a time optimal trajectory that connects z_1 and z_2 exactly if z_1 and z_2 are sufficiently close to each other. More precisely, there exists an $\bar{\varepsilon} > 0$ such that for all $\|z_1 - z_2\| \leq \bar{\varepsilon}$, the $\texttt{Steer}(z_1,z_2)$ procedure returns (x,u,t) such that $x(0) = z_1$, $x(T) = z_2$, and $\dot{x}(t') = f_\alpha(x(t'),u(t'))$ for all $t' \in [0,t]$, and t is minimized.

Given a vertex v, let x_v be the unique trajectory in the tree that starts from the root vertex and reaches v. Let us denote the time that x_v reaches v by $T(v)$; given a state trajectory $x : [0,t] \to X$, let us denote the ending time t with $\texttt{EndTime}(x)$.

If the pursuer is inactive (e.g., it is not moving), the pursuit-evasion problem in Section 2 reduces to a standard time-optimal kinodynamic motion planning problem. The RRT* algorithm that solves this problem is presented in Algorithm 1.

The RRT* algorithm proceeds similarly to other incremental sampling-based motion planning methods (e.g., the RRT [33]) by first sampling a state a from the obstacle-free space (Line 4) and then extending the tree towards this sample (Line 5). The extension procedure of the RRT*, presented in Algorithm 2, first extends the vertex closest to the sample (Lines 2-3); if the extension is collision-free (Line 4), then the end point of the extension, say z_{new}, is added to the tree as a new vertex (Line 5), as in RRT. However, the RRT* $\texttt{Extend}_\alpha$ procedure differs from others in that it connects the new vertex z_{new} to the vertex that lies within a ball of volume $\Theta(\log(n)/n)$ centered at z_{new}, where $n = |V|$ is the number of vertices in the tree, and incurs the smallest cost to reach z_{new} with a collision-free trajectory (Lines 8-12). Moreover, the RRT* $\texttt{Extend}_\alpha$ procedure extends z_{new} back to the vertices in the tree that are within the ball of same size centered at z_{new}; if the extension to such a vertex, say z_{near}, results in a collision-free trajectory that reaches z_{near} with smaller cost, then tree is "rewired" by connecting z_{near} to z_{new}, instead of its current parent (Lines 13 - 18).

The algorithm that is proposed for the solution of the problem in Section 2 builds on RRT*, and relies on the following additional primitive procedures.

Near-Capture Vertices: The $\texttt{NearCapure}_\alpha$ procedure works in a way that is very similar to the $\texttt{Near}_\alpha$ procedure. Given a tree $G = (V,E)$, a state $z \in X_\alpha$, and a number n, the $\texttt{NearCapture}_\alpha(G,z,n)$ procedure returns all vertices z' that are "close" to being captured from z. In other words, and assuming $\alpha = \mathrm{p}$ for simplicity, let $\texttt{CaptureSet}_\mathrm{p}(z) := \{z' \in X_\mathrm{e} : (z',z) \in X_{\mathrm{capt}}\}$. Then, $\texttt{NearCapture}_\mathrm{p}(G,z,n) = \{v \in V \,|\, \text{there exist } y \in \texttt{CaptureSet}_\mathrm{p}(z) \text{ such that } v \in \texttt{Reach}_\mathrm{e}(y,l(n))\}$.

Remove: Given a graph $G = (V,E)$ on X_α, and a vertex $z \in V$, the procedure $\texttt{Remove}(G,z)$ removes z, all its descendants, and their incoming edges from G.

The algorithm proposed to solve the pursuit-evasion game under consideration is given in Algorithm 3. The algorithm maintains two tree structures encoding candidate paths: the evader tree G_e and the pursuer tree G_p. At each iteration, the

Algorithm 1. The RRT* Algorithm

```
1  $V_e \leftarrow \{z_{init}\}$; $E_e \leftarrow \emptyset$; $i \leftarrow 0$;
2  while $i < N$ do
3      $G_e \leftarrow (V_e, E_e)$;
4      $z_{e,rand} \leftarrow \texttt{Sample}_e(i)$;
5      $(V_e, E_e, z_{e,new}) \leftarrow \texttt{Extend}_e(G_e, z_{e,rand})$;
6      $i \leftarrow i + 1$;
```

Algorithm 2. $\texttt{Extend}_\alpha(G, z)$

```
1  $V' \leftarrow V$; $E' \leftarrow E$;
2  $z_{nearest} \leftarrow \texttt{Nearest}_\alpha(G, z)$;
3  $(x_{new}, u_{new}, t_{new}) \leftarrow \texttt{Steer}_\alpha(z_{nearest}, z)$; $z_{new} \leftarrow x_{new}(t_{new})$;
4  if $\texttt{ObstacleFree}_\alpha(x_{new})$ then
5      $V' \leftarrow V' \cup \{z_{new}\}$;
6      $z_{min} \leftarrow z_{nearest}$; $c_{min} \leftarrow T(z_{new})$;
7      $Z_{nearby} \leftarrow \texttt{Near}_\alpha(G, z_{new}, |V|)$;
8      for all $z_{near} \in Z_{nearby}$ do
9          $(x_{near}, u_{near}, t_{near}) \leftarrow \texttt{Steer}_\alpha(z_{near}, z_{new})$;
10         if $\texttt{ObstacleFree}_\alpha(x_{near})$ and $x_{near}(t_{near}) = z_{new}$ and
           $T(z_{near}) + \texttt{EndTime}(x_{near}) < T(z_{new})$ then
11             $c_{min} \leftarrow T(z_{near}) + \texttt{EndTime}(x_{near})$;
12             $z_{min} \leftarrow z_{near}$;
13     $E' \leftarrow E' \cup \{(z_{min}, z_{new})\}$;
14     for all $z_{near} \in Z_{nearby} \setminus \{z_{min}\}$ do
15         $(x_{near}, u_{near}, t_{near}) \leftarrow \texttt{Steer}_\alpha(z_{new}, z_{near})$;
16         if $\texttt{ObstacleFree}_\alpha(x_{near})$ and $x_{near}(t_{near}) = z_{near}$ and
           $T(z_{near}) > T(z_{new}) + \texttt{EndTime}(x_{near})$ then
17             $z_{parent} \leftarrow \texttt{Parent}(z_{near})$;
18             $E' \leftarrow E' \setminus \{(z_{parent}, z_{near})\}$; $E' \leftarrow E' \cup \{(z_{new}, z_{near})\}$;
19 else
20     $z_{new} = \text{NULL}$;
21 return $G' = (V', E', z_{new})$
```

algorithm first samples a state, $z_{e,rand}$, in the evader's state-space (Line 4) and extends the evader tree towards $z_{e,rand}$ (Line 5). If the extension produces a new vertex $z_{e,new}$ (Line 6), then the algorithm checks whether the time that the evader reaches $z_{e,new}$ is less than that at which the pursuer reaches any pursuer vertex within certain distance to $z_{e,new}$ (Lines 7-10). This distance scales as $\Theta(\log(n)/n)$, where n is the number of vertices in the pursuer tree, G_p. If this condition does not hold, then $z_{e,new}$ is removed from evader's tree (Line 10).

Second, the algorithm samples a new state, $z_{p,rand}$, in the pursuer state space (Line 11) and extends the pursuer's tree towards $z_{p,rand}$ (Line 12). If this extension successfully produces a new vertex, $z_{p,new}$ (Line 13), then the algorithm checks

Algorithm 3. Pursuit-Evasion Algorithm

```
1  $V_\mathrm{e} \leftarrow \{x_\mathrm{e,init}\}$; $E_\mathrm{e} \leftarrow \emptyset$; $V_\mathrm{p} \leftarrow \{x_\mathrm{p,init}\}$; $E_\mathrm{p} \leftarrow \emptyset$; $i \leftarrow 0$;
2  while $i < N$ do
3      $G_\mathrm{e} \leftarrow (V_\mathrm{e}, E_\mathrm{e})$; $G_\mathrm{p} \leftarrow (V_\mathrm{p}, E_\mathrm{p})$;
4      $z_\mathrm{e,rand} \leftarrow \mathtt{Sample}_\mathrm{e}(i)$;
5      $(V_\mathrm{e}, E_\mathrm{e}, z_\mathrm{e,new}) \leftarrow \mathtt{Extend}_\mathrm{e}(G_\mathrm{e}, z_\mathrm{e,rand})$;
6      if $z_\mathrm{e,new} \neq$ NULL then
7          $Z_\mathrm{p,near} \leftarrow \mathtt{NearCapture}_\mathrm{e}(G_\mathrm{p}, z_\mathrm{e,new}, |V_\mathrm{p}|)$;
8          for all $z_\mathrm{p,near} \in Z_\mathrm{P,near}$ do
9              if $\mathtt{Time}(z_\mathrm{p,near}) \leq \mathtt{Time}(z_\mathrm{e,new})$ then
10                 $\mathtt{Remove}(G_\mathrm{e}, z_\mathrm{e,new})$;
11     $z_\mathrm{p} \leftarrow \mathtt{Sample}_\mathrm{p}(i)$;
12     $(V_\mathrm{p}, E_\mathrm{p}, z_\mathrm{p,new}) \leftarrow \mathtt{Extend}_\mathrm{p}(G_\mathrm{p}, z_\mathrm{p,rand})$ ;
13     if $z_\mathrm{p,new} \neq$ NULL then
14         $Z_\mathrm{e,near} \leftarrow \mathtt{NearCapture}_\mathrm{p}(G_\mathrm{e}, z_\mathrm{p,new}, |V_\mathrm{e}|)$;
15         for all $z_\mathrm{e,near} \in Z_\mathrm{e,near}$ do
16             if $\mathtt{Time}(z_\mathrm{p,new}) \leq \mathtt{Time}(z_\mathrm{e,near})$ then
17                 $\mathtt{Remove}(G_\mathrm{e}, z_\mathrm{e,near})$;
18     $i \leftarrow i + 1$;
19 return $G_\mathrm{e}$, $G_\mathrm{p}$
```

whether the evader can reach any of the evader vertices that lie within a certain distance to $z_\mathrm{p,new}$ in less time than the pursuer can reach $z_\mathrm{p,new}$ (Lines 14-17). Any evader vertex that is within a certain distance to $z_\mathrm{p,new}$ and that does not satisfy this requirement is removed from the tree with its descendants (Line 17). The distance scales as $\Theta(\log(n)/n)$, where n is the number of vertices in the evader's tree, G_e.

The algorithm returns two trees, namely G_e and G_p. From the evader's tree G_e, the control strategy that makes the evader reach X_goal in minimum time (if one exists) is the solution candidate after N iterations.

4 Analysis

In this section, theoretical guarantees of the algorithm are briefly outlined. Due to lack of space, detailed proofs of the results are left to a full version of this paper.

Let us note the following technical assumptions, which we will assume throughout this section without reference. Firstly, it is assumed that the dynamical systems modeling the evader and the pursuer independently satisfy local controllability properties. Secondly, we will assume that there exists a Stackelberg strategy for the pursuit-evasion game with finite value of the game L^*, and such that sufficiently small perturbations to the strategy also yield a finite value. A formal statement of these assumptions can be found (for the optimal motion planning case) in [39].

First, note the following lemma stating the optimality property of the RRT* algorithm (Algorithm 1) when the algorithm is used to solve a time-optimal kinodynamic motion planning problem. Let $G[i] = (V[i], E[i])$ denote the tree maintained by the RRT* algorithm at the end of iteration i. Given a state $z \in X$, let $T^*(z)$ denote the time an optimal collision-free trajectory reaches z, i.e., $T^*(z) := \inf_u \{T \,|\, x(T) = z$ and $\dot{x}(t) = f(x(t), u(t)), x(t) \notin X_{\text{obs}}$ for all $t \in [0, T]\}$. Let $z \in V[j]$ be a vertex that is in the tree at iteration j. The time that the unique trajectory that is in $G[i]$ for some $i \in \mathbb{N}$ and that starts from the root vertex and reaches z is denoted by $T(z)[i]$.

The following theorem follows directly from the asymptotic optimality of the RRT* algorithm shown in [39]. Let $\mu(\cdot)$ denote the Lebesgue measure.

Theorem 1 (Asymptotic Optimality of RRT* [39]). *If $\gamma > 2^d (1 + 1/d)\, \mu(X \setminus X_{\text{obs}})$, the event that for any vertex z that is in the tree in some finite iteration j the RRT* algorithm converges to a trajectory that reaches z optimally, i.e., in time $T^*(z)$, occurs with probability one. Formally,*

$$\mathbb{P}\big(\big\{ \lim_{i\to\infty} T(z)[i+j] = T^*(z), \quad \forall z \in V[j] \big\}\big) = 1, \qquad \forall j \in \mathbb{N}.$$

Let $T_\alpha(z_\alpha)[i]$ denote the time at which the vertex z_α in $V_\alpha[i]$ is reached, for $\alpha \in \{\text{e}, \text{p}\}$, and let $T^*_\alpha(z_\alpha)$ be the time the time-optimal collision-free trajectory reaches z_α (disregarding the other agent). Theorem 1 translates to the evader tree in a weaker form:

Lemma 1. *Under the assumptions of Theorem 1, applied to the evader tree,*

$$\mathbb{P}\big(\big\{ \lim_{i\to\infty} T_{\text{e}}(z_{\text{e}})[i+j] \geq T^*(z_{\text{e}}), \quad \forall z_{\text{e}} \in V[j] \big\}\big) = 1, \quad \forall j \in \mathbb{N}.$$

Lemma 1 follows directly from Theorem 1 noting that the evader's tree can only include fewer edges (due to removal of evader's vertices near capture), when compared to the standard RRT* algorithm.

A similar property can be shown in terms of capture time estimates. Given $z_{\text{e}} \in X_{\text{e}}$, define $\texttt{CaptureSet}_{\text{e}}(z_{\text{e}})$ as the set of all states in X_{p} reaching which the pursuer can capture the evader, and let $C^*(z_{\text{e}})$ denote the minimum time at which this capture can occur, i.e., $C^*(z_{\text{e}}) := \inf_{u_{\text{p}}} \big\{ T \;\big|\; x_{\text{p}}(T) \in \texttt{CaptureSet}_{\text{p}}(z_{\text{e}}) \big\}$.

Lemma 2. *Let $C_{\text{p}}(z_{\text{e}})[i] := \min \big\{ T_{\text{p}}(z_{\text{p}})[i] \;\big|\; z_{\text{p}} \in \texttt{NearCapture}_{\text{e}}(G_{\text{p}}[i], z_{\text{e}}, i) \big\}$. Then, under the assumptions of Theorem 1, applied to the pursuer tree,*

$$\mathbb{P}\big(\big\{ \lim_{i\to\infty} C_{\text{p}}(z_{\text{e}})[i] = C^*(z_{\text{e}}) \big\}\big) = 1, \quad \forall z_{\text{e}} \in X_{\text{e}}.$$

Proof (Sketch). Let the set $\texttt{DomainNearCapture}_{\text{e}}(z, n)$ be defined as $\{z_{\text{p}} \in X_{\text{p}} \,|\, \exists y \in \texttt{CaptureSet}_{\text{e}}(z), z_{\text{p}} \in \texttt{Reach}_{\text{p}}(y, l(n))\}$, where $l(n)$ was introduced in the definition of the $\texttt{NearCapture}$ procedure. Note that (i) $\texttt{DomainNearCapture}_{\text{e}}(G_{\text{p}}[i], z_{\text{e}}, i) \supseteq \texttt{CaptureSet}_{\text{e}}(z_{\text{e}})$ for all $i \in \mathbb{N}$, and (ii) $\bigcap_{i \in \mathbb{N}} \texttt{DomainNearCapture}_{\text{e}}(G_{\text{p}}[i], z_{\text{e}}, i) = \texttt{CaptureSet}_{\text{e}}(z_{\text{e}})$. Thus, the set $\texttt{DomainNearCapture}_{\text{e}}(G_{\text{p}}[i], z_{\text{e}}, i)$ converges to $\texttt{CaptureSet}_{\rm e}(z_{\rm e})$ from above as $i \to \infty$. Let $X^*_{\rm capt}(z_{\rm e})$ be the subset of $\texttt{CaptureSet}_{\rm e}(z_{\rm e})$ that the pursuer can reach within time $C^*(z_{\rm e})$. The key to proving this lemma is to show that the set $\texttt{DomainNearCapture}_{\rm e}(G_{\rm p}[i], z_{\rm e}, i)$ is sampled infinitely often so as to allow the existence of a sequence of vertices that converges to a state in $X^*_{\rm capt}$. Then, for each vertex in the sequence, by Theorem 1, the RRT* algorithm will construct trajectories that converge to their respective optimal trajectory almost surely, which implies the claim.

To show that the sets $\texttt{DomainNearCapture}_{\rm e}(G_{\rm p}[i], z_{\rm e}, i)$ are sampled infinitely often as $i \to \infty$, note that the probability that there is no sample inside the set $\texttt{DomainNearCapture}(G_{\rm p}[i], z_{\rm e}, i)$ is $(1 - \frac{\gamma_{\rm capt}}{\mu(X_{\rm p})} \frac{\log i}{i})^i$. In addition, $\sum_{i \in \mathbb{N}} \Big(1 - \frac{\gamma_{\rm capt}}{\mu(X_{\rm p})} \frac{\log i}{i}\Big)^i \leq \sum_{i \in \mathbb{N}} (1/i)^{\frac{\gamma_{\rm capt}}{\mu(X_{\rm p})}}$ is finite for $\gamma_{\rm capt} > \mu(X_{\rm p})$. Thus, by the Borel-Cantelli lemma [40], the event that there are no samples inside $\texttt{NearCapture}(G_{\rm p}[i], z_{\rm e}, i)$ occurs only finitely often with probability one; hence, the same sequence of sets is sampled infinitely often with probability one. □

The next lemma states that all vertices satisfy the soundness property.

Lemma 3. *Let B_j denote the following event: for any vertex $z_{\rm e}$ that is in evader's tree by the end of iteration j, if the pursuer can reach $z_{\rm e}$ before the evader, then $C_{\rm p}(z_{\rm e})[i]$ converges to a value that is smaller than the value that $T_{\rm e}(z_e)[i]$ converges to as i approaches infinity, i.e., $B_j := \{((C^*(z_{\rm e}) \leq T^*(z_{\rm e})) \Rightarrow (\lim_{i\to\infty} C_{\rm p}(x_{\rm e}) \leq \lim_{i\to\infty} T_{\rm e}(z_{\rm e})),\ \forall z_e \in V_e[j]\}$. Then, $\mathbb{P}(B_j) = 1$ for all $j \in \mathbb{N}$.*

Proof. Fix some $j \in \mathbb{N}$. Consider the events $\{\lim_{i\to\infty} T_e(z_e)[i+j] \geq T^*(z_e),\ \forall z_e \in V_e[j]\}$ and $\{\lim_{i\to\infty} C_p(z_e)[i+j] = C^*(z_e)\}$, both of which occur with probability one by Lemmas 1 and 2, respectively. Hence, their intersection occurs with probability one, i.e.,

$$\mathbb{P}\left(\left\{\lim_{i\to\infty} T_e(z_e)[i+j] \geq T^*(z_e) \ \wedge\ \lim_{i\to\infty} C_p(z_e)[i+j] = C^*(z_e),\ \forall z_e \in V_e[j]\right\}\right) = 1.$$

Finally, $\lim_{i\to\infty} T_e(z_e)[i+j] \geq T^*(z_e) \wedge \lim_{i\to\infty} C_p(z_e)[i+j] = C^*(z_e)$ logically implies $((C^*(z_{\rm e}) \leq T^*(z_{\rm e})) \Rightarrow (\lim_{i\to\infty} C_{\rm p}(x_{\rm e}) \leq \lim_{i\to\infty} T_{\rm e}(z_{\rm e}))$. Substituting the latter in place of the former in the equation above yields the result. □

Let $x_{\rm e}[i]$ denote the trajectory that is in evader's tree, $G_e[i]$, by the end of iteration i and that reaches the goal region in minimum time. Recall that T^* is the ending time of the minimum-time collision-free trajectory that reaches the goal region and avoids capture.

The next theorem states the probabilistic soundness of Algorithm 3. That is, the probability that any evader strategy returned by the algorithm is sound (i.e., avoids capture by the pursuer) approaches one as the number of samples increases. More precisely, for all $\varepsilon > 0$ and all $t \in [0, T^*]$, the probability that the state $x_{\rm e}[i](t)$ avoids capture, if the pursuer is delayed for ε units of time in the beginning of the game, approaches one as $i \to \infty$.

Theorem 2 (Probabilistic Soundness). *Let $A_{\varepsilon,t}[i]$ denote the event $\{t < C^*(x[i](t)) + \varepsilon\}$. Then, $\lim_{i\to\infty} \mathbb{P}(A_{\varepsilon,t}[i]) = 1$, for all $\varepsilon > 0$ and all $t \in [0,T]$.*

Proof (Sketch). Let $\mathscr{Z}[j] = \{z_1, x_2, \ldots, z_K\} \subseteq V_e[j]$ denote the set of all vertices in the evaders tree that are along the path $x_e[j]$. Let $\mathscr{T}[j] = \{t_1, t_2, \ldots, t_K\}$ denote the corresponding time instances, i.e., $z_k = x_e[t_j](t_k)$ for all $k \in \{1,2,\ldots,K\}$. By Lemma 3, the theorem holds for the time instances corresponding to the states in $\mathscr{Z}[j]$. However, it must also be shown that the same holds for all trajectories that connect consecutive states in $\mathscr{Z}[j]$. Such trajectories are referred to as intermediate trajectories from here on.

Let $t_{\max}[i] := \max_{t_k, t_{k+1} \in \mathscr{T}[i]} (t_{k+1} - t_k)$. The algorithm provided in this paper does not check the soundness of intermediate trajectories, but checks only that of the vertices. However, it can be shown that for any $\varepsilon > 0$, $\lim_{i\to\infty} \mathbb{P}(\{t_{\max}[i] < \varepsilon\}) = 1$. Roughly speaking, with probability one, the time-optimal path is never achieved, but the algorithm converges towards that optimal as the number of samples approaches infinity. Since each intermediate path that is along $x_e[j]$ is sub-optimal with probability one, in the process of convergence it is replaced with a lower cost path that includes two or more vertices of the tree in some later iteration $i > j$.

Since $t_{\max}[i] < \varepsilon$ logically implies that $t < C^*(x[i](t)) + \varepsilon$ for all $t \in [0,T]$, $\{t_{\max}[i] < \varepsilon\} \subseteq \{t < C^*(x[i](t)) + \varepsilon,\ \forall t \in [0,T]\}$, which implies $\mathbb{P}(\{t_{\max}[i] < \varepsilon\}) \leq \mathbb{P}(\{t < C^*(x[i](t)) + \varepsilon\})$. Taking the limit of both sides yields the result. □

Let us also note the following theorems regarding the probabilistic completeness and asymptotic optimality of the algorithm. The proofs of these theorems are rather straightforward and are omitted due to lack of space.

Theorem 3 (Probabilistic Completeness). *Under the assumptions of Theorem 1, Algorithm 3 finds a trajectory that reaches the goal region while avoiding collision with obstacles and capture by pursuers, if such a trajectory exists, with probability approaching one as the number of samples approaches infinity.*

Theorem 4 (Asymptotic Optimality). *Let $L[i]$ be the cost of the minimum-time trajectory in the evader's tree at the end of iteration i that reaches the goal region, if any is available, and $+\infty$ otherwise. Then, under the assumptions of Theorem 1, $L[i]$ converges to the value of the pursuit-evasion game, L^*, almost surely.*

5 Simulation Examples

In this section, two simulation examples are presented. In the first example, an evader modeled as a single integrator with velocity bounds is trying to reach a goal set, while avoiding capture by three pursuers, each of which is modeled as a single integrator with different velocity bounds. More precisely, the differential equation describing the dynamics of the evader can be written as follows:

$$\frac{d}{dt}x_e(t) = \frac{d}{dt}\begin{bmatrix} x_{e,1}(t) \\ x_{e,2}(t) \end{bmatrix} = u_e(t) = \begin{bmatrix} u_{e,1}(t) \\ u_{e,2}(t) \end{bmatrix},$$

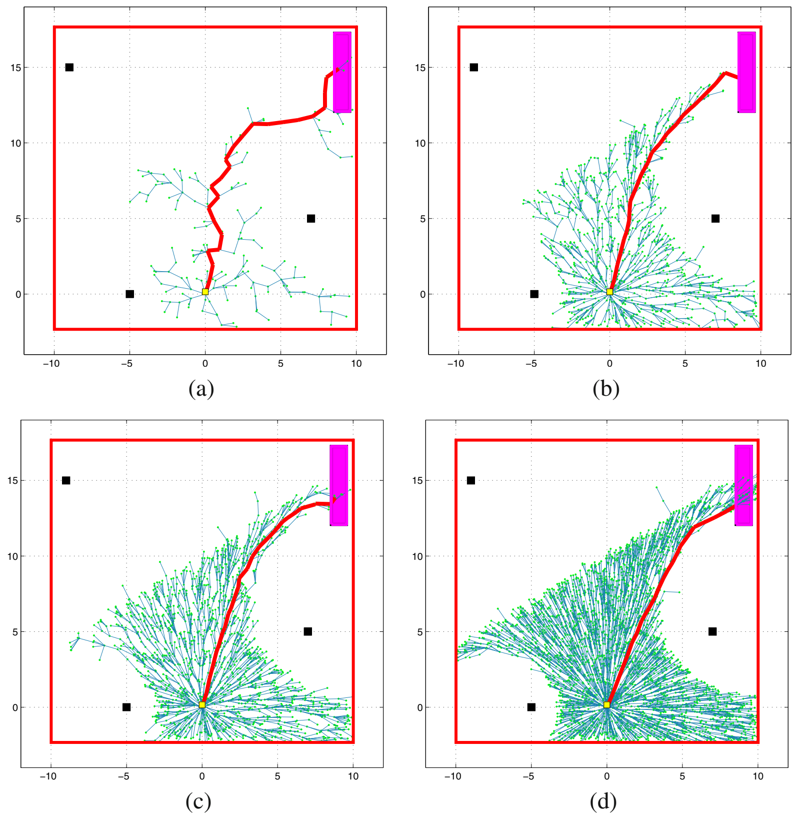


Fig. 1 The evader trajectory is shown in an environment with no obstacles at the end of 500, 3000, 5000, and 10000 iterations in Figures (a), (b), (c), and (d), respectively. The goal region is shown in magenta. Evader's initial condition is shown in yellow and the pursuers' initial conditions are shown in black. The first pursuer, P_1, which can achieve the same speed that the evader can achieve, is located in top left of the figure. Other two pursuers can achieve only half the evader's speed.

where $\|u_{\mathrm{e}}(t)\|_2 \leq 1$. The dynamics of the pursuer is written as follows:

$$\frac{d}{dt}x_{\mathrm{p}}(T) = \frac{d}{dt}\begin{bmatrix} x_{\mathrm{p}_1}(t) \\ x_{\mathrm{p}_2}(t) \\ x_{\mathrm{p}_3}(t) \end{bmatrix} = \frac{d}{dt}\begin{bmatrix} x_{\mathrm{p}_1,1}(t) \\ x_{\mathrm{p}_1,2}(t) \\ x_{\mathrm{p}_2,1}(t) \\ x_{\mathrm{p}_2,2}(t) \\ x_{\mathrm{p}_3,1}(t) \\ x_{\mathrm{p}_3,2}(t) \end{bmatrix} = u_{\mathrm{p}}(t) = \begin{bmatrix} u_{\mathrm{p}_1}(t) \\ u_{\mathrm{p}_2}(t) \\ u_{\mathrm{p}_3}(t) \end{bmatrix} = \begin{bmatrix} u_{\mathrm{p}_1,1}(t) \\ u_{\mathrm{p}_1,2}(t) \\ u_{\mathrm{p}_2,1}(t) \\ u_{\mathrm{p}_2,2}(t) \\ u_{\mathrm{p}_3,1}(t) \\ u_{\mathrm{p}_3,2}(t) \end{bmatrix},$$

where $\|u_{\mathrm{p}_1}(t)\|_2 \leq 1$ and $\|u_{\mathrm{p}_k}(t)\|_2 \leq 0.5$ for $k \in \{2,3\}$.

First, a scenario that involves an environment with no obstacles is considered. The evader's trajectory is shown in Figures 1(a)-1(d) in several stages of the algorithm. The algorithm quickly identifies an approximate solution that reaches the goal and stays away from the pursuers. The final trajectory shown in Figure 1(d) goes towards the goal region but makes a small deviation to avoid capture. The same scenario is considered in an environment involving obstacles and the evader's tree is shown in different stages in Figure 2(a)-2(d). Notice that the evader may choose to "hide behind the obstacles" to avoid the pursuers, as certain parts of the state space that are not reachable by the evader are reachable in presence of obstacles.

In the second example, the motion of the pursuer and of the evader is described by a simplified model of aircraft kinematics. Namely, the projection of the vehicle's position on the horizontal plane is assumed to follow the dynamics of a Dubins vehicle (constant speed and bounded curvature), while the altitude dynamics is modeled as a double integrator. The differential equation describing dynamics of the evader is given as follows. Let $x_\mathrm{e}(t) = [x_{\mathrm{e},1}(t), x_{\mathrm{e},2}(t), x_{\mathrm{e},3}(t), x_{\mathrm{e},4}(t), x_{\mathrm{e},5}(t)]^T$ and $f(x_\mathrm{e}(t), u_\mathrm{e}(t)) = [v_\mathrm{e} \cos(x_{\mathrm{e},3}(t)), v_\mathrm{e} \sin(x_{\mathrm{e},3}(t)), u_{\mathrm{e},1}(t), x_{\mathrm{e},5}(t), u_{\mathrm{e},2}(t)]^T$, and

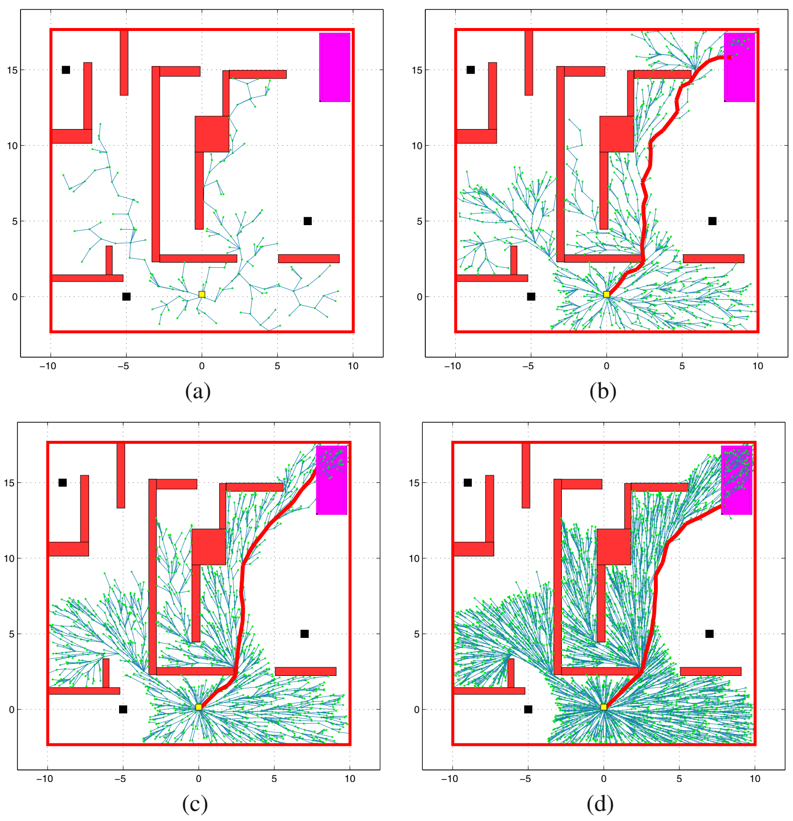


Fig. 2 The scenario in Figure 1 is run in an environment with obstacles.

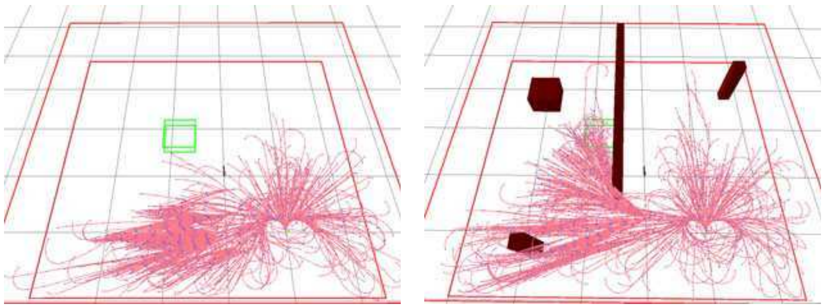

(a) (b)

Fig. 3 Figures (a) and (b) show the trees maintained by the evader at end of the 3000th iteration in an environment without and with obstacles, respectively. The initial state of the evader and the pursuer are marked with a yellow pole (at bottom right of the figure) and a black pole (at the center of the figure), respectively. Each trajectory (shown in purple) represents the set of states that the evader can reach safely (with certain probability approaching to one).

$\dot{x}_e(t) = f(x_e(t), u_e(t))$, where $v_e = 1$, $|u_{e,1}(t)| \leq 1$, $|u_{e,2}(t)| \leq 1$, $|x_{e,5}| \leq 1$. In this case, v_e denotes the longitudinal velocity of the airplane, $u_{e,1}$ denotes the steering input, and $u_{e,2}$ denotes the vertical acceleration input. The pursuer dynamics is the same, except the pursuer moves with twice the speed but has three times the minimum turning radius when compared to the evader, i.e., $v_p = 2$, $|u_{p,1}| \leq 1/3$.

A scenario in which the evader starts behind pursuer and tries to get to a goal set right next to the pursuer is shown in Figure 3. First, an environment with no obstacles is considered and the tree maintained by the evader is shown in Figure 3(a), at end of 3000 iterations. Notice that the evader tree does not include a trajectory that can escape to the goal set (shown as a green box). Second, the same scenario is run in an environment involving obstacles. The trees maintained by the evader is shown in Figure 3(b). Note that the presence of the big plate-shaped obstacle prevents the pursuer from turning left directly, which allows the evader to reach a larger set of states to the left without being captured. In particular, the evader tree includes trajectories reaching the goal.

Simulation examples were solved on a laptop computer equipped with a 2.33 GHz processor running the Linux operating system. The algorithm was implemented in the C programming language. The first example took around 3 seconds to compute, whereas the second scenario took around 20 seconds.

6 Conclusions

In this paper, a class of pursuit-evasion games, which generalizes a broad class of motion planning problems with dynamic obstacles, is considered. A computationally efficient incremental sampling-based algorithm that solves this problem with probabilistic guarantees is provided. The algorithm is also evaluated with simulation examples. To the best of authors' knowledge this algorithm constitutes the first incremental sampling-based algorithm as well as the first anytime algorithm for solving pursuit-evasion games. Anytime flavor of the algorithm provides advantage in real-time implementations when compared to other numerical methods.

Although incremental sampling-based motion planning methods have been widely used for almost a decade for solving motion planning problems efficiently, almost no progress was made in using similar methods to solve differential games. Arguably, this gap has been mainly due to the inability of these algorithms to generate optimal solutions. The RRT* algorithm, being able to almost-surely converge to optimal solutions, comes as a new tool to efficiently solve complex optimization problems such as differential games. In this paper, we have investigated a most basic version of such a problem. Future work will include developing algorithms that converge to, e.g., feedback saddle-point equilibria of pursuit-evasion games, as well as relaxing the separability assumption on the dynamics to address a larger class of problems.

References

1. Isaacs, R.: Differential Games. Wiley, Chichester (1965)
2. Basar, T., Olsder, G.J.: Dynamic Noncooperative Game Theory, 2nd edn. Academic Press, London (1995)
3. Mitchell, I.M., Bayen, A.M., Tomlin, C.J.: A time-dependent Hamilton-Jacobi formulation of reachable sets for continuous dynamic games. IEEE Transactions on Automatic Control 50(7), 947–957 (2005)
4. Lachner, R., Breitner, M.H., Pesch, H.J.: Three dimensional air combat: Numerical solution of complex differential games. In: Olsder, G.J. (ed.) New Trends in Dynamic Games and Applications, pp. 165–190. Birkhäuser, Basel (1995)
5. Vanek, O., Bosansky, B., Jakob, M., Pechoucek, M.: Transiting areas patrolled by a mobile adversary. In: Proc. IEEE Conf. on Computational Intelligence and Games (2010)
6. Simaan, M., Cruz, J.B.: On the Stackelberg strategy in nonzero-sum games. Journal of Optimization Theory and Applications 11(5), 533–555 (1973)
7. Bardi, M., Falcone, M., Soravia, P.: Numerical methods for pursuit-evasion games via viscosity solutions. In: Barti, M., Raghavan, T.E.S., Parthasarathy, T. (eds.) Stochastic and Differential Games: Theory and Numerical Methods. Birkhäuser, Basel (1999)
8. Ehtamo, H., Raivio, T.: On applied nonlinear and bilevel programming for pursuit-evasion games. Journal of Optimization Theory and Applications 108, 65–96 (2001)
9. Breitner, M.H., Pesch, H.J., Grimm, W.: Complex differential games of pursuit-evasion type with state constraints, part 1: Necessary conditions for optimal open-loop strategies. Journal of Optimization Theory and Applications 78, 419–441 (1993)
10. Breitner, M.H., Pesch, H.J., Grim, W.: Complex differential games of pursuit-evasion type with state constraints, part 2: Numerical computation of optimal open-loop strategies. Journal of Optimization Theory and Applications 78, 444–463 (1993)
11. Russell, R.D., Shampine, L.F.: A collocation method for boundary value problems. Numerische Mathematik 19(1), 1–28 (1972)
12. van Stryk, O., Bulirsch, R.: Direct and indirect methods for trajectory optimization. Annals of Operations Research 37, 357–373 (1992)

13. Sethian, J.A.: Level Set Methods and Fast Marching Methods: Evolving Surfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science. Cambridge University Press, Cambridge (1996)
14. Latombe, J.C.: Robot Motion Planning. Kluwer Academic Publishers, Boston (1991)
15. LaValle, S.: Planning Algorithms. Cambridge University Press, Cambridge (2006)
16. Petti, S., Fraichard, Th.: Safe motion planning in dynamic environments. In: Proc. Int. Conf. on Intelligent Robots and Systems (2005)
17. Hsu, D., Kindel, R., Latombe, J., Rock, S.: Randomized kinodynamic motion planning with moving obstacles. International Journal of Robotics Research 21(3), 233–255 (2002)
18. Reif, J., Sharir, M.: Motion planning in presence of moving obstacles. In: Proceedings of the 25th IEEE Symposium FOCS (1985)
19. Erdmann, M., Lozano-Perez, T.: On multiple moving objects. Algorithmica 2, 477–521 (1987)
20. Sutner, K., Maass, W.: Motion planning among time dependent obstacles. Acta Informatica 26, 93–122 (1988)
21. Fujimura, K., Samet, H.: Planning a time-minimal motion among moving obstacles. Algorithmica 10, 41–63 (1993)
22. Farber, M., Grant, M., Yuzvinski, S.: Topological complexity of collision free motion, planning algorithms in the presence of multiple moving obstacles. In: Farber, M., Ghrist, G., Burger, M., Koditschek, D. (eds.) Topology and Robotics, pp. 75–83. American Mathematical Society, Providence (2007)
23. Zucker, M., Kuffner, J., Branicky, M.: Multipartite RRTs for rapid replanning in dynamic environments. In: IEEE Conference on Robotics and Automation (ICRA) (2007)
24. van der Berg, J., Overmars, M.: Planning the shortest safe path amidst unpredictably moving obstacles. In: Workshop on Algorithmic Foundations of Robotics VII (2008)
25. Parsons, T.: Pursuit-evasion in a graph. In: Theory and Applications of Graphs, pp. 426–441 (1978)
26. Isler, V., Kannan, S., Khanna, S.: Randomized pursuit-evasion with limited visibility. In: Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, New Orleans, Louisiana, pp. 1060–1069. Society for Industrial and Applied Mathematics (2004)
27. Adler, M., Räcke, H., Sivadasan, N., Sohler, C., Vöocking, B.: Randomized pursuit-evasion in graphs. Combinatorics. Probability and Computing 12(03), 225–244 (2003)
28. Suzuki, I., Yamashita, M.: Searching for a mobile intruder in a polygonal region. SIAM Journal on computing 21, 863 (1992)
29. Gerkey, B., Thrun, S., Gordon, G.: Clear the building: Pursuit-evasion with teams of robots. In: Proceedings AAAI National Conference on Artificial Intelligence (2004)
30. Isler, V., Kannan, S., Khanna, S.: Locating and capturing an evader in a polygonal environment. Algorithmic Foundations of Robotics VI, 251–266 (2005)
31. Isler, V., Sun, D., Sastry, S.: Roadmap based pursuit-evasion and collision avoidance. In: Robotics: Science and Systems Conference (2005)
32. Kavraki, L.E., Svestka, P., Latombe, J., Overmars, M.H.: Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Transactions on Robotics and Automation 12(4), 566–580 (1996)
33. LaValle, S.M., Kuffner, J.J.: Randomized kinodynamic planning. International Journal of Robotics Research 20(5), 378–400 (2001)

34. Kuwata, Y., Teo, J., Fiore, G., Karaman, S., Frazzoli, E., How, J.P.: Real-time motion planning with applications to autonomous urban driving. IEEE Transactions on Control Systems 17(5), 1105–1118 (2009)
35. Karaman, S., Frazzoli, E.: Incremental sampling-based algorithms for optimal motion planning. In: Robotics: Science and Systems, RSS (2010)
36. Dean, T., Boddy, M.: Analysis of time-dependent planning. In: Proceedings of AAAI (1989)
37. Zilberstein, S., Russell, S.: Approximate Reasoning Using Anytime Algorithms. In: Imprecise and Approximate Computation, vol. 318, pp. 43–62. Springer, Heidelberg (1995)
38. Canny, J.: The Complexity of Robot Motion Planning. MIT Press, Cambridge (1988)
39. Karaman, S., Frazzoli, E.: Optimal kinodynamic motion planning using incremental sampling-based methods. In: IEEE Conf. on Decision and Control, Atlanta, GA (2010)
40. Grimmett, G., Stirzaker, D.: Probability and Random Processes, 3rd edn. Oxford University Press, Oxford (2001)

Multiagent Pursuit Evasion, or Playing Kabaddi

Kyle Klein and Subhash Suri

Abstract. We study a version of pursuit evasion where *two or more pursuers* are required to capture the evader because the evader is able to overpower a single defender. The pursuers must coordinate their moves to fortify their approach against the evader while the evader maneuvers to disable pursuers from their unprotected sides. We model this situation as a game of *Kabaddi*, a popular South Asian sport where two teams occupy opposite halves of a field and take turns sending an *attacker* into the other half, in order to win points by tagging or wrestling members of the opposing team, while holding his breath during the attack. The game involves team coordination and movement strategies, making it non-trivial to formally model and analyze, yet provides an elegant framework for the study of multiagent pursuit-evasion, for instance, a team of robots attempting to capture a rogue agent. Our paper introduces a simple discrete (time and space) model for the game, offers analysis of winning strategies, and explores tradeoffs between maximum movement speed, number of pursuers, and locational constraints.[1]

1 Introduction

Pursuit-evasion games provide an elegant and tractable framework for the study of various algorithmic and strategic questions with relevance to exploration or monitoring by autonomous agents. Indeed, there is a rich literature on these games under

Kyle Klein
Computer Science, UC Santa Barbara, CA 93106, USA
e-mail: kyleklein@cs.ucsb.edu

Subhash Suri
Computer Science, UC Santa Barbara, CA 93106, USA
e-mail: suri@cs.ucsb.edu

[1] A 4-page abstract announcing some of these results, without proofs and without differential speed, was presented at the 15th Canadian Conference on Computational Geometry, August 2010.

various names [13], such as *man-and-the-lion* [12, 16, 9], *cops-and-robber* [6, 1, 8, 3, 4], *robot-and-rabbit* [6], and *pursuit-evasion* [14, 5, 7], just to name a few.

In this paper we study a (discrete time, discrete space) version of pursuit evasion where *two or more pursuers* are required to capture the evader because the evader is able to overpower a single (and isolated) defender. These situations arise in pursuit of a *rogue non-cooperative agent*, which could be a malfunctioning robot, a delirious evacuee, or a noncooperative patient. Thus, the pursuers are forced to coordinate their moves to fortify their approach against the evader while the evader maneuvers to disable pursuers from their unprotected sides. In the basic formulation of the game, all agents have the same capabilities including the maximum movement speed, but we also derive some interesting results when one side can move faster than the other.

In modeling our pursuit-evasion scenario, we draw inspiration from the game of *Kabaddi*, which is a popular South Asian sport. The game involves two teams occupying opposite halves of a field, each team taking turns to send an "attacker" into the other half, in order to win points by tagging or wrestling members of the opposing team [17]. The attacker must hold his breath during the entire attack and successfully return to his own half—the attacker continuously chants "kabaddi, kabaddi, ⋯" to demonstrate holding of the breath. There are several elements of this game that distinguish it from the many other pursuit games mentioned above, but perhaps the most significant difference is that it typically requires two or more defenders to capture the opponent, while the attacker is able to capture a single isolated defender by itself. This *asymmetry* in the game adds interesting facets to the game and leads to interesting strategies and tradeoffs.

While the use of multiple pursuers is common in many existing pursuit evasion games, the main concern in those settings is to simply distribute pursuers in the environment to keep the evader from visiting or reentering a region. This is indeed the case in all graph searching [2, 11, 14] or visibility based pursuit evasion [5, 7, 15]. In the lion-and-the-man game also there are known results that show that multiple lions can capture the man when the man lies inside the convex hull of the lions [10]. By contrast, the main question in Kabaddi is whether the defenders can ever *force* the attacker inside their convex hull, perhaps even by sacrificing some of their agents. The other games such as the *cops-and-robber* differ from kabaddi in the way capture occurs as well as the *information* about the evader's position. For instance, the current position of all the players is public information in kabaddi while the position of the robber or evader is often assumed to be unknown to cops or pursuers. Furthermore, it is also typically assumed that each cop (robot) follows a fixed trajectory that is *known* to the robber (rabbit). This makes sense in situations where the defenders (cops) have fixed patrol routes, but not in interactive games like kabaddi. The problems and results in the graph searching literature are also of a different nature than ours [2, 11], although variations using differential speed [4] and capture from a distance [3] have been considered in graphs as well.

Finally, in the *visibility-based* pursuit-evasion games, the evader is often assumed to have infinite speed, and the capture is defined as being "seen" by some defender—both infinite visibility or limited-range visibility models have been considered [5, 8].

By contrast, kabaddi involves equal speed agents and requires a physical capture that leads to a very different set of strategies and game outcomes. With this background, let us now formalize our model of kabaddi.

1.1 The Standard Model

We consider a *discrete* version of the game, in which both time and space are discrete: the players take alternating turns, and move in discrete steps. In particular, the game is played on a $n \times n$ grid S, whose cells are identified as tuples (i, j), with $i, j \in \{1, 2, \ldots, n\}$. We will mainly use the Kabaddi terminology, namely, attacker and defenders, with the former playing the role of the evader and the latter the pursuers. Our main analysis will focus on the case of one attacker and two defenders, although in the latter part of the paper, we do derive some results for the case of d defenders, for any $d > 1$.

We use the letters A and D to denote the *attacker* and a *defender*, respectively. When there are multiple defenders, we use subscripts such as D_1, D_2, etc. We need the concepts of *neighborhood*, *moves*, and *capture* to complete the description of the game. Throughout, we assume that precisely two defenders are required to capture the attacker.

Neighborhood. The *neighborhood* $N(p)$ of a cell $p = (i, j)$ is the set of (at most) 9 cells, including p itself, adjacent to p, or equivalently the set of all cells with L_∞ distance at most 1 from p. In Figure 1, the neighborhood of A is shown with a box around it. Slightly abusing the notation, we will sometimes write $N(A)$ or $N(D)$ to denote the neighborhood of the current position of A or D.

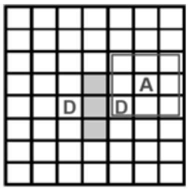


Fig. 1 The standard model of kabaddi. A can capture the defender closer to it, which is inside $N(A)$. The defenders can capture A at any position in the shaded region, which is the common intersection of their neighborhoods.

Moves. The attacker and the defenders take turns making their moves, with the attacker moving first. In one step, the attacker and the defenders can move to any cell in their neighborhood. *All the defenders can move simultaneously in one step.*

Capture. *A captures* a defender D if it is the unique defender lying inside the neighborhood of A. That is, with two defenders, D_1 is captured when $D_1 \in N(A)$ and $D_2 \notin N(A)$. (Notice that A only needs to enclose a defender within its neighborhood to capture it.)

Conversely, the defenders capture the attacker, when A lies in the common intersection of the two defenders' neighborhoods. That is, $A \in (N(D_1) \cap N(D_2))$.

Game Outcome. The attacker *wins* the game if he can capture one or more defenders, and the defenders win the game if they can capture the attacker. If neither side wins, then the game is a tie.

This particular form of capture has a tendency to make defenders always stick together, and fails to model the real world phenomenon where defenders try to "surround" the attacker—see figure above. We therefore introduce a *minimum separation* condition on the defenders in the following way:

no defender can be inside the neighborhood of another defender.

These rules together define our *standard model* of kabaddi. Other models can be obtained by varying the definition of the *neighborhood* and relaxing the separation condition for defenders, and we obtain some results to highlight the impact of these modeling variables.

Safe Return and Holding of the Breadth. In Kabaddi, the attacker must hold his breath during the attack, and after the attack successfully return to his side. These are non-trivial issues to model tractably, and we exclude them from our current model, instead relying on the following interpretation: *the worst-case number of moves before the game's outcome serves as a proxy for the breath, and the attacker can conservatively decide at some point to return to his side.* However, if this duration is known to the defenders, then they can attempt to interfere with his return. We leave these interesting, but complicated, issues for future work. One could argue that these issues are not important in the multiagent pursuit-evasion problem.

1.2 Our Results

In the case of a single attacker A against a single defender D, the game resembles the discrete version of the man-and-the-lion. We include a simple analysis of this case for two reasons: first, it serves as a building block for the multi-defender game; and second it allows us to highlight the impact of player's *speed* on the game outcome, which we believe is a new direction in pursuit evasion problems. Unsurprisingly, in the single defender case, the attacker can always capture the defender D in $O(n)$ number of steps, which is clearly optimal, upto a constant factor, in the worst-case.

We show that a speed of $1+\Theta(1/n)$ is both necessary and sufficient for the defender to indefinitely evade the attacker. In particular, a defender with the maximum speed $1+5/(\frac{n}{4}-3)$ can evade the attacker indefinitely, but a defender with the maximum speed of $1+1/n$ can be captured.

The game becomes more challenging to analyze with two defenders, where the attacker continuously runs the risk of being captured himself, or have the defenders evade him forever. Our main result is to show that the attacker has a winning strategy in worst-case $O(n)$ moves. One important aspect of the standard model is the *separation* requirement for the defenders—each must remain outside the neighborhood of the other. Without this restriction, we show that the two defenders, whom we call *strong defenders* to distinguish from the standard ones, can force a draw: neither the attacker nor a defender can be captured. A further modification of the model, which disallows the *diagonal* moves, tips the scale further in the favor of strong defenders, allowing them to capture the attacker in $O(n^2)$ steps.

Extending the analysis to more than two players is a topic for ongoing and future work, and seems non-trivial. Surprisingly, for the standard model, it is not obvious that even $\Theta(n)$ defenders can capture the attacker, nor it is obvious that the attacker can win against k defenders, for $k > 2$. (The definition of capture remains the same: two defenders are enough to capture the attacker.)

However, if we endow the agents with different speeds, then we can obtain some interesting results, as in the case of the single defender mentioned earlier. In particular, if the attacker can make $\min\{10, d-1\}$ single steps in one move, then it can avoid capture indefinitely against d defenders, and if it can make $\min\{11, d\}$ steps per move, then it can capture all d defenders in time $O(dn)$. Thus, the attacker has a winning or non-losing strategy with $O(1)$ speed against an unbounded number of players, assuming a safe initial position.

2 One on One Kabaddi

We begin with the simple case of the attacker playing against a single defender. Besides being of interest in its own right, it also serves as building block for the more complex game against two defenders. We show that in this case the attacker always has a winning strategy in $O(n)$ moves.

Throughout the paper, we assume that the grid is aligned with the axes, and use $\Delta x = |D_x - A_x|$ and $\Delta y = |D_y - A_y|$, resp., for the x (horizontal) and the y (vertical) distance between A and D.

Theorem 1. *The attacker can always capture a single defender in a $n \times n$ game of kabaddi in $O(n)$ moves.*

Proof. The attacker's basic strategy is to chase the defender towards a wall, keeping him trapped inside a continuously shrinking rectangular region. Specifically, as long as $\min\{\Delta x, \Delta y\} > 0$ on its move, the attacker makes the (unique) diagonal move towards the defender, reducing both Δx and Δy by one. Because the grid is $n \times n$, the attacker can make at most n such moves before either Δx or Δy becomes zero. Without loss of generality, suppose $\Delta x = 0$. From now on, the attacker always moves to maintain $\Delta x = 0$ while reducing Δy by one in each move. Because Δy can be initially at most n, the attacker can reduce to it one in at most $n-1$ moves, at which point it has successfully captured the defender because both Δx and Δy are at most 1. This completes the proof.

3 Attacker against Two Defenders

The game is more complex to analyze against two defenders. We begin by isolating some necessary conditions for the game to terminate, or for the next move to be safe. We then discuss the high level strategy for the attacker, and show that it can pursue the defenders using that strategy *without being captured* itself. Together with a bound for the duration of the pursuit, this yields our main result of $O(n)$ steps win for A in the standard model. We denote the two defenders by D_1 and D_2, and use D to refer to a non-specific defender when needed. Throughout the game, *we ensure that whenever A makes a move, it is safe in the sense that it cannot be captured by the defenders in their next move.*

Lemma 1. *On A's turn, if* $\max\{\Delta x, \Delta y\} \leq 2$ *for at least one of the defenders, then A can capture a defender in one step. Conversely, on the defenders' turn, if* $\max\{\Delta x, \Delta y\} > 2$ *for one of the defenders, then they cannot capture A on their move.*

Proof. We first observe that neither defender can be inside the neighborhood of A, namely, $N(A)$. This holds because a single defender inside $N(A)$ must have been captured in A's last move and if both the defenders are inside $N(A)$, then they would have captured A in their last move. Thus, we must have $\max\{\Delta x, \Delta y\} \geq 2$ for both the defenders.

Let D_1 be the defender that satisfies the conditions of the lemma, meaning that $\max\{\Delta x, \Delta y\} = 2$. If both the defenders satisfy the condition, then let us choose the one for which $\Delta x + \Delta y$ is smaller; in case of a tie, choose arbitrarily. Without loss of generality, assume that D_1 lies in the upper-left quadrant from A's position (i.e. north-west of A). We now argue that A can always capture D_1 as follows. See Figure 2.

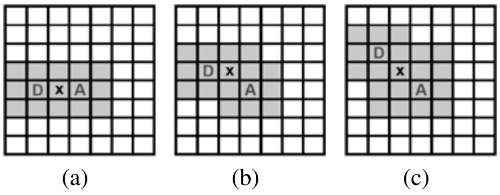


Fig. 2 Illustrating the three cases in Lemma 1: $\Delta x + \Delta y = 2$ (a), 3 (b) and 4 (c). The shaded area is the region that cannot contain the second defender.

If $\Delta x + \Delta y = 2$, then we must have either $\Delta x = 2, \Delta y = 0$ or $\Delta x = 0, \Delta y = 2$. In the former case, A can capture D_1 by moving to its x-neighbor (shown in Figure 2 (a)), and in the latter by moving to its y-neighbor. Since the second defender must lie outside $N(A) \cup N(D_1)$, this move cannot cause A to be captured. Similarly, if $\Delta x + \Delta y = 3$ (shown Figure 2 (b)), then we have either $\Delta x = 2, \Delta y = 1$, or $\Delta x = 1, \Delta y = 2$. In both cases, A captures D_1 by moving to its north-west neighbor $(A_x - 1, A_y + 1)$. Observe that, by the minimum separation rule, if there is a defender at $(A_x - 2, A_y + 1)$, then there can't be one at $(A_x - 1, A_y + 2)$, and vice versa ensuring the safety of this move—there also cannot be a defender at $(A_x, A_y + 2)$ because that would contradict the choice of the closest defender by distance.

Finally, if $\Delta x + \Delta y = 4$ (shown Figure 2 (c)), then A captures D_1 by moving to $(A_x - 1, A_y + 1)$. This is a safe move because the only position for D_2 that can capture A is at $(A_x, A_y + 2)$, but in that case D_2 is the defender with the minimum value of $\Delta x + \Delta y$, contradicting our choice of the defender to capture. This completes the first claim of the lemma. For the converse, suppose that $\Delta x > 2$ for defender D_1. Then, after the defenders' move, A is still outside the neighborhood of D_1, and so A is safe. This completes the proof.

The attacker initiates its attack by first aligning itself with one of the defenders in either x or y coordinate, without being captured in the process. The following two technical lemmas establish this.

Lemma 2. *A can move to the boundary in $O(n)$ moves without being captured.*

Proof. By assumption, A is currently safe. We first check whether A can capture a defender in the next move: if so, he wins. Otherwise, by Lemma 1, we must have that $\max\{\Delta x, \Delta y\} > 2$ for both D_1 and D_2. The attacker A now (arbitrarily) chooses a defender, say, D_1 and moves so as to increase both its x and y distances to that defender by one—this is always possible unless A is already on the boundary. Because this always maintains $\max\{\Delta x, \Delta y\} > 2$ with respect to D_1, by Lemma 1, the defenders cannot capture A, and is A guaranteed to reach the boundary in $O(n)$ steps.

Lemma 3. *By moving along the boundary, A can always force either $\Delta x = 0$ or $\Delta y = 0$ for one of the defenders in $O(n)$ moves, without being captured.*

Proof. Without loss of generality, assume that A is on the bottom boundary, and that at least one of the defenders, say, D_1 lies in its upper-right quadrant (i.e. has larger x coordinate). Then, A's strategy is to always moves right on its turn, and is guaranteed to achieve $\Delta x = 0$ with D_1 at some point. We only need to show that A cannot be captured during this phase. But if A were captured at position $(i, 0)$, then the defenders must be at positions $(i-1, j_1)$ and $(i+1, j_2)$, for $j_1, j_2 \in \{0, 1\}$—these are the only positions whose neighborhoods contain the cell $(i, 0)$ in common. However, the position of A one move earlier was $(i-1, 0)$, so the first defender would necessarily satisfy the conditions of Lemma 1 and would have been captured by A already.

3.1 The Second Phase of the Attack

Having reached the starting position for this second phase of the game, we assume without loss of generality that A is at the bottom boundary, and that after A's last move, $\Delta x = 0$ for one of the defenders. From now on, A will always ensure that $\Delta x \leq 1$ for one of the defenders *after each of A's moves*. The x-distance can become $\Delta x = 2$ *after the defenders' move* but A will always reduce it to 1 in its next move.

By Lemma 1, if both Δx and Δy are at most 2, then A can win the game. On the other hand, if the players are too far apart, then both sides are safe for the next move. Thus, all the complexity of the game arises when the distance between A and

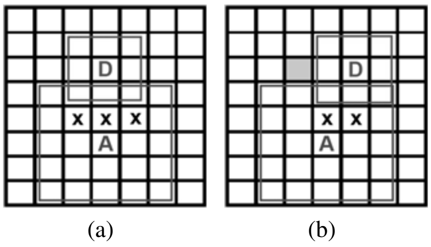


Fig. 3 Proofs of Lemmas 4 (a) and 5 (b).

the defenders is 3, requiring careful and strategic moves by both sides. We show that A can always follow an attack strategy that ensures a win in $O(n)$ steps, while avoiding capture along the way.

In order to measure the *progress* towards A's win, we use the distance from A's current position to the top boundary of the grid *while ensuring that* $\Delta x \leq 1$ *continues to hold*. In particular, define $\Phi(A)$ as the gap between the current y position of A and the top boundary. That is, $\Phi(A) = (n - A_y)$, where this gap is exactly $n-1$ when the second phase begins with A on the bottom boundary. We say that A *makes progress* if $\Phi(A)$ shrinks by at least 1, while Δx remains at most 1 for some defender. Clearly, when the $\Phi(A)$ reaches one, A has a guaranteed win (by Lemma 1). If the attacker succeeds in capturing a defender, then we consider that also progress for the attacker.

The overall plan for our analysis is the following:

1. If $\max\{\Delta x, \Delta y\} \leq 2$ for at least one defender, then the attacker wins in one move (Lemma 1). If $\Delta y > 3$ for some defender, then A can move to reduce Δy by one, while keeping $\Delta x \leq 1$, and this move is safe by Lemma 1. Thus, the only interesting cases arise when $\Delta y = 3$; these are handled as follows.
2. If $\Delta y = 3$ and $\Delta x = 0$ for some defender, then Lemma 4 below shows that A makes progress in $O(1)$ number of moves.
3. If $\Delta y = 3$ and $\Delta x = 1$ or 2 for some defender, then Lemmas 5 and 6 show that A can make progress in $O(n)$ number of moves.

In the following, we use the notation $N^2(p)$ to denote the 2-*neighborhood* of a cell p, meaning all the positions that can be reached from p in two moves.

Lemma 4. *On A's move, if* $\Delta y = 3$ *and* $\Delta x = 0$ *holds for some defender, then A makes progress in one move.*

Proof. Figure 3 (a) illustrates the game configuration for this case, where the defender satisfying the distance condition $\Delta x = 0, \Delta y = 3$ is shown as D. There are three positions for A to advance and make progress, and they are marked as x in the figure—in each case, the y distance reduces by 1, while Δx remains at most 1. We only need to show that A can move to one of these positions without being captured itself.

In order to prove this, we observe that (1) neither defender is currently inside $N^2(A)$ because that is a winning configuration for A by Lemma 1; (2) the second defender is not in $N(D)$, as required by the separation rule for defenders. Thus, the second defender must be outside $N^2(A) \cup N(D)$. But in order to foil A's move to all three x positions, the second defender must also be within the 2-neighborhood of *all* the x positions. That, however, is impossible, as is readily confirmed by inspection of Figure 3 (a). Thus, A can safely move to one of the positions marked as x, and guarantee progress. We note that when A and D are on the boundary, there are two x positions instead of three, and in that case A can always move to the x directly north and make progress.

Lemma 5. *On A's move, if $\Delta y = 3$ and $\Delta x = 1$ holds for some defender, then A makes progress in $O(n)$ number of moves.*

Proof. Figure 3 (b) illustrates the game configuration for this case, where the defender satisfying the distance condition $\Delta x = 1, \Delta y = 3$ is shown as D. (We assume without loss of generality that $D_x = A_x + 1$ because the case $D_x = A_x - 1$ is entirely symmetric.) In this case, there are two positions marked x that allow A to make progress by reducing Δy. In order to foil A's move, the second defender must be positioned so as to cause A's capture at both these positions. Reasoning as in the previous lemma, however, D_2 has to lie outside both $N(D)$ as well as $N^2(A)$. It is easy to see that there is precisely one position for D_2, shown as the shaded cell, that threatens A's capture at both the x positions.

This is a case where A cannot ensure progress in a single step, and instead a multi-step argument is needed. In particular, A moves to its right neighboring cell, at location $(A_x + 1, A_y)$, which does not improve $\Phi(A)$, but we show that $\Phi(A)$ *will* improve in $O(n)$ steps. Consider the next move of the defenders. The defender labeled D must move to a cell within $N(D)$, and we analyze the progress by A as follows: (i) if D moves up, making its distance from A equal to $\Delta y = 4$, then the next move of A makes a guaranteed progress by moving to make $\Delta y = 3$ and $\Delta x \leq 1$. This move is safe for A by Lemma 1. (ii) if D moves down, making its distance from A equal to $\Delta y = 2$, then, A has a guaranteed win according to Lemma 1. (iii) if D stays in its current cell, then we have $\Delta y = 3$ and $\Delta x = 0$ on A's move, for which Lemma 4 guarantees progress in one move.

Thus, the only interesting cases are if D moves to its left or right neighbor. If D moves left, causing $\Delta y = 3$ and $\Delta x = 1$, then A can immediately make progress because both the defenders are on the left side of A's position (recall that A was forced to make a move without progress because the second defender was in the shaded cell), and so A can safely move diagonally to reduce both Δx and Δy distances to D. In this case we have progress in a total of 2 moves.

On the other hand, if D moves to its right neighbor, then the situation of impasse can persist, because both positions marked x where A can make progress can cause A to be captured. This forces A to continue to mimic D's rightward move by moving to its right neighbor. However, this impasse can continue only for $O(n)$ moves because as soon as D reached the right boundary of the field, he is forced to move up, down, or left, giving A a chance to make progress. This completes the proof of the lemma.

Lemma 6. *If $\Delta y = 3$ and $\Delta x = 2$ for some defender say D_1 then A may make progress in $O(n)$ moves.*

Proof. The proof is similar to the proof of Lemma 5, and omitted due to lack of space.

3.2 Completing the Analysis

We can now state our main theorem.

Theorem 2. *In the standard model of kabaddi on a $n \times n$ grid, the attacker can capture both the defenders in $O(n)$ worst-case moves.*

Proof. We show that, starting from an initial safe position, the attacker always has a move that keep him safe for the next move of the defenders, and that after $O(n)$ moves the attacker can place itself on a boundary with either $\Delta x = 0$ or $\Delta y = 0$ for some defender. Without loss of generality, suppose the attacker reaches the bottom boundary, with $\Delta x = 0$ (Lemmas 2, 3). In the rest of the game, the attacker always maintains $\Delta x \leq 1$ after each of its moves. The attacker's next move is described as follows:

1. If $\max\{\Delta x, \Delta y\} \leq 2$ for some defender, then the attacker can capture a defender in 1 move (Lemma 1, and the remaining defender in $O(n)$ moves.
2. If $\Delta y \geq 4$, then the attacker always moves to reduce Δy and Δx by one, unless Δx is already zero.
3. If $\Delta y = 3$, then depending on whether $\Delta x = 0, 1$ or 2, the attacker's strategy is given by Lemma 4, 5 or 6, respectively.

These cases exhaust all the possibilities, and as argued earlier, the attacker can reduce $\Phi(A)$ by one in $O(n)$ moves. Since the maximum possible value of $\Phi(A)$ is initially $n-1$, and it monotonically decreases, we must reach $\Phi(A) = 1$ in worst-case $O(n^2)$ moves, terminating in a win by A.

We now argue that the $O(n^2)$ bound is pessimistic and that $O(n)$ moves suffice. The key idea is that once the attacker forces $\Delta x = 0$, it only moves to the three cells above it and the one to its right. The three upward moves clearly cause progress, so we only need to argue that the rightward moves happen $O(n)$ times. This follows because the grid has width n, and therefore after at most $n-1$ rightward moves, every additional rightward move must be preceded by some leftward move. Since the attacker always moves upward in its left-directed moves, it makes progress in each of those moves. Then due to the fact A only needs $n-2$ upward moves, there can be at most $2n-3$ right moves (the initial $n-1$ moves plus the $n-2$ moves corresponding to upward moves), and thus at worst $3n-5$ total moves. Thus the attacker captures both defenders in $O(n)$ worst case moves. This completes the proof of the theorem.

4 Strong Defenders

In the standard model, each defender must remain outside the neighborhood of other defenders; that is, $D_i \notin N(D_j)$, for all i, j. The defenders become more powerful when this requirement is taken away. Let us call these *stronger* defenders. In this case we explore what happens when we remove the stipulation that the defenders cannot be within each other's neighborhoods. This creates two stronger defenders and as a result creates a game where ideal play means not only can the attacker not win, but the defenders cannot either. We assume that play starts with defenders already in a side-by-side position, that is, $\Delta x + \Delta y = 1$ with respect to D_1 and D_2's coordinates.

Theorem 3. *Under the* strong *model of defenders, there is a strategy for the defenders to avoid capture forever. At the same time, the attacker also has a strategy to avoid capture.*

Proof. We first argue that the attacker can evade capture. Suppose that the defenders were to capture A in their *next* move. If neither defender is inside $N^2(A)$, then A is clearly safe in its current position for the defenders' next move, so at least one of the defenders, say D_1 is inside $N^2(A)$. Unless $D_2 \in N(D_1)$, by Lemma 1, then A can capture D_1 in its next move. Thus, D_1, D_2 must be adjacent, namely, in each other's 1-neighborhoods.

We now argue that all defender positions from which they can capture A in the next move are *unsafe*, meaning the attacker can capture one of the defenders in its current move. There are only two canonical positions for the defenders with one or both of them in the outer cells of $N^2(A)$: either side-by-side, or diagonal from one another. In the first case, the defenders only threaten the cells in front of them but not those that are diagonal, so A can move to one of those diagonal spaces. In the second case, A can capture by moving to any space diagonal from a defender.

Similarly, we can show that defenders can also avoid capture. Figure 4 shows a representative situation just before the defenders' move. Suppose that the attacker were to capture one of the defenders in its *next move*. We claim that the cells marked as A in the figure are the only places (upto symmetry) for the attacker's *current* position—i.e. these are the positions where A is not captured currently but can capture a defender by moving to the cells shown shaded. This is found by taking the union of the 1-neighborhoods of the three shaded spots (the only places A captures a defender) to find all possible places A may move to capture from, then removing all those that the defenders could capture. This result in a list of spots D cannot capture but must avoid capture from. However, the defenders can avoid this capture by simply "flipping" their orientation, as shown by arrows in the figure. Notice that after the flip the attacker now cannot capture with its move. Also the flip does not rely on the position of the boundary, as the defenders move up only if the attacker is above them, and move down only if the attacker is below them. Thus this can be performed regardless of location.

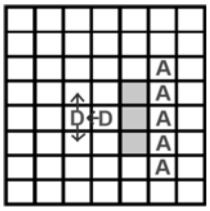


Fig. 4 Illustrates Theorem 3.

5 Strong Defenders with Manhattan Moves

Thus, in the standard model but with strong defenders, we have a tie, and neither side can guarantee a win. In the following, we show that if we disallow the *diagonal moves*, permitting a player to move only to its left, right, up, and down neighbors, then the defenders have a winning strategy. That is, the movement metric is Manhattan metric—a player can only move to a cell within the L_1 distance of 1 from its current cell. The definition of the capture, however, remains the same as in the standard model. Due to lack of space, the proof of the following theorem is omitted from this extended abstract.

Theorem 4. *Two strong defenders playing under the Manhattan moves model can always capture the attacker in $O(n^2)$ moves.*

6 Differential Speed Pursuit Evasion

So far, we have assumed that all players have the same (unit) speed. While we are unable to resolve the outcome of these games when the attacker plays against more than two defenders, we show below that *differential speed* leads to some interesting results. We model the speed as the *number* of unit-step moves a player can make on its turn—each step is the same elementary move used in the standard model. In particular, *on its turn, a player with speed s can repeat the following s times, starting at a cell $p = p_0$:*

$$\textit{move to any cell } p' \in N(p), \textit{ and set } p = p'.$$

We allow the speed to be any rational number. Thus, a player with movement speed $s + \frac{p}{q}$ can make s unit step moves on each turn *plus* it can make $s+1$ steps on every $\lfloor q/p \rfloor$th turn. Please note that this definition is not the same as being able to move to a cell at distance at most s—specifically, our attacker has a chance to visit, and possibly capture, s defenders in a single move. *However, during his turn, if the attacker is ever in the common intersection of two defenders' neighborhoods, then it is captured (as in the standard model).*

We first consider the minimum speed advantage needed by a single defender to escape the attacker forever.

6.1 One on One Game with Speedier Defender

The following theorem shows that a speed of $1 + 1/n$ is not enough for the single defender to evade capture by the attacker.

Theorem 5. *A defender with maximum speed $1 + \frac{1}{n}$ can be captured in $O(n)$ moves by an unit-speed attacker on the $n \times n$ grid.*

Proof. The attacker's strategy is the same as in Theorem 1. We simply observe that despite the speed disadvantage the attacker still reduces either Δx or Δy to zero within n moves. Without loss of generality, assume that Δx becomes zero. After that, the attacker can also enforce $\Delta y = 0$ within n moves. In these n moves, the defender gains only one extra move, which only increases Δx to 1, but is still sufficient for the capture. Thus the defender is captured in $O(n)$ moves by the attacker.

Surprisingly, it turns out that a speed of $1 + \Theta(1/n)$ suffices for the defender to escape, as shown in the following theorem.

Theorem 6. *A defender with maximum speed $1 + 5/(\frac{n}{4} - 3)$ can indefinitely evade the attacker on an $n \times n$ grid.*

Proof. Assume an initial placement of the two agents in which (1) the defender D is at least distance $n/4$ from its closest boundary, which we assume to be the bottom boundary, (2) A is distance $\frac{n}{4} + 3$ from the same boundary, and (3) $\Delta x + \Delta y = 3$. (The defender can easily enforce the condition $\Delta x = 0$, and the remaining conditions are to achieve a safe initial separation between the attacker and the defender.) We argue that the defender can successfully maintain these conditions, and when needed use its extra moves to reestablish them with respect to a different boundary.

The defender's strategy now is to simply mimic the moves of the attacker as long as it can do this without running into a boundary. During these moves, the defender is safe because of the condition $\Delta y = 3$ or $\Delta x = 3$ (cf. Lemma 1).

Since the defender D is at least $n/4$ away from the boundary that is opposite the attacker, its speed advantage guarantees it 5 extra steps before it can no longer mimic a move of the attacker—which can only happen due to running into a boundary. We now assert that the 5 extra moves are sufficient for D to reestablish the starting conditions without being captured. This is illustrated in Figure 5 (a), where only a small portion of the grid surrounding the players is shown for clarity. With its 5 moves (shown labeled $1, 2, \ldots, 5$), the defender is able to restore the initial condition with respect to the right boundary. During this maneuver, the defender maintains a safe distance from A, and therefore is not captured.

Of course, the defender earns its five extra moves *gradually*, and not at once, but it is easy to see that the defender can plan and execute these extra moves (amortize, so to speak) during the at least $n/4$ moves it makes mimicking A, as it earns them. In particular, D always "rotates" around the attacker in the direction of the farther of the two boundaries, which must be at distance at least $n/2$. D cannot run into a boundary because the closest one is at least $n/4$ away and it completes its rotation in $\frac{n}{4} - 3$ turns, during which the 5 extra moves will never decrease the defender's

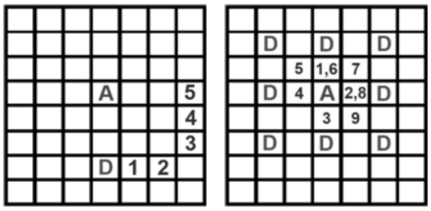


Fig. 5 Figure (a) illustrates the proof of Theorem 6: the defender uses 5 extra moves to reestablish the initial conditions. Figure (b) illustrates Theorem 7: the attacker can capture seven of the maximum possible eight defenders using 9 steps, and return to its original position in the 10th step.

distance to that boundary. The new target boundary is at least $n/2$ away and once the attacker finishes its rotation must still be $n/4$ away. This is because there are at most $\frac{n}{4} - 3$ moves in this direction resulting from moves mimicking A and the three additional moves from the rotation. Thus, after the rotation, the defender is $n/4$ away from a boundary and the attacker is $\frac{n}{4} + 3$ from the same boundary, with A and D both in the same row or column. Thus the defender can continue this strategy forever and avoid capture.

6.2 Speedier Attacker against Multiple Defenders

We now consider the speed advantage of attacker against multiple defenders. We showed earlier that in the standard model, the unit-speed attacker wins against two unit-speed defenders. However, the game against more than two defenders remains unsolved. In the following we show that with a constant factor speed advantage, a single attacker can win against any number of defenders.

Theorem 7. *An attacker with speed s can indefinitely avoid capture against $s+1$ defenders, for $s < 10$. An attacker with speed $s = 10$ can avoid capture against any number d of defenders.*

Proof. Let us first consider $s < 10$. The attacker follows a lazy strategy, which is to sit idly unless it is in danger of being captured in the defenders' next turn. Specifically, if no defenders are in $N^2(A)$, the attacker is safe (by Lemma 1). If some defenders enter $N^2(A)$, then the attacker can capture the defender closest to it using Lemma 1, in a single elementary step, with $s-1$ steps (and at most s defenders) remaining before his turn is up. We repeat the argument from the new location of A, until either A is safe for the next turn of the defenders, or it has captured all but one defenders. Thus, either A can remain safe indefinitely, or if only one defender remains it can win.

When $s \geq 10$, we note that due to the minimum separation constraint among the defenders, at most 8 defenders can simultaneously exist inside $N^2(A)$—clearly, no defender lies in $N(A)$ because that is already a captured position, and there are 16 cells in $N^2(A) - N(A)$, and no two consecutive ones can have defenders in them. Figure 5 (b) shows A's strategy to capture seven of the maximum possible eights defenders in nine steps, and then return to its original position in the 10th step. It is easy to check that the attacker can achieve a similar result for any configuration of fewer than eight defenders.

The following theorem, whose proof is omitted due to lack of space, shows that an additional increase of speed allows the attacker to capture, and not just evade, any number of defenders.

Theorem 8. *An attacker with speed $s \leq 10$ can capture s or fewer defenders in $O(sn)$ turns. An attacker with speed $s = 11$ can capture any number d of defenders in $O(dn)$ turns.*

7 Discussion

We considered a pursuit-evasion game in which two pursuers are required to capture an evader. We modeled this game after Kabaddi, which introduces a new and challenging game of physical capture for mathematical analysis. We believe that Kabaddi offers an elegant and useful framework for studying attack and defensive moves against a team of opponents who can strategically coordinate their counterattacks. Our analysis shows that even with two defenders the game reveals significant complexity and richness.

Our work poses as many open questions as it answers. Clearly, in order to obtain our initial results, we have made several simplification in the game of Kabaddi. While these simplifications do not affect the relevance of our results to multiagent pursuit-capture, they are crucial for a proper study of kabaddi. The most significant among them is the proper modeling of "holding the breadth" and "safe return." Among the more technical questions, analyzing the game for more then two defenders remains open in the standard model. The minimum separation rule leads to some pesky modeling problems because the attacker could sit in a corner cell and not be captured. So some modification is needed in the rules to avoid such deadlocks. Finally, we have not addressed the game when more than two defenders are required for the capture.

References

1. Aigner, M., Fromme, M.: A game of cops and robbers. Discrete Applied Mathematics 8(1), 1–12 (1984)
2. Bienstock, D., Seymour, P.: Monotonicity in graph searching. J. Algorithms 12(2), 239–245 (1991)
3. Bonato, A., Chiniforooshan, E., Pralat, P.: Cops and robbers from a distance. Theoretical Computer Science (in press, 2010)
4. Frieze, A., Krivelevich, M., Loh, P.-S.: Variations on cops and robbers (unpublished manuscript)

5. Guibas, L.J., Latombe, J.-C., LaValle, S.M., Lin, D., Motwani, R.: Visibility-based pursuit-evasion in a polygonal environment. IJCGA 9(5), 471–494 (1999)
6. Halpern, B.: The robot and the rabbit–a pursuit problem. The American Mathematical Monthly 76(2), 140–145 (1969)
7. Isler, V., Kannan, S., Khanna, S.: Randomized pursuit-evasion with local visibility. SIAM J. Discrete Math. 20(1), 26–41 (2006)
8. Isler, V., Karnad, N.: The role of information in the cop-robber game. TCS 399(3), 179–190 (2008)
9. Karnad, N., Isler, V.: Lion and man game in the presence of a circular obstacle. In: IROS, pp. 5045–5050 (2009)
10. Kopparty, S., Ravishankar, C.V.: A framework for pursuit evasion games in r^n. Information Processing Letters 96(3), 114–122 (2005)
11. LaPaugh, A.: Recontamination does not help to search a graph. J. ACM 40(2), 224–245 (1993)
12. Littlewood, J.E.: Littlewood's Miscellany. Cambridge University Press, Cambridge (1986)
13. Nahin, P.J.: Chases and Escapes. Princeton University Press, Princeton (2007)
14. Parsons, T.D.: Pursuit-evasion in a graph. In: Alavi, Y., Lick, D.R. (eds.) Theory and Application of Graphs, pp. 426–441. Springer, Heidelberg (1976)
15. Sachs, S., Rajko, S., LaValle, S.M.: Visibility-based pursuit-evasion in an unknown planar environment. International Journal of Robotics Research 23(1), 3–26 (2004)
16. Sgall, J.: Solution of david gale's lion and man problem. Theor. Comput. Sci. 259(1-2), 663–670 (2001)
17. Wikipedia. Kabaddi (2009), `http://en.wikipedia.org/wiki/Kabaddi`

Reconfiguring Chain-Type Modular Robots Based on the Carpenter's Rule Theorem

Jungwon Seo, Steven Gray, Vijay Kumar, and Mark Yim

Abstract. Reconfiguring chain-type modular robots has been considered a difficult problem scaling poorly with increasing numbers of modules. We address the reconfiguration problem for robots in 2D by presenting centralized and decentralized algorithms based on the Carpenter's Rule Theorem [4]. The theorem guarantees the existence of instantaneous collision-free unfolding motions which monotonically increase the distance between all joint pairs until an open chain is straightened or a closed chain is convexified. The motions can be found by solving a convex program. Compared to the centralized version, the decentralized algorithm utilizes local proximity sensing and limited communications between subsets of nearby modules. Because the decentralized version reduces the number of joint pairs considered in each convex optimization, it is a practical solution for large number of modules.

1 Introduction

Forming shapes from groups of robotic modules is a goal for many *Modular Self-reconfigurable Robots (MSRs)* and *Self-assembling Structures*. Such approaches often utilize modules with nice space-filling properties [9, 23, 19]. The modules rearrange themselves to form shapes that suit the task at hand [23, 8, 7]. In addition to the mechanical issues inherent in building a system that has a desired shape and bonding mechanisms, research has focused on motion planning for these modules. The problem in this context is to determine collision-free motions for the modules to rearrange from an initial configuration to a goal configuration.

There are three classes of MSRs based on the style of reconfiguration: chain, lattice, and mobile [22]. Chain reconfiguration involves forming chains of arbitrary numbers of modules [21, 15] which may break apart, combine into larger chains,

Jungwon Seo · Steven Gray · Vijay Kumar · Mark Yim
Department of Mechanical Engineering and Applied Mechanics, University of Pennsylvania
e-mail: {juse,stgray,kumar,yim}@seas.upenn.edu

or join face-to-face. This class of reconfiguration involves long chains with up to n degrees-of-freedom (n is the number of modules); planning for chains requires self-collision detection as well as forward and inverse kinematic computations which scale poorly as n increases. Randomized path planning techniques have been applied to this type of reconfiguration [2, 16]. Lattice reconfiguration involves modules that sit on a lattice while other modules move around each other to neighboring lattice positions. Moving modules from one location to another has been well-addressed in the literature [12, 17, 13]. Lastly, mobile reconfiguration uses the environment to maneuver and has primarily been explored in stochastic fluidic systems [18].

Modules that are permanently connected by joints can be folded to form relatively strong structures, as permanent joints can be made stronger than bonds that must be able connect and disconnect. Such modules are useful for applications involving large internal forces (e.g., a reconfigurable wrench). Achieving desired mechanical properties for shapes like the wrench is a goal of programmable matter [24]. While one might suspect that requiring modules to maintain a permanently connected chain would limit the possible configurations, it has been shown that any 2D shape can be formed by folding a sufficiently long chain of diagonally connected squares [8, 9]. In three dimensions, *origami* demonstrates the versatility of permanently connected folded shapes. Robotic folded sheets have been shown in [10, 8]. Whereas origami uses uncut sheets of flat material, this work focuses on module chain that can be folded into larger structures.

2 Preliminaries

2.1 Carpenter's Rule Theorem and the CDR Algorithm

Consider a linkage of rigid line segments connected at revolute joints to form an open chain or a closed chain on the plane. The *Carpenter's Rule Theorem* states that every *simple* open chain can be straightened and every *simple* closed chain can be convexified in such a way that the links do not cross or touch [4].

Let $\mathbf{p} = (\mathbf{p}_1^T \quad \mathbf{p}_2^T \quad \cdots \quad \mathbf{p}_n^T)^T$ denote a configuration of a simple chain of n joints by specifying joint coordinates in the plane, $\mathbf{p}_i = (p_{ix} \quad p_{iy})^T$. An example is shown in Fig. 1(a). For open chains, $\mathbf{p}_1$ and $\mathbf{p}_n$ refer to the two unary joints at the ends. For closed chains, n-joints correspond to n-vertices of the simple n-gon. The configuration space P is defined as a collection of all such configurations. Thus, when joints $\mathbf{p} \in P$ are connected in order, the chain is neither *self-touching* nor *self-crossing*. Note that we will factor out rigid transformations by pinning down a link.

We now summarize the result by Connelly, Demaine, and Rote (*The CDR Algorithm*) [4]. Assume that none of the joint angles (the smaller of two angles at $\mathbf{p}_i$) is π. Consider the following convex program with respect to $\mathbf{v} = (\mathbf{v}_1^T \quad \mathbf{v}_2^T \quad \cdot \quad \mathbf{v}_n^T)^T$, where $\mathbf{v}_i = (v_{ix} \quad v_{iy})^T$ is the instantaneous velocity of $\mathbf{p}_i$.

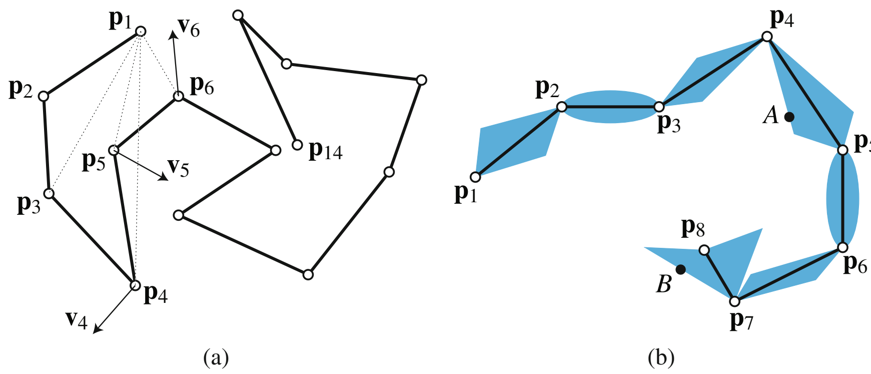


Fig. 1 (a) An open chain of line segments (*bars*) with 14 joints. Dashed lines represent some of the *struts* connected to joint 1. (b) An open chain of rigid bodies. Each base link (black line segment) has *slender adornments* except for the last link. For example, when A traverses the adornment boundary from $\mathbf{p}_4$ to $\mathbf{p}_5$, the distance between $\mathbf{p}_4$ ($\mathbf{p}_5$) and A increases (decreases) monotonically. The distance between $\mathbf{p}_7$ ($\mathbf{p}_8$) and B, however, does not increase (decrease) monotonically.

$$\text{minimize} \sum_i \|\mathbf{v}_i\|^2 + \sum_{\{i,j\}\in S_{original}} \frac{1}{(\mathbf{v}_i - \mathbf{v}_j)\cdot(\mathbf{p}_i - \mathbf{p}_j) - \|\mathbf{p}_j - \mathbf{p}_i\|} \tag{1}$$

$$\text{subject to } (\mathbf{v}_j - \mathbf{v}_i)\cdot(\mathbf{p}_j - \mathbf{p}_i) > \|\mathbf{p}_j - \mathbf{p}_i\| \text{ , for } \{i,j\} \in S_{original} \tag{2}$$

$$(\mathbf{v}_j - \mathbf{v}_i)\cdot(\mathbf{p}_j - \mathbf{p}_i) = 0 \text{ , for } \{i,j\} \in B_{original} \tag{3}$$

$$\mathbf{v}_1 = \mathbf{v}_2 = 0 \tag{4}$$

The set $S_{original}$ of *struts* is a collection of all joint pairs $\{i,j\}$ not connected to the same rigid bar and the set $B_{original}$ of *bars* contains only joint pairs attached to the same rigid bar. Since $(\mathbf{v}_j - \mathbf{v}_i)\cdot(\mathbf{p}_j - \mathbf{p}_i)$ can be related to the time rate of change of $||\mathbf{p}_j - \mathbf{p}_i||$, the above formulation asks if one can find an instantaneous motion where every joint $\mathbf{p}_k$ moves away from all other joints except for $\mathbf{p}_{k-1}$ and $\mathbf{p}_{k+1}$, the joints connected to $\mathbf{p}_k$ by rigid bars. The Carpenter's Rule Theorem proves that this convex program always has a feasible solution until any joint angle reaches π. In other words, the theorem verifies the existence of instantaneous collision-free *unfolding* (straightening or convexifying) motions. (2) shows that stronger constraints where we expand struts at a rate of at least unity are also feasible. (Consider $(\mathbf{v}_j - \mathbf{v}_i)\cdot\frac{(\mathbf{p}_j - \mathbf{p}_i)}{\|\mathbf{p}_j - \mathbf{p}_i\|}$.) The motion predicted by above formulation is called a *global-scale strictly expansive motion*.

Moreover, the solution to Eqns. (1 - 4), $f(\mathbf{p})$, is differentiable in a neighborhood Q of $\mathbf{p}$, where Q is a collection of configurations around $\mathbf{p}$ in which any joint angle is not π. Thus an unfolding path $\mathbf{p}(t)$ can be obtained by solving the dynamical system, $\dot{\mathbf{p}} = \mathbf{v} = f(\mathbf{p})$. Whenever any joint angle reaches π, the two adjacent links must be merged and a new unfolding path for the modified linkage must be obtained at that time. Finally, given a configuration $\mathbf{p}_c$, we get $\mathbf{p}(t) : [0,T] \subset \mathbb{R} \to P$ such that $\mathbf{p}(0) = \mathbf{p}_c$ and $\mathbf{p}(T) = \mathbf{p}_0$, where $\mathbf{p}_0$ is the straightened or convexified configuration.

2.2 Slender Adornment

Extending the Carpenter's Rule Theorem to linkages of rigid bodies requires a guarantee of universal foldability. A family of planar shapes referred to in the literature as having the property of *slender adornment* [5] provides this guarantee. Slender adornment is defined such that the distance between each joint and a point moving along the exterior of the rigid body link changes monotonically (Fig. 1(b)).

According to [5], we can regard each *slender* link as a line segment (*base*) connecting two revolute joints such that the whole chain system can be treated simply as a mathematical linkage (*base linkage*) for finding an unfolding path using global-scale strictly expansive motions.

3 Problem Description and Main Contribution

We shall investigate how to reconfigure a chain-type, modular self-reconfigurable robot moving on a plane to move between any two shapes while maintaining connections between modules and avoiding collisions. The system can be thought of as a serial chain (open or closed) of bodies connected by revolute joints. Joint connectivities are fixed to facilitate development of physical prototypes.

Specifically, we shall focus on chains of either line segments or cubes (squares in 2D). The former can abstract many useful systems such as robot manipulators, and the latter is particularly interesting for its space-filling property which enables us to represent interesting 2D shapes. As used in the previous section, *unfolding* straightens or convexifies complicated shapes while *folding* "complicates" shapes. We reconfigure by repeating folding and unfolding.

The main contribution of this paper is the decentralized reconfiguration planning based on the Carpenter's Rule Theorem [4, 5]. Our result allows decoupled planning for a class of articulated robot, something long considered infeasible due to interdependencies of the motions (page 390 in [11]). Compared to the current state-of-the-art in planning methods, for example, probabilistic algorithms, our algorithms do not need to build a roadmap *a priori* nor do they need specialized parametrization to handle closed loops [16, 20]. The formalism from Sec. 2 naturally handles closed loops. For example, in Fig. 1(a), a closed chain can be easily modeled by adding a bar between links 1 and 14. The bar is added by adding joint pair $\{1,14\}$ to $B_{original}$ and deleting the pair from $S_{original}$. The need to check a randomly generated configuration for collisions, often the most costly step for a probabilistic algorithm, is eliminated. Furthermore, a collision-free path is guaranteed without needing to consider the specific module shape. Although we focus on two body types (line segments and cubes), our methods can be readily applied to any *slender* body shape. There are some disadvantages as well: the need for slender adornment can be viewed as a constraint and the current formulation cannot handle external obstacles, unless used in conjunction with other planning methods.

We will mainly discuss *unfolding* motions. Folding motions are obtained by reversing unfolding motions. Thus, unfolding motions can be implemented online, whereas folding motions have to be computed prior to actuation.

4 Modeling Modular Robot Chains

4.1 Modules

A *module* is defined as a pair of links connected by a joint. The i^{th} module will have a joint i, "left" link i_L, and "right" link i_R (see Fig. 2). Observe that half modules (including a unary joint) are attached to the ends of an open chain and two modules are connected rigidly. Sometimes it may be necessary to fix some modules when their joint angle reaches π during unfolding. By abuse of terminology, the meta structure will be simply called a *linkage* where a *link* can have more than one module due to fixing. An example is shown in Fig. 2.

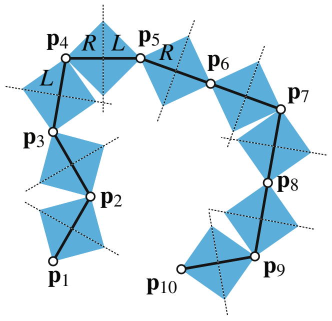


Fig. 2 A chain-type modular robot with cube bodies. Each module is delimited by dashed lines. The i^{th} module has two half bodies (i_L and i_R). If we assume that the robot is now unfolding itself, joints 6 and 8 are fixed so there are two long links $\mathbf{p}_5\mathbf{p}_7$ and $\mathbf{p}_7\mathbf{p}_9$.

Definition 1. *The* predecessor *of a module i,* PR*(i), is the closest unfixed module on the lefthand side of i. Similarly, its* successor, SU*(i), is defined as the closest unfixed module on its righthand side. The argument can contain a subscript, L or R. For example,* $\text{PR}(5) = 4$ *and* $\text{PR}(5_R) = 5$ *in Fig. 2.*

4.2 Representing Self-touching Shapes

We need to consider how to represent various shapes in a module configuration $\mathbf{p}$. If there is neither self-crossing between modules (which can be rigid bodies) nor self-touching between *bases* (Sec. 2.2), then the shape can be represented using $\mathbf{p}$. A more interesting application, however, may contain self-touchings between bases such as filling a region with a cube chain. Assume that the region to be filled is depicted as a polyomino since we have square pixels. We first construct a spanning tree of the polyomino (Fig. 3 left) by finding a spanning tree of the graph $G = (V,E)$ in which we treat each square as a vertex in V and each line segment shared by two

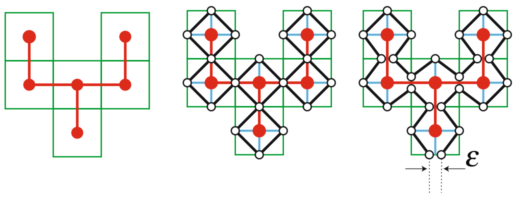


Fig. 3 A hexomino resembling a wrench head which is to be filled by an open chain of 24 cubes. An additional parameter ε was introduced to remove self-touchings.

adjacent squares as an edge in E. By dividing the constituent pixels (or squares) into 4 sub-pixels [9] and connecting diagonals of each smaller pixel we can always construct a piecewise-linear and simply-connected path which zigzags around the spanning tree (Fig. 3 center). This path can be thought of as a desired configuration for the *base linkage* of a cube chain. This, however, introduces a problem since the Carpenter's Rule Theorem does not allow self-touching between bases. We introduce a positive nonzero parameter ε to remove any self-touchings (Fig. 3 right).

Other approximation methods may be used to remove self-touching configurations. Such methods, however, imply that the faces of a module in a real prototype must incorporate a measure of compliance; they cannot be perfectly rigid.

5 Algorithms for Unfolding

5.1 *The Centralized Algorithm*

We shall directly apply the Carpenter's Rule theorem and centralized computation to obtain an unfolding motion. There exists one leader module which gathers position information from all modules, computes unfolding paths for them, and orders them to move. Theoretically, the leader module will solve $\dot{\mathbf{p}} = f(\mathbf{p})$, where the righthand side is the solution to Eqns. (1 - 4) for the set A of *active* modules after fixing any modules of joint angle π to make them *inactive*. Note that two half-modules at the ends of an open chain (and their unary joint) are assumed to be active at any time. Recall that $f(\mathbf{p})$ can be integrated to generate a smooth integral curve until any additional joint angle reaches π (type I cusp). The linkage is then simplified with one less link by fixing the module at π and a new integral curve will be attached to the previous one recursively. The centralized algorithm approximating the analytic integral curve follows in Table 1. An example can be found in [8].

SOLVEGLOBALPROGRAM() solves (1 - 4) to find an instantaneous motion. LINEAR2ANGULAR() converts linear velocity vectors $\mathbf{v}$ into angular velocities ω. For example, ω_5 can be calculated from $\mathbf{v}_4$, $\mathbf{v}_5$, and $\mathbf{v}_6$ in Fig. 1(a). The current configuration is then updated by applying ω for Δt in RECONFIGUREROBOT(). For practical purposes, we assume small, finite step size Δt, although the theoretical results hold when the step sizes are infinitesimal. Whenever the i^{th} joint angle reaches

Table 1 Centralized unfolding algorithm

```
function CENTRALIZED-UNFOLDING returns unfolding motion
input : p, an initial configuration to be unfolded

  while neither straight nor convex do
    v ← SOLVEGLOBALPROGRAM(p)
    $\omega$ ← LINEAR2ANGULAR(v)
    p ← RECONFIGUREROBOT($\omega$, $\Delta t$)
    if one joint = $\pi$ then
      p ← FIXMODULE(p)
    end if
  end while
```

π, the module i will be fixed by simply being eliminated from database and setting a new bar between $\text{PR}(i)$ and $\text{SU}(i)$ using FIXMODULE().

5.2 *The Decentralized Algorithm*

In many cases, it is beneficial to lessen the burden on the leader module. Indeed, a set of modules in a local neighborhood can be defined through local proximity sensing and used to formulate a decentralized version of the reconfiguration planning. We shall show that local proximity sensing can relax the convex program in (1 - 4) leading to a new problem with fewer constraints. The decentralized algorithm can then be used to compute desirable infinitesimal motions to be combined to construct a piecewise smooth unfolding path $\mathbf{p}(t)$. Cusps in $\mathbf{p}(t)$, however, occur not only whenever any joint angle reaches π (type I cusp) but also whenever the proximity relationship changes (type II cusp) as the linkage explores its configuration space.

5.2.1 Sensor Model

At each joint $\mathbf{p}_i$ we will attach a proximity sensor with two radii describing concentric circles centered at each sensor $\mathbf{p}_i$ with radius r_{SR} and $r_{SR}+\delta$, respectively. These two positive parameters will be compared to $d(\mathbf{p}_i, j_{(\cdot)})$, the minimal distance between a sensor at $\mathbf{p}_i$ and points on module $j_{(\cdot)}$, where the subscript can be L or R, in order for module i to identify its local neighborhood (Fig. 4(a)). For example, any $j_{(\cdot)}$ with $d(\mathbf{p}_i, j_{(\cdot)}) < r_{SR}$ will be declared as within i's neighborhood. Later we will show the distributed algorithm prevents collisions between module i and its local neighborhood (Sec. 5.2.2∼5.2.4).

Finally, let module i gather the following information based on hysteretic behavior due to the double sensing boundary.

Definition 2. *N_i is a set of half-modules comprising the neighborhood of i. Half-module $j_{(\cdot)}$ becomes a member of the neighborhood when $d(\mathbf{p}_i, j_{(\cdot)}) \leq r_{SR}$ and will remain a member of the neighborhood until $d(\mathbf{p}_i, j_{(\cdot)}) \geq r_{SR}+\delta$. As soon as $d(\mathbf{p}_i, j_{(\cdot)}) = r_{SR}+\delta$ for any $j_{(\cdot)} \in N_i$, $j_{(\cdot)}$ is no longer a member of the neighborhood.*

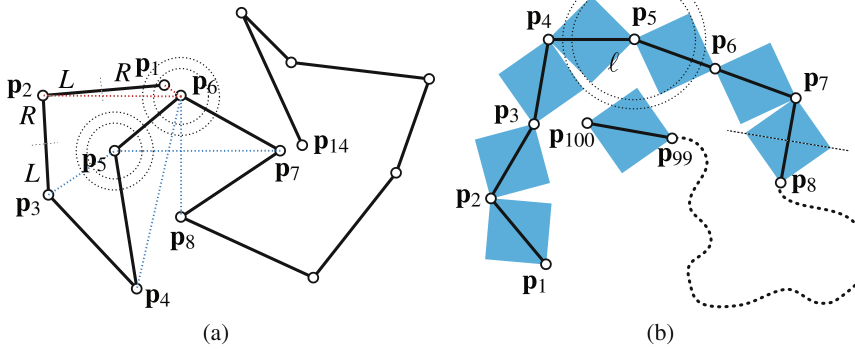


Fig. 4 (a) Sensor model for a line segment chain. Inner circles and outer circles have radius r_{SR} and $r_{SR} + \delta$, respectively. Red dashed lines starting from $\mathbf{p}_6$ represent $\{6,1\},\{6,2\} \in S(t)^6_{temp}$. Blue dashed lines starting from $\mathbf{p}_6$ are $\{6,4\},\{6,8\} \in S(t)^6_{fixed}$. Since module 5 has an empty local neighborhood, only $\{5,3\},\{5,7\} \in S(t)^5_{fixed}$ are defined. (b) Sensor model for a cube chain.

Intuitively, if i "sees" that half-module $j_{(\cdot)}$ is inside its inner sensing range, i will track $j_{(\cdot)}$ until it is totally out of sight beyond the outer sensing range.

5.2.2 Relaxing Constraints and Local Motion

Before formulating a decentralized algorithm, we shall consider here and in Sec. 5.2.3 how to guarantee the existence of an unfolding path under the local proximity sensing. It is convenient to assume yet again that a central processor is solving the existence problem until the decentralized version is discussed in Sec. 5.2.4. We shall begin with line segment chains where r_{SR} can be very small because two mid-link points can never collide unless preceded by a joint-joint or joint-link collision.

Proximity sensors will only be used on active modules because all collisions are involved with joints. Based on the sensing result at time t, we can define a set $S(t)^i_{temp}$ of struts as a collection of struts from i to predecessors or successors of its neighborhood where $i \in A$. The basic idea is to define a set of temporary struts to address potential collisions when separation distances are below r_{SR} (Fig. 4(a)).

$$S(t)^i_{temp} = \{\{i,j\} | j \in \{\textsc{pr}(k_{(\cdot)}), \textsc{su}(k_{(\cdot)}) | \forall k_{(\cdot)} \in N_i\}\} \tag{5}$$

As $\mathbf{p}_i$ gets farther from $\textsc{pr}(k_{(\cdot)})$ and $\textsc{su}(k_{(\cdot)})$, the distance between $\mathbf{p}_i$ and any point on the link connecting $\textsc{pr}(k_{(\cdot)})$ and $\textsc{su}(k_{(\cdot)})$ also increases [4]. This guarantees that there is no collision involved with joint i. In addition, a fixed set of struts $S(t)^i_{fixed}$ is defined (Fig. 4(a)):

$$S(t)^i_{fixed} = \{\{i, \textsc{pr}(\textsc{pr}(i))\}, \{i, \textsc{su}(\textsc{su}(i))\}\} \tag{6}$$

$$S(t)^i = S(t)^i_{temp} \cup S(t)^i_{fixed} \tag{7}$$

The struts in $S(t)^i_{fixed}$ unfold two active modules on either side of i. For example, $\{6,4\}$ in Fig. 4(a) is in charge of unfolding module 5. A set $B(t)^i$ of bars is:

$$B(t)^i = \{\{i, \mathrm{PR}(i)\}, \{i, \mathrm{SU}(i)\}\} \tag{8}$$

Note that if $B(t)^i$ and $S(t)^i$ have common elements, they will be removed from $S(t)^i$.

Now we can relax (1 - 4) using the fact that $S(t) = \bigcup_i S(t)^i \subset S_{original}$ and $B(t) = \bigcup_i B(t)^i = B_{original}$. Recall that $S_{original}$ is the set of all struts between any two active modules where one is not a direct predecessor or successor of the other and $B_{original}$ is the set of all bars between any two successive active modules:

$$\text{minimize} \sum_{i \in A} \|\mathbf{v}_i\|^2 + \sum_{\{i,j\} \in S(t)} \frac{1}{(\mathbf{v}_i - \mathbf{v}_j) \cdot (\mathbf{p}_i - \mathbf{p}_j) - \|\mathbf{p}_j - \mathbf{p}_i\|} \tag{9}$$

$$\text{subject to } (\mathbf{v}_j - \mathbf{v}_i) \cdot (\mathbf{p}_j - \mathbf{p}_i) > \|\mathbf{p}_j - \mathbf{p}_i\| \text{ , for } \{i,j\} \in S(t) \subset S_{original} \tag{10}$$

$$(\mathbf{v}_j - \mathbf{v}_i) \cdot (\mathbf{p}_j - \mathbf{p}_i) = 0 \text{ , for } \{i,j\} \in B(t) = B_{original} \tag{11}$$

$$\mathbf{v}_{a_1} = \mathbf{v}_{a_2} = 0, \quad a_1, a_2 \text{ are any two successive active modules.} \tag{12}$$

Since the convex program for $S_{original}$ is always feasible, this relaxed convex program also has a well-defined solution $\mathbf{v}$ at any time. $\mathbf{v}$ unfolds every joint in a greedy manner (due to $S(t)_{fixed}$) and avoids collisions (due to $S(t)_{temp}$) while maintaining the rigid constraints in $B(t)$.

Local Motion for a Cube Chain

For a cube chain, r_{SR} cannot be arbitrarily small because two mid-link points can collide in contrast to the line segment case. In other words, the size of a cube body determines minimum allowable sensing range. The value can be obtained using simple geometry in terms of ℓ, the side length (Fig. 4(b)). If $d(\mathbf{p}_i, j_{(\cdot)})$ refers to the minimal distance between sensor $\mathbf{p}_i$ and module $j_{(\cdot)}$, r_{SR} should satisfy:

$$r_{SR} > \ell \tag{13}$$

Recalling that a cube can be also treated as a line segment (base), we may want to make a sensor to detect points on the bases. Then $d(\mathbf{p}_i, j_{(\cdot)})$ now refers to the minimum distance between a joint and a base; a greater sensing range is needed.

$$r_{SR} > (1 + \sqrt{2}/2) \times \ell \tag{14}$$

As in the case of the line segment chain, only active modules will use their proximity sensors. Since r_{SR} is larger than a module, any collision will happen only after a sensor detects danger. It is, however, not sufficient to construct $S(t)^i_{temp}$ from module i which has found some neighborhood since the neighborhood can collide with other points on links adjacent to i. So, additional struts are needed to prevent these mid-link collisions. To avoid collision between two links of slender *symmetric* adornment like the cube chain, we should try to expand all distances between the end points of the links provided they are not connected by a bar [3]. This idea can be readily implemented by defining $S(t)^i_{trans}$:

$$S(t)^i_{trans} = \{\{i,j\} | j \in \{\mathrm{PR}(k_{(\cdot)}), \mathrm{SU}(k_{(\cdot)}) | \forall k_{(\cdot)} \in N_{\mathrm{PR}(i)} \cup N_{\mathrm{SU}(i)}\}\} \tag{15}$$

For example, sensor 5 in Fig. 4(b) will also construct struts $\{4,99\}, \{4,100\} \in S(t)^4_{trans}$ and $\{7,99\}, \{7,100\} \in S(t)^7_{trans}$ in addition to $\{5,99\}, \{5,100\} \in S(t)^5_{temp}$. These additional struts guarantee no collision between link $\mathbf{p}_4\mathbf{p}_5$ (or $\mathbf{p}_5\mathbf{p}_7$) and $\mathbf{p}_{99}\mathbf{p}_{100}$. The definitions for other sets are the same except for:

$$S(t)^i = S(t)^i_{temp} \cup S(t)^i_{fixed} \cup S(t)^i_{trans} \tag{16}$$

We can also obtain the relaxed problem (9 - 12) to get $\mathbf{v}$ at any time.

Hybrid System Model

We have shown that there exists an instantaneously safe unfolding motion $\mathbf{v}$ in spite of the relaxation. We shall call it *local-scale strictly expansive motion* in contrast to the global-scale strictly expansive motion. Since $S(t)$ changes over time, the vector field on the right hand side of $\dot{\mathbf{p}} = f(\mathbf{p})$ changes over time. Thus, we have a set of all possible vector field $\mathfrak{F} = \{f_1(\mathbf{p}), f_2(\mathbf{p}), \cdots\}$, forming a *hybrid system*, in contrast to the single vector field $f(\mathbf{p})$ of the original formulation. $\mathfrak{F}$ corresponds to a set of all possible combinations of $S(t)$ and $B(t)$ and the domain of $f_i(\mathbf{p})$, U_i, can be represented as an open subset in the configuration space P which satisfies $d(\mathbf{p}_a, b_{(\cdot)}) > r_{SR}$ for some $a, b_{(\cdot)}$ (they are not in a local neighborhood) and $d(\mathbf{p}_s, t_{(\cdot)}) < r_{SR} + \delta$ for some $s, t_{(\cdot)}$ (they are in a local neighborhood). The following theorem will be useful to construct the global solution.

Theorem 1. *Each vector field $f_i(\mathbf{p})$ in $\mathfrak{F} = \{f_1(\mathbf{p}), f_2(\mathbf{p}), \cdots\}$ is differentiable with respect to $\mathbf{p}$ in $U_i \bigcap Q$ where U_i is $f_i(\mathbf{p})$'s domain and Q is a collection of configurations around $\mathbf{p}$ in which any additional joint angle is not π.*

Proof. We only have to check that $f_i(\mathbf{p})$ satisfies five conditions from Lemma 7 in [4] which established the smooth dependence of the solution on the problem-definition data $A(\mathbf{p})$ and $b(\mathbf{p})$ in parametric optimization problems of the type:

$$\min\{g(\mathbf{p},\mathbf{v}) : \mathbf{v} \in \Omega(\mathbf{p}) \subseteq \mathbb{R}^m,\ A(\mathbf{p})\mathbf{v} = b(\mathbf{p})\} \tag{17}$$

Our relaxed problem can be regarded as this type where $g(\mathbf{p},\mathbf{v})$ refers to the objective function (9) and $\Omega(\mathbf{p})$ the feasible set from (10). $A(\mathbf{p})$ and $b(\mathbf{p})$ can be constructed from (11) and (12).

1. *Is the objective function twice continuously differentiable and strictly convex as a function of* $\mathbf{v} \in \Omega(\mathbf{p})$*, with a positive definite Hessian,* $\forall \mathbf{p} \in U_i \bigcap Q$ *?*
   The objective function is the sum of quadratic functions and additional smooth convex terms. Thus it is twice continuously differentiable and strictly convex.
2. *Is $\Omega(\mathbf{p})$ an open set,* $\forall \mathbf{p} \in U_i \bigcap Q$ *?*
   $\Omega(\mathbf{p})$ is open since the inequalities (10) are strict.

3. *Are the rows of the constraint matrix* $A(\mathbf{p})$ *linearly independent,* $\forall \mathbf{p} \in U_i \bigcap Q$ *?*
   Because the equality constraints are the same as the original formulation, this condition still holds [4].
4. *Are* $A(\mathbf{p})$*,* $b(\mathbf{p})$*, and* ∇g *(with respect to* $\mathbf{v}$*) continuously differentiable in* $\mathbf{p} \in U_i \bigcap Q$ *?*
   $A(\mathbf{p})$, $b(\mathbf{p})$ are linear. ∇g is also continuously differentiable from the fact in 1.
5. *Does the optimum point* $\mathbf{v}^*(\mathbf{p})$ *exist for every* $\mathbf{p} \in U_i \bigcap Q$ *?*
   The relaxed problem is also convex. We can always find a unique solution.

In conclusion, $f_i(\mathbf{p})$ satisfies the five conditions establishing the smooth dependence of $f_i(\mathbf{p})$ on $\mathbf{p}$ in $U_i \bigcap Q$. □

5.2.3 Global Motion with Hysteretic Behavior

We now show how to construct an integral curve (unfolding path) on the configuration space governed by the hybrid system and hysteretic behavior in sensing.

Theorem 2. *Consider the hybrid dynamic system* $\dot{\mathbf{p}} = \mathbf{v} = f_{(\cdot)}(\mathbf{p})$ *where* $f_{(\cdot)}(\mathbf{p}) \in \mathfrak{F}$ *and assume the hysteretic behavior from Definition 2. There exists a unique integral curve which represents the unfolding path from given initial configuration* $\mathbf{p}_c$*.*

Proof. Given a configuration $\mathbf{p}_c$, we can designate a unique vector field until one of the neighborhood memberships expires or a new membership is issued. In fact, $\mathbf{p}_c$ is located strictly inside the domain of the designated vector field since we have a nonzero margin before a new membership is issued or an existing membership expires due to the hysteresis.

Recalling from Theorem 1 that each $f_{(\cdot)}(\mathbf{p})$ is differentiable, we can define a unique maximal integral curve in the domain which cannot be extended beyond a certain positive limit on time $T \leq \infty$ starting from $\mathbf{p}_c$. But the integral curve should reach the boundary of the current vector field (type II cusp) in finite time because of the finite growth rate (at least unit rate) of the strut constraints in $S(t)_{temp}$.

As soon as it reaches the boundary, a switching of vector fields occurs and a unique integral curve will be constructed again exploiting the fact that the switching point is also located strictly inside new vector field's domain due to the hysteresis. This new integral curve will be connected to the existing integral curve, but these processes will last only finitely until we get to a type I cusp since the struts in $S(t)_{fixed}$ are also growing at least with unit rate at any instant. Type I cusps can also appear only finitely. Therefore we can finish the unfolding in finite time. □

Hysteresis plays an important role in the above theorem. It guarantees that the points where vector fields switch are actually located strictly inside new vector field's domain. Thus it allows local motion to be always computed by a well-defined vector field exclusively from $\mathfrak{F}$. What would have happened without the hysteresis?

Each vector field in $\mathfrak{F}$ has repulsive nature in that any integral curve starting from strict inside of its domain tends to escape the domain by expanding struts which will change current neighborhood relationship. If there is no hysteretic behavior, in other words, if $\delta \rightarrow 0$, this repulsive nature of vector field is very likely to result in *sliding* *mode* [6, 14] on domain boundaries. To describe sliding mode, we need a third vector field which is different from the two vector fields on either side of the boundary. This new vector field, however, may not be desired since it is not an element of $\mathfrak{F}$ in which every element was proved to generate safe local motions. To be more specific, the new vector field may not guarantee local expansiveness for collision avoidance or more than unit rate of strut expansion for convergence and completeness. Thus the hysteresis is required for the solution to avoid any undesirable sliding mode.

5.2.4 Decentralized Algorithm

Recall that $A = \{a_1, a_2, \cdots a_m\}$ is a set of active modules. Without loss of generality, we can rewrite it as $A = \{1, 2, \cdots, m\}$ since we have been working only with active modules. We then have m subsystems where each module k is coupled only with modules in I_k, a set of all modules which appear in $S(t)^k \cup B(t)^k \cup D(t)^k$, where $D(t)^k$ is a set of struts which ends at k, for example, $\{6, 1\} \in D(t)^1$ in Fig. 4(a). In other words, considering $D(t)^k$ means that we will take the modules which detect k into account. Also note that $k \in I_k$. No matter how many active modules there are, we only have to maintain a limited number of local contacts which can be found using $\text{PR}(\cdot)$ and $\text{SU}(\cdot)$ pointers. The pointers do not require global information so each module doesn't need to have a specific ID. The number of required local contacts for each module has an constant upper bound particularly for a cube chain since the body shape and the sensing area are compact on the plane. Compared to the centralized algorithm in which every active module is coupled with all others, this fact allows us to formulate a decentralized algorithm by decomposing Eqns. (9 - 12). Assume that each module is equipped with a perfect localizer as before. First we shall designate a reference module to which every localizer should refer or pin down a module to address (12).

Now we need some concepts and notations from [1]. A hypergraph can be used to represent the decomposition structure where the nodes are active modules and the *nets* (or hyperedges) are constraints among them. For example, the fact that $\mathbf{v}_6$ should be shared by six others (modules 1, 2, 4, 5, 7, and 8) in Fig. 4(a) corresponds to a net (hyperedge) in the decomposition structure. Let $\mathbf{u}_k \in \mathbb{R}^{X_k}$ be a collection of velocity of modules which belong to I_k. If $I_k = \{m, n, k, p, q\}$, then an example would be, $\mathbf{u}_k = (\mathbf{v}_m^T \quad \mathbf{v}_n^T \quad \mathbf{v}_k^T \quad \mathbf{v}_p^T \quad \mathbf{v}_q^T)^T \in \mathbb{R}^{X_k}$. Let $\mathbf{u} = (\mathbf{u}_1^T \quad \mathbf{u}_2^T \quad \cdots \quad \mathbf{u}_m^T)^T \in \mathbb{R}^X$, $X = X_1 + \cdots + X_m$. We will use the notation $(u)_i$ to denote the i^{th} scalar component of $\mathbf{u}$. The basic strategy is to independently solve for each module k which has a cost function $h_k(\mathbf{u}_k)$, and constraints $\mathbf{u}_k \in \mathfrak{C}_k$, which is a subset of constraints in (10) and (11) featuring $\mathbf{v}_k$, and impose the nets to establish consistency among shared variables. In terms of $\mathbf{u}$, various components of $\mathbf{u}$ should be the same as defined in a net. This idea can be efficiently represented by introducing a vector $\mathbf{z} \in \mathbb{R}^Y$, Y is the number of nets, which gives the common values to each net by calculating $E\mathbf{z}$ where $E \in \mathbb{R}^{X \times Y}$:

$$E_{ij} = \begin{cases} 1 & (u)_i \text{ is in net } j \\ 0 & \text{otherwise} \end{cases} \tag{18}$$

Table 2 Decentralized unfolding algorithm for an active module k

```
function DECENTRALIZED-UNFOLDING returns instantaneous unfolding motion $\mathbf{u}_k$
input : $\mathbf{p}_{I_k}$, position of modules in $I_k$

  $\mu_k \leftarrow 0$
  while $\Delta\mu_k > \varepsilon$ (small positive constant) do
    $\mathbf{u}_k \leftarrow$ SOLVELOCALPROGRAM($\mathbf{p}_{I_k}, \mu_k$)
    $\mu_k \leftarrow$ UPDATEPRICES($\mu_k$)
  end while
```

Lastly, let $E_k \in \mathbb{R}^{X_k \times Y}$ denote the partitioning of the rows of E into blocks associated with module k such that $\mathbf{u}_k = E_k\mathbf{z}$.

Our problem is then to solve a master problem:

$$\text{minimize} \sum_{k=1}^{m} h_k(\mathbf{u}_k) \tag{19}$$

$$\text{subject to } \mathbf{u}_k \in \mathfrak{C}_k, \quad k = 1, \cdots, m \tag{20}$$

$$\mathbf{u}_k = E_k\mathbf{z}, \quad k = 1, \cdots, m \tag{21}$$

We will let

$$h_k(\mathbf{u}_k) = \sum_{i \in I_k} \|\mathbf{v}_i\|^2 + \sum_{\{i,j\} \in S(t)^k \cup D(t)^k} \frac{1}{(\mathbf{v}_i - \mathbf{v}_j) \cdot (\mathbf{p}_i - \mathbf{p}_j) - \|\mathbf{p}_j - \mathbf{p}_i\|} \tag{22}$$

Note that we have the same set of constraints as the previous formulation since the union of $\mathfrak{C}_k$'s is equal to (10) and (11) and we already have addressed (12). The above formulation is, however, different from the previous formulation since the cost function $\sum_{k=1}^{m} h_k(\mathbf{u}_k)$ is different from (9). Still, there is no problem to apply the results in Sec. 5.2.2 and 5.2.3 since $\sum_{k=1}^{m} h_k(\mathbf{u}_k)$ is just another weighted sum of terms in (9) which do not affect convexity and differentiability.

We will apply *dual decomposition* featuring a *projected subgradient method* [1] to solve this master problem. Then each active module only solves the following local problem:

$$\text{minimize } h_k(\mathbf{u}_k) + \mu_k^T \mathbf{u}_k \quad \text{subject to } \mathbf{u}_k \in \mathfrak{C}_k \tag{23}$$

To be more specific, this can be elaborated as:

$$\text{minimize} \sum_{i \in I_k} \|\mathbf{v}_i\|^2 + \sum_{\{i,j\} \in S(t)^k \cup D(t)^k} \frac{1}{(\mathbf{v}_i - \mathbf{v}_j) \cdot (\mathbf{p}_i - \mathbf{p}_j) - \|\mathbf{p}_j - \mathbf{p}_i\|} + \mu_k^T \mathbf{u}_k \tag{24}$$

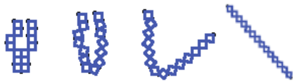
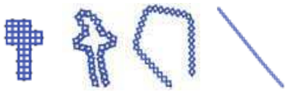
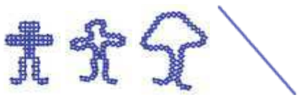
$$\text{subject to } (\mathbf{v}_j - \mathbf{v}_i) \cdot (\mathbf{p}_j - \mathbf{p}_i) > \|\mathbf{p}_j - \mathbf{p}_i\| \text{, for } \{i,j\} \in S(t)^k \cup D(t)^k \tag{25}$$

$$(\mathbf{v}_j - \mathbf{v}_i) \cdot (\mathbf{p}_j - \mathbf{p}_i) = 0 \text{, for } \{i,j\} \in B(t)^k \tag{26}$$

where μ_k is the Lagrange multiplier for $\mathbf{u}_k$.

Each active module runs the following decentralized algorithm independently.

Table 3 Computation time ratio. Implementation was done in MATLAB and used cvx; the results have been normalized by the time needed to compute the Humanoid reconfiguration using the centralized algorithm. Note that snapshots are rescaled for visibility.

Shapes	Reconfiguration	Relaxed solution computation time / Centralized solution computation time
Gripper (16 modules)		0.03/0.05
Hammer (32 modules)		0.18/0.55
Humanoid (48 modules)		0.19/1

SOLVELOCALPROGRAM() solves (24 - 26). To update the Lagrange multiplier μ_k in UPDATEPRICES(), a module k has to communicate only locally with others in I_k. Refer to [1] for details about updating prices. Recall again that each module finds linear velocity as a result of this computation. Thus each module may need to be equipped with a synchronized clock to stop above decentralized work, transform linear velocity into angular velocity, and actuate its joint simultaneously. As mentioned earlier, we need to reverse unfolding motions to get folding motions.

We have compared the centralized algorithm (1 - 4) to the relaxed problem (9 - 12); the results are shown in Table 3. The results have been scaled to show the improvement in computation time between the formulations and allude to the promising results expected from the end-to-end decentralization algorithm of Table 2. Note that the relaxed problem takes much less time even though there is an additional step to find neighborhood, holding other conditions constant. Thus the fully decentralized implementation, deferred for future work, is expected to be much faster with less communication. The results have been scaled by the time needed to compute the Humanoid reconfiguration using the centralized algorithm. Note that this example was not optimized for runtime efficiency, but rather serves to show how the algorithm scales with the number of modules as well as the difference between centralized and relaxed versions.

6 Conclusion

This work presented practical algorithms for collision-free reconfiguration planning for chain-type modular robots. Both the centralized and decentralized versions were developed using the Carpenter's Rule Theorem and maintained connections between modules. The decentralized algorithm will be particularly beneficial as the number of modules increases since each module (without a unique ID) only has to maintain a limited number of contacts.

We are interested in testing our algorithms by implementing the sensor model in a current modular robot system. The pointers, PR$(\cdot)$ and SU$(\cdot)$, will be easily implemented using a neighbor-to-neighbor communication scheme. As mentioned before, the capability of our algorithms to deal with close-packed configuration can be integrated with other popular motion planning methods to handle obstacles and improve overall performance.

Acknowledgements. This work is funded in part by DARPA Grant W911NF-08-1-0228 (Programmable Matter).

References

1. Boyd, S., Xiao, L., Mutapcic, A., Mattingley, J.: Notes on decomposition methods (2007)
2. Casal, A.: Reconfiguration planning for modular self-reconfigurable robots. Ph.D. thesis, Stanford University (2002)
3. Connelly, R.: Expansive motions (2006)
4. Connelly, R., Demaine, E., Rote, G.: Straightening polygonal arcs and convexifying polygonal cycles. In: Annual IEEE Symposium on Foundations of Computer Science, vol. 0, p. 432 (2000), `http://doi.ieeecomputersociety.org/10.1109/SFCS.2000.892131`
5. Connelly, R., Demaine, E.D., Demaine, M.L., Fekete, S.P., Langerman, S., Mitchell, J.S.B., Ribó, A., Rote, G.: Locked and unlocked chains of planar shapes. In: SCG 2006: Proceedings of the twenty-second annual symposium on Computational geometry, pp. 61–70. ACM, New York (2006), `http://doi.acm.org/10.1145/1137856.1137868`
6. Filippov, A.F.: Differential Equations with Discontinuous Righthand Sides. Kluwer Academic Publishers, Dordrecht (1988)
7. Goldstein, S.C., Campbell, J.D., Mowry, T.C.: Programmable matter. Computer 38(6), 99 (2005), `http://dx.doi.org/10.1109/MC.2005.198`
8. Gray, S., Seo, J., White, P., Zeichner, N., Yim, M., Kumar, V.: A toolchain for the design and simulation of foldable programmable matter. In: Proceedings of the ASME International Design Technical Conferences and Computer and Information in Engineering Conference (2010)
9. Griffith, S.: Growing machines. Ph.D. thesis, Massachusetts Institute of Technology (2004)
10. Hawkes, E., An, B., Benbernou, N.M., Tanaka, H., Kim, S., Demaine, E.D., Rus, D., Wood, R.J.: Programmable matter by folding. In: Proceedings of the National Academy of Sciences (2010), doi: 10.1073/pnas.0914069107
11. Latombe, J.C.: Robot Motion Planning. Kluwer Academic Publishers, Norwell (1991)
12. Nguyen, A., Guibas, L., Yim, M.: Controlled module density helps reconfiguration planning. In: Algorithmic and Computational Robotics: New Directions: The Fourth Workshop on the Algorithmic Foundations (WAFR). AK Peters, Ltd. (2001)
13. Rus, D., Vona, M.: Self-reconfiguration planning with compressible unit modules. In: Proceedings of IEEE/RSJ IEEE International Conference on Robotics and Automation, Detroit, vol. 4, pp. 2513–2520 (1999)

14. Sastry, S.: Nonlinear Systems: Analysis, Stability, and Control. Springer, Heidelberg (1999)
15. Shen, W., Will, P.: Docking in self-reconfigurable robots. In: Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, vol. 2, pp. 1049–1054 (2001), doi:10.1109/IROS.2001.976307
16. Tang, X., Thomas, S., Amato, N.: Planning with reachable distances: Fast enforcement of closure constraints. In: IEEE International Conference on Robotics and Automation, pp. 2694–2699 (2007), doi:10.1109/ROBOT.2007.363872
17. Walter, J., Welch, J., Amato, N.: Distributed reconfiguration of metamorphic robot chains. Distributed Computing 17(2), 171–189 (2004)
18. White, P.J., Kopanski, K., Lipson, H.: Stochastic self-reconfigurable cellular robotics. In: Proceedings of IEEE/RSJ International Conference on Robotics and Automation, New Orleans, LA, USA, vol. 3, pp. 2888–2893 (2004)
19. White, P.J., Posner, M.L., Yim, M.: Strength analysis of miniature folded right angle tetrahedron chain programmable matter. In: Proceedings of IEEE/RSJ International Conference on Robotics and Automation (2010)
20. Yakey, J., LaValle, S.M., Kavraki, L.E.: Randomized path planning for linkages with closed kinematics chains. IEEE Transactions on Robotics and Automation 17(6), 951–959 (2001)
21. Yim, M., Duff, D.G., Roufas, K.D.: Polybot: a modular reconfigurable robot. In: Proceedings of IEEE/RSJ IEEE International Conference on Robotics and Automation, vol. 1, p. 514 (2000), `http://dx.doi.org/10.1109/ROBOT.2000.844106`
22. Yim, M., White, P.J., Park, M., Sastra, J.: Modular self-reconfigurable robots. In: Meyers, R.A. (ed.) Encyclopedia of Complexity and Systems Science, pp. 5618–5631. Springer, Heidelberg (2009)
23. Yim, M., Zhang, Y., Lamping, J., Mao, E.: Distributed control for 3d metamorphosis. Autonomous Robots 10(1), 41 (2001), `http://dx.doi.org/10.1023/A:1026544419097`
24. Zakin, M.: ProgrammableMatter-The Next Revolution in Materials. Military Technology 32(5), 98 (2008)

Robomotion: Scalable, Physically Stable Locomotion for Self-reconfigurable Robots

Sam Slee and John Reif

Abstract. Self-reconfigurable robots have an intriguingly flexible design, composing a single robot with many small modules that can autonomously move to transform the robot's shape and structure. Scaling to a large number of modules is necessary to achieve great flexibility, so each module may only have limited processing and memory resources. This paper introduces a novel distributed locomotion algorithm for lattice-style self-reconfigurable robots which uses constant memory per module with constant computation and communication for each attempted module movement. Our algorithm also guarantees physical stability in the presence of gravity. By utilizing some robot modules to create a static support structure, other modules are able to move freely through the interior of this structure with minimal path planning and without fear of causing instabilities or losing connectivity. This approach also permits the robot's locomotion speed to remain nearly constant even as the number of modules in the robot grows very large. Additionally, we have developed methods to overcome dropped messages between modules or delays in module computation or movement. Empirical results from our simulation are also presented to demonstrate the scalability and locomotion speed advantages of this approach.

1 Introduction

Throughout nature, a recurring concept is that of a collection of simple structures combining to form something much more complex and versatile. Self-reconfigurable (SR) robots seek to implement this concept using a collection of robotic "*modules*"

Sam Slee
Department of Computer Science, Duke University
e-mail: sgs@cs.duke.edu

John Reif
Department of Computer Science, Duke University
e-mail: reif@cs.duke.edu

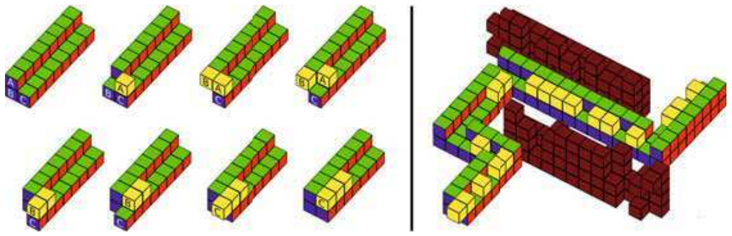


Fig. 1 **Left:** Three modules at the back of a Robomotion tunnel coordinate to move into that tunnel. **Right:** A Robomotion tunnel snakes around two brown obstacles. Yellow modules are moving.

which can autonomously move to reconfigure the overall shape and structure of the robot. A key problem faced by these robots is how to generate locomotion. Many prior algorithms have done this, but very few have been able to scale to robots with very large numbers of modules while only using robotic structures – configurations of modules – which are stable in the presence of gravity. We introduce such a method here with our Robomotion algorithm, which focuses on 3 main goals: scalability, speed, and stability.

For *Goal 1*, scalability, we mean that the algorithm could reasonably be expected to execute on a SR robot, implemented in hardware, composed of thousands or millions of modules. This means that the memory per module must be sub-linear in the number of modules and decisions made by each module can only be based on local information (i.e. from nearby neighbor modules). As we describe in Section 3, our algorithm achieves per module constant bounds on memory, processing, and communication. This has allowed us to simulate robots with over 2 million modules.

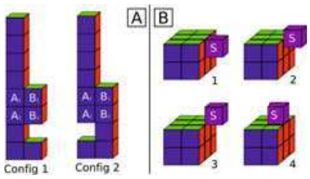


Fig. 2 **A:** A subtle change making a big stability difference. **B:** A convex-corner transition from side to top by Module *S*.

For *Goal 2*, speed, we are actually interested in the locomotion speed of the robot. There are two parts to creating fast locomotion: the physical movement approach and the means of controlling it. For large collections of modules, a small group of them will not be strong enough to move the rest of the robot. Prior work avoided this limitation by having some modules remain stationary while other modules flowed from the robot's back to its front. We have also adapted this approach for our Robomotion algorithm. For equally scalable control, a very distributed algorithm is required. Our algorithm achieves this by using straight tunnels through the robot's interior through which modules can travel. These tunnels act much like highways: fewer movement options allow for increased speed and efficiency.

For *Goal 3*, stability, we want to guarantee that our robot will not collapse under its own weight. This is a difficult global property to maintain because SR robots are able to form so many different shapes, many of which are unstable in the presence of gravity. Figure 2 shows how a slight change to 1 module's position can dramatically affect stability. In Config 1, the inter-module connections A_1-B_1 and A_2-B_2 experience very small tension and shear forces due to the weights of modules B_1 and B_2, respectively. However, by moving 1 module, in Config 2 those same inter-module connections must support all modules stacked above A_1, a far greater amount of shear and tension force bounded only by the height of the stack. Compression forces in other parts of the robot are about the same for either situation but are less important. Similar to large manmade structures, most SR robot hardware implementations handle compression forces much better than shear or tension forces on inter-module connections.

Although difficult to maintain, stability is a critical property. A real robot which loses efficiency will just move slowly. A real robot which loses stability will break hardware units. This cannot be a property that is probabilistically met. It must be guaranteed. For our Robomotion algorithm presented here, we guarantee stability while assuming only a very limited set of physical abilities for modules.

In addition to our 3 main goals, we make our algorithm more realistic by handling potential hardware errors: message drops and module delays in computation or movement. We also disallow convex-corner transitions by individual modules (move around a convex corner formed by 2 orthogonal surfaces as in Figure 2, Part B) as many hardware implementations have had difficulty executing these movements.

To accomplish our goals, our algorithm has a distributed approach through coordination within small groups of modules. These groups are dynamically formed whenever they are needed, and disassembled when no longer needed. In a sense, modules act like biological stem cells, able to join any group and take on whatever role is required. Dynamically forming a group requires consensus that each module has agreed to join the group and precisely which role each will take. Proving that a consensus will always be reached becomes difficult when the needs of the robot could change at any time (i.e. the robot could be instructed to reverse its locomotion direction or to turn), when different modules hear about those changes at different times (since we rely on module-to-module communication), and when messages sent between modules are not guaranteed to always be received.

The remainder of our paper is organized as follows. In Section 2 we survey the relevant literature. In Section 3 we give a general overview for how Robomotion addresses our goals described above. Section 4 describes our simulation of Robomotion and gives results from several experimental trials we used to compare our algorithm to the leading prior work on this topic. In Section 5 we present our conclusions and describe future work. Detailed pseudo-code and proofs are omitted here but are included at http://www.cs.duke.edu/∼reif/paper/slee/robomotion/robomotion.pdf. The primary contributions of this paper are Robomotion's algorithmic design and the experimental results from its simulation.

2 Related Work

In order to achieve highly scalable locomotion – for robots with thousands or millions of modules – we must have distributed control algorithms that use sub-linear

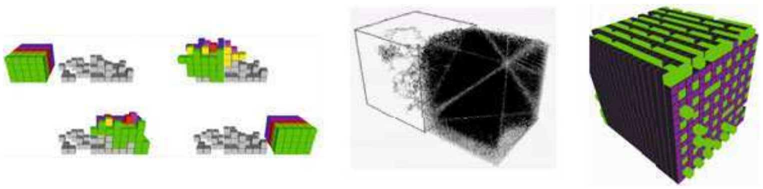


Fig. 3 **Left:** A Waterflow style algorithm by Butler et. al. [4] **Middle:** The Million Module March algorithm. [5] **Right:** Robomotion in simulation.

memory and processing per module. The best previous example is the Million Module March (MMM) algorithm by Fitch and Butler [5]. The robot's goal is to reach a specified goal region and each module executes the same, limited plan: repeatedly calculate its best route to a desired goal region based on the routes chosen by neighboring modules. In this way modules closer to the goal region pass information back to modules further away. This approach is flexible and was shown to work on a robot with just over 2 million modules. However, it did not consider physical stability and so would create configurations which would be unstable in the presence of gravity. Also, the constant replanning of routes in MMM can lead to inefficient locomotion.

Aside from MMM, the other main approach to highly scalable locomotion for lattice-style SR robots is not a specific algorithm but a general technique known as *waterflow* or cluster flow. In this algorithm, modules in the middle of the robot stay in place while modules at the back move up, slide across the top, and then come to the front of the robot to reattach there. Continual repetitions of this generate locomotion that looks like water flowing along the ground. Algorithms implementing this technique have been made by several research groups [1, 4, 7].

As other researchers have noted[1], the key to generating fast locomotion with these techniques is to have a high ratio of moving modules to total modules in the robot. In this paper we refer to this ratio as the Simultaneous Active Movement (SAM) rate which we describe further in Section 3. Both the waterflow technique and MMM restrict module movement to the exterior surface of the robot. For dense robot shapes, which are more likely to be stable, we would expect the SAM rate to drop as the number of modules grows and more modules become trapped in the robot's interior. We empirically verify this with our simulation results given in Section 4, and also show that Robomotion instead maintains a high SAM rate by allowing interior module movement. Of course, the SAM rate of an algorithm will drop even faster if we use a slow control algorithm. One highly scalable implementation of waterflow by Butler et. al. used stateless local rules to reduce computation [4]. However, later analysis found this approach to be unwieldy as standard program-based control had tens of rules while stateless local controllers had hundreds of rules. [2]

General reconfiguration algorithms could also be used for locomotion by continually requesting new configurations in the desired direction of movement. However, most are not highly scalable as they require linear memory for at least one module.

[1] From page 5 of [1]: "the speed is proportional to the ratio of moving meta-modules relative to the total number of meta-modules."

Some are also too slow due to requiring linear time for planning or, worst of all, using a centralized planner. One reconfiguration algorithm that did use sub-linear memory is the scale-independent algorithm of Nagpal and Stoy [14]. This specified the goal region as overlapping rectangular boxes and then placed modules into the goal region relative to the scale of the boxes. This algorithm forms static scaffolding in the goal region first so other "wandering" modules can flow freely through the gaps that are formed. However, Nagpal and Stoy found that their scheme had worse performance compared to traditional reconfiguration algorithms in terms of number of moves, time steps, and messages.

A similar reconfiguration algorithm by Stoy was not scale independent but still used that scaffolding technique [13]. Here static modules in the scaffolding would send out signals to attract wandering modules to new locations. This algorithm could be adapted to locomotion but would have some movement inefficiencies since multiple wandering modules can be attracted to a single open location. Also, no guarantees about physical stability are made and the algorithm's guarantee for connectivity (a property described in Section 3.2) only holds if modules in the scaffolding never move once they join that scaffolding. For locomotion, portions of the scaffolding would continually need to be removed and moved elsewhere. Doing this safely becomes difficult when the robot could change its movement direction at any time and module actions are asynchronous. Another approach given by Ravichandran, Gordon, and Goldstein [10] used only $O(\log n)$ memory per module while finding a bijection between initial positions of modules and the desired target positions. However, that work did not focus on the motion plan to reconfigure those modules.

Some reconfiguration algorithms allow "tunneling" module movement through the interior of the robot's structure [15, 3], a key property of our own Robomotion algorithm. However, since there is no central coordination between the planned paths for modules, the SAM rate for these prior tunneling algorithms is likely to be similar to that of MMM, which also uses a decentralized, greedy approach. The SAM rate for the MMM algorithm is considered in Section 4 of this paper. Also, these tunneling algorithms do not guarantee physical stability.

The Robomotion algorithm described in this paper is for lattice-style SR robots, but fast locomotion has been demonstrated in hardware for small chain-style SR robots. In this style the robots form kinematic chains or loops. Work by Yim et. al. may have the fastest locomotion demonstration [11], but other work by Yim [17], Shen et. al. [12], and Murata et. al. [9, 16] show various locomotion gaits with walkers, snake-like sidewinders, or rolling loops. However, if gravity is considered, these techniques would not scale to moving very large robots (with thousands or millions of modules) and are not directly applicable to lattice-style robots.

Physical stability occurs in hardware demonstrations for SR robots, but the topic of stability has not been heavily studied for highly scalable theoretical algorithms. Prior work by Shen et. al. has shown how to balance a chain-type SR robot by calculating the center of mass of the robot in a distributed manner [8]. However, this did not consider physical stability in the sense of guaranteeing limits on shear and tension forces experienced between adjacent modules. These stability guarantees are a key contribution of the Robomotion algorithm presented in this paper, along with its high scalability and movement efficiency benefits.

3 Overview of Intuition for Robomotion

To make our theoretical algorithm applicable to as many lattice-style robot hardware types as possible, Robomotion assumes the commonly-used Sliding Cube abstract model [6]. In this model each module is represented as a cube and modules are able to slide along flat surfaces formed by other cubes and make convex and concave transitions between those surfaces. Communication occurs with messages passed between adjacent modules. It has been shown that algorithms made for this model can be executed by a range of hardware implementations [6]. To further extend the applicability of our Robomotion algorithm, we disallow convex transitions as this action is hard for many hardware implementations. We now describe our Robomotion algorithm and the intuition behind it. Again, our main goals for this work were: fast locomotion speed, physical stability, and high scalability.

3.1 Fast Locomotion

To help evaluate the speed of our locomotion algorithm, we define the *Simultaneous Active Movement (SAM) rate* as the ratio of the number of moving modules compared to the total number of modules in the robot. Prior work has noted how locomotion speed for lattice-style SR robots correlates closely with this metric [1] and that this rate tends to correlate with the exterior surface area of the robot[2] [1, 5]. Asymptotically, an ideal SAM rate would be 1:1. This occurs when, on average, a constant fraction of modules in the robot are able to simultaneously move. To get a high SAM rate we'll need to have the surface area of the robot – which is the area along which modules can travel – to be asymptotically equal to the volume of the robot. A robot only two modules high could have the top surface of modules move for a 1:1 SAM rate, but in general as the number of modules grows large keeping one dimension asymptotically shorter than another is difficult to achieve or maintain.

Instead, our Robomotion algorithm uses interior movement to reach a 1:1 SAM rate. Modules begin at the back of the robot and move to new positions at the front, just like in the Waterflow approach described in our Related Work Section. However, now modules move through interior tunnels instead of across the exterior surface of the robot. By using a constant fraction of the robot's volume to move modules, we can achieve a 1:1 SAM rate. The tradeoff is that interior movement makes physical stability a harder property to maintain.

[2] From page 9 of [5]: "We note that for simple cubic shapes, the surface area of the robot increases with $n^{2/3}$, so in fact we should expect parallelism relative to n to decrease as the robot gets bigger."

3.2 Stable Locomotion

Most SR robot algorithms do not consider stability, but most do maintain another global property: *connectivity*. For this property, each module in the robot has a path (series of physical connections) to any other module in the robot. For stability, it helps to consider the common prior approach to connectivity: safety searches.

Prior Solution: Safety Searches. With this method, before any module *m* moves it first finds alternate paths to connect its neighbors. Module *m* then locks into place any modules on these paths before *m* itself can move. This technique was used by the highly scalable MMM algorithm [5]. However, there are three main drawbacks to this approach. (1) If we allow unbounded searches, a single search could be very slow. In the worst case we would have a single loop of all *n* modules in a robot and would have to go through all of them to make an alternate path, finally allowing 1 module to move. (2) If we bound searches, then we might miss the opportunity to move modules that could have safely moved, thereby lowering the robot's SAM rate. Finally, (3) without any global coordination, the movement of modules can become very random and the structure formed for the robot becomes hard to predict.

For modules moving through the robots interior, these paths are really tunnels. Without global coordination, random module movements can quickly create complex internal mazes of tunnels and verifying stability becomes much more complex than the simple "find any path" searches needed for connectivity. Just as we would not trust the stability of gold mining tunnels that were randomly dug above and below each other, for an SR robot we need to be more coordinated in how we form these interior movement tunnels.

Our New Solution: Support Columns. Our Robomotion algorithm avoids these tricky situations by using static *support columns* placed at repeated intervals through the robotic structure. Modules in these support columns do not move and so can provide support. Modules between these columns attach to them for support and are free to move without fear of causing instabilities. To form these support columns, we use repeated groupings of 3 modules which we refer to as *L-groups*. An example L-group is pictured at the far left of Figure 4 while the other portions of that figure show how L-groups can be stacked into a column or into a set of adjacent columns to form a full robot structure. The gaps in pictures (b) and (c) of Figure 4 – which account for 1/4 of the available volume in those structures – show where other modules could travel freely through the structure. If we can keep new modules constantly moving through these gaps then we'll have our desired 1:1 SAM rate.

Define a *slice* of modules as a set of modules all having the same coordinate position for a given axis direction (assuming those directional axes are aligned with the rows and columns of our robot's lattice structure). Robomotion generates locomotion by repeatedly disassembling the columns of modules in the backmost slice of the robot, sending those modules through the interior tunnels, and finally assembling a new slice of modules (forming new support columns) at the front of the robot. We can actually disassemble or assemble an entire slice in parallel and still maintain

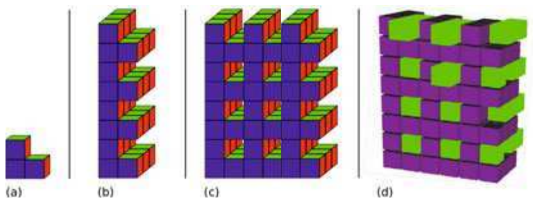


Fig. 4 From left: (a) a single L-group; (b) 1 line of columns; (c) multiple adjacent column lines to show what a full robot looks like; (d) our simulation implementation with L-groups in purple and moving modules in green.

stability. Keeping columns close together bounds the number of modules hanging from the side of any given column module. Assembling at most 1 slice at the front, and 1 slice at the back, of the robot at a given time limits the number of modules hanging from the front and back of any column module. If each module has roughly the same weight, this also means that we've bounded the amount of shear or tension force experienced on the connection between any pair of modules.

3.3 Scalable Locomotion

This support column structure also permits a highly scalable locomotion algorithm. The only safety checks we need come while assembling or disassembling a column (as part of a slice). These checks are: (1) consensus between modules in an L-group to decide to disassemble or to decide if assembly of that L-group is complete, and (2) messages between L-groups (in the back slice) from the top down to state when a column is ready to be disassembled, and messages from the bottom up (front slice) to state when a new column has finished being assembled, and (3) 1-directional horizontal messages (perpendicular to movement direction) between L-groups to state when a slice is ready to be disassembled or has finished being assembled.

Only constant-bounded memory and communication is needed. Disassembly or assembly communication really only involves 5 modules: the 3 modules in an L-group, a module from an adjacent L-group above (for disassembly) or below (for assembly) stating that the column is ready, and a module from a side-adjacent L-group stating that a slice is ready. Thus, a module only needs to know state about its own L-group (if it's in one) and to receive messages from neighbors. Thus, each module needs only a constant-bounded amount of memory if we put a limit on the number of messages "yet to be read" for any module. Overflow messages could be dropped as that challenge is also handled by Robomotion (as described later in Subsection 3.6). Modules moving through the robots interior can just follow directions given by adjacent L-group modules. Therefore moving modules only need to remember their state (i.e. that they are moving) and the most recent move direction order received.

Only constant-bounded processing is needed as well. Inside of the robot, only 1 module per L-group does anything: it sends direction messages to moving modules

```
Send MOVE-DIRECT messages to any adjacent FREE modules.
if IS-BACKEND == TRUE and DISASSEMBLE-NOW == TRUE then
    Send a BECOME-BACKEND message to module in FORWARD direction.
    (switch to being a FREE module after getting the reply message.)
end
if given a BECOME-BACKEND message then
    Set IS-BACKEND = TRUE.
end
if IS-FRONTEND == TRUE and ASSEMBLE-NOW == TRUE then
    Send a BECOME-L-GROUP message to any module in FORWARD direction.
    (set IS-FRONTEND = FALSE after getting the reply message.)
end
if given a REVERSE-DIRECTION message then
    Swap the values of IS-FRONTEND and IS-BACKEND.
    Swap the values of FORWARD and BACKWARD directions.
end
```

Algorithm 1. The *L-GROUP* module algorithm. *(high level)*

```
if given a MOVE-DIRECT message then
    Execute the requested move, if legal.
end
if given a BECOME-L-GROUP message then
    Switch to being a L-GROUP module in the current location.
    Set IS-FRONTEND = TRUE.
end
```

Algorithm 2. The *FREE* module algorithm. *(high level)*

which pass by. Moving modules just follow orders and thus require almost no processing. The only remaining work is the safety checks performed when assembling/disassembling columns of L-groups. This entails (1) directing any adjacent moving modules to the right locations, and (2) agreeing with other members of the same L-group when its time to disassemble or when assembly is complete. So Robomotion needs only a constant-bounded amount of processing per L-group per module action (or action request in case a moving module freezes and ignores repeated move direction messages). Thus, we have now outlined how Robomotion meets all of our stated main goals: fast locomotion, guaranteed physical stability and connectivity, and high scalability (limited memory / processing / communication per module).

3.4 Algorithm Outline: One Tunnel

Control for Robomotion is mostly done by controlling a single tunnel and then repeating that control structure for each tunnel in the robot. We assume an initial configuration of n modules, shaped as a contiguous set of solid vertical columns with an even number > 2 of modules in each dimension. We assume the robot begins execution with each module knowing its initial role: either being a stationary *L-GROUP* module or being a moving *FREE* module. Each *L-GROUP* module begins with knowledge of the *forward* direction (initial direction of desired locomotion) and the locations of the 2 other members of its L-group. *L-GROUP* modules at the front of the robot begin with an IS-FRONTEND = TRUE status and *L-GROUP* modules at the back begin with an IS-BACKEND = TRUE status while all other modules begin with FALSE for those values. If desired, we could instead broadcast a single message with the desired direction of movement to all modules and then each module could quickly calculate its own role and L-group neighbors.

Modules may send and receive messages with neighbors to which they are directly attached. We assume these messages may be lost when communicated but, if sent successfully, they arrive instantly. Modules may delay in their movement or computation but will never fail. To handle message drops, we use a question-reply format. One module *A* will repeat its "question" message to module *B* until *B* sends back a "reply" message confirming that it successfully received *A*'s message (or until *A* no longer desires to send its initial message). Module *B* just assumes that, if its reply message is lost, there will be another chance to reply when *A* repeats its initial question message. All messages used by Robomotion are designed so no error occurs if duplicate messages are received. For control of a tunnel as a single unit, we assume an external controller (i.e. a human at a laptop) which can broadcast signals to the robot to tell it to reverse its movement direction or make an orthogonal turn. These commands are then executed by individual modules in a distributed fashion.

Given the initial movement direction chosen, our goal is to generate locomotion in that direction indefinitely, or until a REVERSE-DIRECTION request is received. For each request, the robot will eventually succeed in reversing the direction of locomotion throughout every tunnel. At all times the robot will be in a configuration which is physically stable in the presence of gravity.

The psuedo-code shown in Algorithms 1 and 2 we give a high-level view of how modules interact within a tunnel. In that code, a "legal" move for a *FREE* module is one where it moves into an unoccupied space and has a solid path to travel into that space (i.e. a flat surface made by 2 adjacent *L-GROUP* modules). The DISASSEMBLE-NOW and ASSEMBLE-NOW variables are explained in the next subsection. Later, subsection 3.6 gives further descriptions of how we execute orthogonal turns or handle situations where only some modules receive a given broadcast signal from the external controller. Full details and proofs are omitted here due to length requirements.

3.5 Algorithm Outline: Connecting Adjacent Tunnels

With a working algorithm to generate locomotion with one tunnel, we now only need to keep adjacent tunnels moving at about the same rate. We do this by passing messages through a slice whenever an L-group determines that it has finished assembling or is safe to be disassembled. Algorithm 3, which would also be run by *L-GROUP* modules, shows how this works for disassembly. Whenever a module

```
if a DIS-READY message is received from above then
    Set DIS-READY-UP = TRUE.
end
if a DIS-READY message from a side direction then
    Set DIS-READY-SIDE = TRUE.
end
if DIS-READY-UP == TRUE and DIS-READY-SIDE == TRUE and no adjacent modules
in BACKWARD direction then
    Set DISASSEMBLE-NOW = TRUE.
    Send a DIS-READY message to the L-group modules below.
    Send a DIS-READY message to the L-group modules in side directions.
end
```

Algorithm 3. The *L-GROUP* module collaboration algorithm. *(high level)*

switches to an *L-GROUP* algorithm, it initializes DIS-READY-UP = TRUE if there are no other L-groups above it and DIS-READY-SIDE = TRUE if there is no other L-group to one side of its L-group. Otherwise these values are initialized to FALSE. A process similar to this would be done for assembly and the ASSEMBLE-NOW safety check variable.

3.6 Other Challenges Solved

Convex-corner Transitions. Many existing hardware implementations cannot perform convex-corner turns and using several individual modules to form groups of fully functional "meta-modules" is expensive and unwieldly. However, there are two viable options for getting individual modules around a convex-corner. Option 1 is to have that module travel along a 3rd surface which is orthogonal to the surfaces making that corner. Figure 5 shows an example of this. Option 2 would be for a second moving module to come and push the first module past that convex-corner, or pull the first module back from that corner.

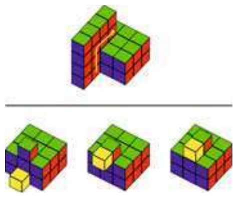


Fig. 5 Options 1 (top) and Option 2 for aiding convex-corner transitions.

For Robomotion, the only convex-corner transitions needed are when an L-group is being disassembled or assembled. We have 2 methods that can allow these corner movements, one method for each of our 2 options given previously. Method 1 is to have at least 3 modules "beyond" the last L-group in the tunnel. That is, behind the back of the tunnel (and back of the robot) or in front of the front of the tunnel/robot. In this case two of those modules, along with the next L-group in the tunnel, act as a 3rd orthogonal surface for other modules to travel in or out of the tunnel. For our second method, we simply have a *FREE* module wait at the end of the tunnel, making a flat surface at the back of the robot with the last L-group in that tunnel. Now any module "beyond" that L-group can slide next to that waiting *FREE* module, and the waiting module could pull it into the tunnel. Conversely, a waiting module at the front of the tunnel could be pushed out beyond the frontmost L-group by another *FREE* module coming behind it in the tunnel.

Making orthogonal turns Conceptually, making a tunnel turn is simple: change one *L-GROUP* module to *FREE* and when it moves a gap is formed to start a new tunnel. In practice we need modules in the frontmost 2 L-groups to reset their roles, forming 2 new L-groups facing in the desired orthogonal direction. Figure 6 shows an example of this. One of those new L-groups will also communicate with the last L-group in the old tunnel to eventually take over the IS-BACKEND status and disassemble that last L-group in the old tunnel. Finally, to coordinate multi-tunnel turns, only every other L-group is marked as a "valid" turn starter. Since tunnels have a 2-module width, this prevents the formation of incompatible parallel tunnels due to an "off by one" error.

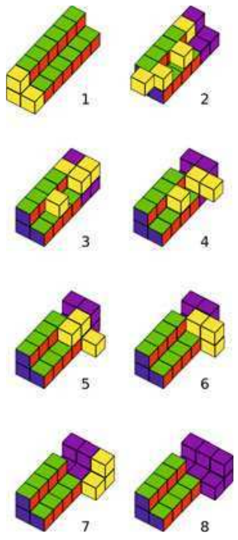


Fig. 6 A tunnel beginning a turn. Yellow modules are *FREE* and purple modules are part of the new orthogonal tunnel.

Module Delays. Movement through the interior of the robot is acyclic (forward or backward in a tunnel), so no deadlock can occur. Similarly, dependancies between adjacent L-groups (for safety checks) are 1-directional and so deadlock is avoided. Finally, within an L-group, 1 module (designated by being in the middle) makes all the group decisions. It acts on these decisions after getting confirmation from the other 2 L-group members, so no race conditions or deadlocks occur within an L-group. Thus, Robomotion can withstand delays in module movement or computation without error.

Delays of external messages. For our model, we anticipate that an external control may be used to send "change locomotion direction" messages to the robot as a whole. The robot is composed of many individual modules and so there could be delays in passing such an external message to all modules in the robot. To handle these delays, we place a time stamp on all external messages. This time stamp is the only use of memory that would not be constant-bounded. However, even this is bounded if modules are trusted to respond to such external messages within some large time span, after which we can reset the time stamp counter to zero.

3.7 Shape and Terrain Limitations

Robomotion can achieve stable locomotion for any shape that can be composed from vertical towers of modules with small overhangs. This is because setting a constant bound on the shear or tension force experienced by any module also sets a constant bound on the length of any overhang coming off any tower (given the weights of modules in that overhang). Any physically stable configuration of modules faces this limitation. Stronger materials or careful arrangement of modules could increase the constant factor length. However, the overhang would be bounded by a constant regardless.

Considering terrain with obstacles, Robomotion can travel around obstacles but not as naturally as algorithms like Waterflow or MMM. If Robomotion is confronted with a few obstacles, it can turn to move around them or could "turn" one tunnel into an adjacent one, merging them, to flow through small gaps. However, for a complex maze or "briar patch" of obstacles, then it may not be worth constructing Robomotion's high-speed tunnels. In these cases we may have to accept that there is no fast way to travel through and resort to using the Waterflow method (so we can still have physical stability) until we've moved past those obstacles. The situation is like building highways in a mountainous region: the faster we want to go, the straighter and smother the road needs to be.

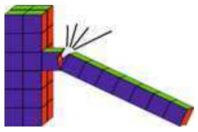


Fig. 7 Very long overhangs will likely break.

4 Simulation Results

In addition to our theoretical results, we have also simulated our Robomotion algorithm to experimentally verify its performance. Our simulation is written in Java and can optionally use Java3D to display the robot modules. The basis of this simulation was meant to aid us in making comparisons to the leading locomotion algorithm in prior work: the Million Module March. While our simulation runs on a serial computer, it is a distributed rather than centralized implementation in that each module executes its own independent algorithm. We enforce that the only communication between modules is message passing between adjacent, connected modules. For sensing, a module A (which is cube shaped) can detect if it has an adjacent neighbor on any of its 6 sides or if it can safely move forward into an adjacent lattice location. Since our modules can not make convex-corner transitions, this means that the space is open and that there is a flat surface for travel (i.e. adjacent neighbor modules) between A's current location and the desired new lattice location.

Modules do not know their global positions. The total robot configuration and shape is not known by any module and is not known by any external controller. Thus, each module is forced to act based only on local information or on messages received from adjacent modules. Also, since there is no global knowledge of module positions, any external controller cannot specify exact placements for modules in a desired configuration. Instead, we allow less exact specifications like direction and distance for locomotion or the desired number of parallel tunnels at the front of the robot. This method for controlling the robot is more limiting, but we believe it is more realistic for a scalable system in real hardware since it allows us to use only a constant amount of memory within each module.

Since our simulation runs on a single processor, the modules of course must execute in a serialized fashion. However, to make this more realistic, at the start of the simulation a randomized order is chosen for the modules. Our simulation then executes in iterations. Within each iteration the modules execute in the randomized order that was chosen and, when its turn comes, each module executes its current algorithm.

To simulate message passing, a message sent from module A to module B will be placed on a queue for B when A executes its algorithm. Module B will then read all messages on its queue whenever it has its next turn to execute its own algorithm. Note that this implementation actually makes Robomotion appear slower than it would be when executed on actual hardware. This is because messages sent between modules typically take 1 iteration for travel, the same travel time as a physical module moving between adjacent lattice locations. This communication delay occurs more in Robomotion than in other locomotion algorithms because of our question-reply format for handling messages (used to avoid errors due to dropped messages).

4.1 Speed Comparisons

To make our comparison as direct as possible, we executed simulations for the same test that was performed for the Million Module March algorithm. We ran cube-shaped collections of n modules a distance of $n^{1/3} - 1$ module lengths over flat ground. This distance was chosen by the MMM authors because $n^{1/3}$ is the length of 1 side of the robot's cubic shape and they wanted a 1-module length overlap between the start and goal locations for the robot. Tests were run for different values of n to see how the algorithm performed for different robot sizes. The most important speed statistic for a locomotion algorithm is probably how long it takes the robot to travel a single module-length or "unit" distance. Thus, we took the total number of time steps taken by a robot during a locomotion test and divided by the $n^{1/3} - 1$ distance

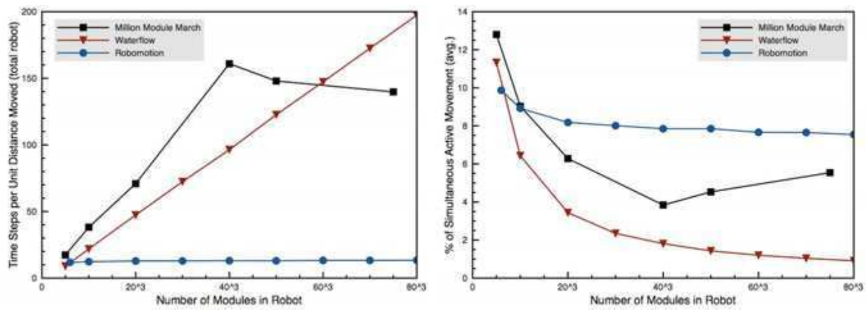


Fig. 8 Left: Robomotion stays fast even with huge numbers of modules in the robot. **Right:** Robomotion SAM rate remains high even as the number of modules grows large.

traveled for robots of size n. Statistics for Robomotion come from our simulation while numbers for MMM are taken from its own publication. [5]

For this test we have also simulated a simple implementation of the "Waterflow" style of locomotion to use as a baseline comparison along with MMM and Robomotion. As mentioned earlier in our Related Work Section (Section 2), Waterflow is a style of algorithm where modules at the back of the robot slide up, slide across the top of the robot, and then slide down the front to become the new front of the robot. In this way, the modules tend to "flow" like water toward the desired goal location. Many different research groups have devised different ways to control this style of locomotion, yet any implementation should have roughly the same module movements and so should give roughly the same locomotion speed (differing only by small constant factors).

In Figure 8 we illustrate the statistical speed comparison between these 3 key locomotion algorithms. Here a "time step" is the time for a single module to move a single module length (unit distance). Experimentally, we found that the running time for the Waterflow algorithm had near-perfect linear growth compared to $n^{1/3}$, where n is the number of modules in the robot. Up through robots of size 40^3 MMM had similar running time growth, but was less efficient than Waterflow. However, from $n = 40^3$ to $n = 75^3$ MMM actually had a slight speed-up. We're not sure of the cause, but one educated guess is that traveling a longer distance (49 and 74 unit distances for the last 2 data points shown) gave modules in the robot enough time to spread out into a flatter robot shape. The dense cube shape of the initial configuration probably limited movement.

Meanwhile, the running time for our Robomotion algorithm is a stark contrast to the other two. We maintain near-constant speed for all values of n that we tested. Specifically, the running time hovered between 12.33 time steps per unit distance traveled for $n = 10^3$ and 13.43 for $n = 80^3 = 512,000$ modules. This consistent performance is not surprising since each "tunnel" through the robot is predominantly independent of all others. The only slight delays are the message checks sent between tunnels to keep any one from getting ahead of (or falling behind) in its movement speed compared to adjacent tunnels. Experimentally, there was very little effect (less than 1 time step per unit distance traveled) when we added these checks to our algorithm. If implemented on actual hardware, we expect the effect would be even smaller since messages should travel much faster than moving modules but our simulation used 1 time step for either action. On hardware, we also expect that Robomotion and Waterflow would both gain in speed since these algorithms have modules travel long straight distances, from the robot's back to its front. These modules could increase movement speed or conserve momentum by not stopping during this trip since there's no need to re-plan their travel path.

In addition to the speed of the robot, we also looked at the Simultaneous Active Movement (SAM) rate for these 3 locomotion algorithms. Recall that we defined this measure in Section 3 as the ratio of the number of moving modules compared to the total number of modules in the robot. Figure 8 shows the average percentage of modules that were moving at any given time, which is basically just taking the total number of actuations made and dividing by the total time steps taken and by the total number of modules. The results in this graph are very similar to the previous graph. Waterfow and MMM both start with good SAM percentages, but lose efficiency as the number of modules in the robot grows large. This time it is Waterflow which is slightly less efficient, and beyond $n = 40^3$ there is again a slight improvement for MMM. In contrast to these effects, Robomotion once again remains steady, and fast, ranging from about 9.8% to 7.5% as the number of modules increase. We note that while Robomotion is substantially better on both metrics, its advantage on raw locomotion speed seems slightly better than its SAM rate advantage. A likely cause is that Robomotion becomes more efficient with longer tunnels, and so for large values of n (when the cube shape is longer in each dimension) it was able to be nearly as fast even while having a very slight drop in SAM percentage.

5 Conclusion

In this paper we have presented a novel locomotion algorithm for lattice-style self-reconfigurable robots. This algorithm is highly scalable, produces movement which is more efficient than prior locomotion approaches, and always keeps the robot physically stable in the presence of gravity. We do this without using convex-corner transitions and withstand possible failures in message passing and delays in module execution. Thus, we believe this to be a good step toward developing scalable control algorithms which would actually work on real hardware implementations.

For future work, the way in which we make orthogonal turns is one potential area. Our current algorithm focuses on individual tunnels, so turning multiple tunnels can be inefficient. Extending our algorithm to very rough terrain is another important step, but is difficult for any physically stable algorithm. As we've described, straight support columns must be maintained for any configuration to minimize shear and tension forces. A likely compromise may be a heuristic-based algorithm that uses Robomotion for flat or semi-rough terrain but reverts to Waterflow when faced with highly irregular terrain (where high-speed locomotion may be impossible anyway).

References

1. Brandt, D., Christensen, D.J.: A new meta-module for controlling large sheets of atron modules. In: IROS (October 2007)
2. Brandt, D., Østergaard, E.H.: Behaviour subdivision and generalization of rules in rule-based control of the atron self-reconfigurable robot. In: International Symposium on Robotics and Automation (ISRA), pp. 67–74 (August 2004)
3. Butler, Z., Byrnes, S., Rus, D.: Distributed motion planning for modular robots with unit-compressible modules. In: IROS (June 2001)
4. Butler, Z., Kotay, K., Rus, D., Tomita, K.: Generic decentralized control for a class of self-reconfigurable robots. In: IEEE International Conference on Robotics and Automation (ICRA) (August 2002)
5. Fitch, R., Butler, Z.: Million module march: Scalable locomotion for large self-reconfiguring robots. The International Journal of Robotics Research (December 2008)

6. Kotay, K., Rus, D.: Generic distributed assembly and repair algorithms for self-reconfiguring robots. In: IROS (June 2004)
7. Kurokawa, H., Yoshida, E., Tomita, K., Kamimura, A., Murata, S., Kokaji, S.: Self-reconfigurable m-tran structures and walker generation. Robotics and Autonomous Systems 54, 142–149 (2006), central pattern generator
8. Moll, M., Will, P., Krivokon, M., Shen, W.-M.: Distributed control of the center of mass of a modular robot. In: IROS (December 2006)
9. Murata, S., Kamimura, A., Kurokawa, H., Yoshida, E., Tomita, K., Kokaji, S.: Self-reconfigurable robots: Platforms for emerging functionality. In: Iida, F., Pfeifer, R., Dong, Z., Kuniyoshi, Y. (eds.) Embodied Artificial Intelligence. LNCS (LNAI), vol. 3139, pp. 312–330. Springer, Heidelberg (2004)
10. Ravichandran, R., Gordon, G., Goldstein, S.C.: A scalable distributed algorithm for shape transformation in multi-robot systems. In: IROS (December 2007)
11. Sastra, J., Chitta, S., Yim, M.: Dynamic rolling for a modular loop robot. International Journal of Robotics Research (IJRR) (2007)
12. Shen, W.-M., Krivokon, M., Chiu, H.C.H., Everist, J., Rubenstein, M., Venkatesh, J.: Multimode locomotion via superbot robots. In: IEEE International Conference on Robotics and Automation (ICRA) (April 2006)
13. Støy, K.: Controlling self-reconfiguration using cellular automata and gradients. In: International Conference on Intelligent Autonomous Systems (IAS) (December 2004)
14. Støy, K., Nagpal, R.: Self-repair through scale independent self-reconfiguration. In: IROS (September 2004)
15. Vassilvitskii, S., Yim, M., Suh, J.: A complete, local and parallel reconfiguration algorithm for cube style modular robots. In: IEEE International Conference on Robotics and Automation (ICRA) (March 2002)
16. Yoshida, E., Murata, S., Kamimura, A., Tomita, K., Kurokawa, H., Kokaji, S.: A self-reconfigurable modular robot: Reconfiguration planning and experiments. International Journal of Robotics Research (IJRR) 21(10-11), 903–915 (2003)
17. Zhang, Y., Yim, M., Eldershaw, C., Duff, D., Roufas, K.D.: Phase automata: A programming model of locomotion gaits for scalable chain-type modular robots. In: IROS (August 2003)

Adaptive Time Stepping in Real-Time Motion Planning

Kris Hauser

Abstract. Replanning is a powerful mechanism for controlling robot motion under hard constraints and unpredictable disturbances, but it involves an inherent tradeoff between the planner's power (e.g., a planning horizon or time cutoff) and its responsiveness to disturbances. We present a real-time replanning technique that uses adaptive time stepping to learn the amount of time needed for a sample-based motion planner to make monotonic progress toward the goal. The technique is robust to the typically high variance exhibited by planning queries, and we prove that it is asymptotically complete for a deterministic environment and a static objective. For unpredictable environments, we present an adaptive time stepping contingency planning algorithm that achieves simultaneous safety-seeking and goal-seeking motion. These techniques generate responsive and safe motion in simulated scenarios across a range of difficulties, including applications to pursuit-evasion and aggressive collision-free teleoperation of an industrial robot arm in a cluttered environment.

1 Introduction

Robots must frequently adjust their motion in real-time to respond to unmodeled disturbances. A common approach to deal with nonlinear dynamics and hard state and control constraints is to reactively replan at each time step (Figure 1). This basic approach has been studied under various nomenclature (model predictive control, receding horizon control, or real-time planning) and using various underlying planners (numerical optimization, forward search, or sample-based motion planners), and is less susceptible to local minima than myopic potential field approaches. But replanning, in all its forms, faces a fundamental tradeoff based on the choice of time limit: too large, and the system loses responsiveness; too short, and the planner may fail to solve

Kris Hauser
School of Informatics and Computing, Indiana University
e-mail: `hauserk@indiana.edu`

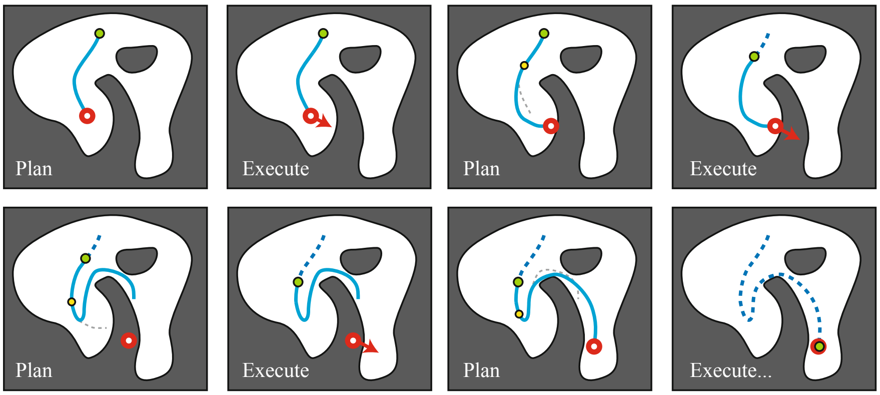


Fig. 1 A point robot (green) interleaves execution and replanning to reach an unpredictably moving target (red).

difficult problems in the allotted time, which sacrifices global convergence and safety. Empirical tuning by hand is the usual approach. But the time needed to solve a planning query can vary by orders of magnitude not only between problems, but also between different queries in the same problem, and even on the same query (in the case of randomized planners). Unless variability is addressed, the safety and completeness of real-time replanning is in doubt.

This paper presents two replanning algorithms that address safety and completeness not by reducing variability in planning time, but by tolerating and adapting to it. They use a sample-based planner to build partial plans whose endpoints monotonically improve an objective function, and adaptively learn a suitable time step on-the-fly by observing whether the planner is able to make progress within the time limit. The first algorithm, described in Section 3, guarantees safe motion in deterministic, predictable environments by construction, and furthermore we prove that the state of the robot is guaranteed to globally optimize the objective function in expected finite time for a large class of systems. We apply it to real-time obstacle avoidance for a simulated 6DOF industrial robot moving dynamically in a cluttered environment. The second algorithm, described in Section 4, uses a conservative contingency planning approach to achieve a higher probability of safety in unpredictable or adversarial environments, and we apply it to a pursuit-evasion problem. Experiments suggest that adaptive time stepping is more consistent than constant time stepping across problem variations for both algorithms.

2 Related Work

Bounded Rationality in Real-Time Agents. Real-time planning architectures have a long history of study in artificial intelligence, control theory, and robotics, but few have explicitly addressed the problem of "bounded rationality", where limited computational resources hamper an agent's ability to produce timely, optimal plans. A notable exception is the CIRCA real-time agent architecture [12] that separates the agent's control into high-level planning and low-level reactive control tasks. The high-level task conveys controller specifications to the low-level task whenever planning is complete. The disadvantage of this approach is that uncertainty in computation time causes uncertainty in state, leading to harder planning problems. By contrast our approach is constructed to avoid state uncertainty, at least when the system is deterministic, which makes planning more tractable.

Replanning Applications and Implementations. Model predictive control (MPC), aka receding horizon control, is a form of replanning that at each time step formulates a optimal control problem truncated at some horizon. Such techniques have been successful in robot navigation [2, 16]; for example the classic dynamic windowing technique introduced for indoor mobile robot navigation is essentially MPC by another name [16]. In nonlinear systems, truncated optimal control problems are often solved using numerical optimization or dynamic programming [1, 11]. In discrete state spaces, efficient implementations of replanning algorithms include the D* and Anytime A* algorithms which are based on classic heuristic search [10, 17, 18].

Sample-based motion planners such as randomly-exploring random trees (RRTs) and expansive space trees (ESTs) have been applied to real-time replanning for dynamic continuous systems [4, 5, 7, 20]. RRT and EST variants have been applied to 2D helicopter navigation [5], free-floating 2D robots [7], and and car-like vehicles [13] among moving obstacles, as well as exploring an unknown environment [3]. Our algorithms also use sample-based planners.

Time Stepping in Replanning. Although many authors have proposed frameworks that can handle nonuniform time steps [3, 5, 13, 20], few actually *exploit* this capability to adapt to the power of the underlying planner. We are aware of one paper in the model predictive control literature [14] that advances time exactly by the amount of time taken for replanning. The weakness of this approach is that if replanning is slow, the actions taken after planning are based on outdated state estimates, leading to major instability and constraint violations. Our work avoids this problem by setting planner cutoffs and projecting state estimates forward in time at the start of planning.

Safety Mechanisms. Several mechanisms have been proposed to improve the safety of replanning in dynamic environments. Feron et al introduced the notion of τ-safety, which indicates that a trajectory is safe for at least time τ [5]. Such a certificate establishes a hard deadline for replanning. Hsu et al introduced the notion of an "escape trajectory" as a contingency plan that is taken in case the planner fails to find a path that makes progress toward the goal [7]. We use a contingency planning technique for unpredictable environments that is much like a conservative escape trajectory approach, except that it always ensures the conservative path is followed in case of a planning failure.

The notion of inevitable collision states (ICS) was introduced by Petti and Fraichard to the problem of real-time planning for a car-like vehicle among moving obstacles [13]. An ICS is a state such that no possible control can recover from a collision, and considering ICS as virtual obstacles prevents unnecessary exploration of the state space. In practice, testing for ICS can only be done approximately, and the conservative test proposed in [13] may prevent the robot from passing through states that are actually safe. Our work provides similar safety guarantees without explicit testing for ICS.

Speeding up Replanning. Many approaches have sought to improve responsiveness by simply reducing average replanning time. Some common techniques are to reuse information from previous plans [4], to use precomputed coarse global plans to essentially reduce the depth of local minima [2, 8, 16, 19], or a combination [3, 20]. These approaches are mostly orthogonal to the choice of time step and can be easily combined with adaptive time stepping.

3 Replanning in Deterministic Environments

In real-time replanning the robot interleaves threads of *replanning* and *execution*, in which the robot (at high rate) executes a partial trajectory that is intermittently updated by the replanning thread (at a lower rate) without interrupting execution. The planner is given a time cutoff Δ, during which it generates a new safe trajectory originating from a state propagated in the future by time Δ. This section presents and analyzes the adaptive time-stepping technique and an application to real-time assisted teleoperation of a robot manipulator in deterministic environments.

3.1 Assumptions and Notation

The state of the robot x lies in a state space S, and its motion must obey differential constraints $\dot{x} \in U(x,t)$ (note that this is simply a more compact way of writing control constraints). We assume that the robot has a possibly imperfect model of the environment and how it evolves over time, and let $F(t) \subseteq S$ denote the subset of feasible states at time t. We say that a trajectory $y(t)$ is *τ-safe* if $\dot{y}(t) \in U(t)$ and $y(t) \in F(t)$ for all $0 \leq t \leq \tau$. If so, we say $y(t)$ is an F_τ trajectory.

In this section we will be concerned primarily with F_∞ trajectories. We will assume that F_∞ feasibility is achieved by ensuring that each trajectory terminates at a feasible stationary state. For certain systems with dynamics, such as cars and helicopters, a “braking” control can be applied. This paper will not consider systems like aircraft that cannot reach zero velocity, although terminal cycles may be considered as a relatively straightforward extension.

We address the problem of reaching a global minimum of a smooth time-invariant potential function $V(x)$ via an F_∞ trajectory $y(t)$ starting from the initial state x_0. Assume the global minimum is known and attained at $V(x) = 0$ without loss of generality. We say any trajectory that reaches $V(x) = 0$ is a *solution trajectory.* We do not consider path cost, and define the cost functional $C(y)$ that simply returns the value of $V(x)$ at the terminal state of the trajectory y. It is important to note that when we refer to an optimal solution, we are referring to the *optimality of the terminal point*, not the trajectory taken to reach it. We also impose the *real-time constraint* that no portion of the current trajectory that is being executed can be modified. So, if a replan is instantiated at time t and is allowed to run for time Δ, then no portion of the current trajectory before time $t + \Delta$ may be modified.

We assume that we have access to an *underlying planner* with the following "any-time" characteristics:

1. The planner iteratively generates F_∞ trajectories starting from an initial state and time $y(t_0)$ given as input.
2. Planning can be terminated at any time, at which point it returns the trajectory that attains the least value of the cost functional $C(y)$ found so far.
3. If the planner is given no time limit on any query that admits a solution trajectory, then the planner finds a solution in expected finite time.

A variety of underlying planning techniques (e.g., trajectory optimization, forward search, and sample-based motion planning) can be implemented in this fashion. All experiments in this paper are conducted with minor variants of the sampling-based planners RRT [9] and SBL [15], which grow trees using forward integration of randomly-sampled control inputs. The running time of such planners is variable across runs on a single query, and can vary by orders of magnitude with the presence of narrow passages in the feasible space.

3.2 Adaptive Time-Stepping with Exponential Backoff

Here we describe our variable-time step replanning algorithm and a simple but effective exponential backoff strategy for learning an appropriate time step. Pseudocode is listed in Algorithm 1 in Figure 2. The replanning thread takes time steps $\Delta_1, \Delta_2, \ldots$. In each time step, the planner is initialized from the state on the current trajectory at time $t_k + \Delta_k$, and plans until Δ_k time has elapsed (Line 2). If the planner finds a trajectory with lower cost than the current trajectory (Line 3) then the new trajectory is spliced into current trajectory at the junction $t_k + \Delta_k$ (Line 4). Otherwise, the current trajectory is left unaltered and replanning repeats.

```
Algorithm 1. Replanning with an Adaptive Time-Step
Initialization:
0a. $y(t)$ is set to an $F_\infty$ initial trajectory starting from $t = 0$.
0b. $\Delta_1$ is set to a positive constant.

Repeat for $k = 1, \ldots$:
1. Measure the current time $t_k$
2. Initialize a plan starting from $y(t_k + \Delta_k)$, and plan for $\Delta_k$ time
3. If $C(\hat{y}) \leq C(y) - \epsilon$ for the best trajectory $\hat{y}(t)$ generated so far, then
4.     Replace the section of the path after $t_k + \Delta_k$ with $\hat{y}$
5.     Set $\Delta_{k+1} = 2/3\Delta_k$
6. Otherwise,
7.     Set $\Delta_{k+1} = 2\Delta_k$
```

Fig. 2 Pseudocode for the replanning algorithm.

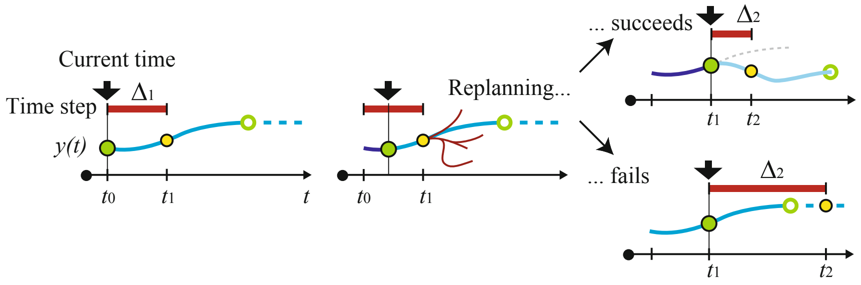


Fig. 3 Each replanning iteration chooses a time step (left), initiates a plan starting from the predicted future state (center), and either succeeds or fails. Upon success, the robot progresses on the new plan and the time step is contracted. Upon failure, the robot retains the original plan and the time step is increased.

Note that the condition in Line 3 requires a decrease by some small constant $\epsilon > 0$. (To allow the planner to reach the global minimum exactly, an exception can be made on the final step when $C(\hat{y}) = 0$ is attained.) This simplifies later analysis by preventing the theoretical occurrence of an infinite number of infinitesimal cost improvements.

Lines 5 and 7 implement a simple exponential backoff strategy for choosing the time cutoff. This permits recovery from a local minimum of $V(x)$ in case several planning failures are encountered in sequence. Such strategies are widely used in protocols for handling network congestion, and there is a rough analogy between uncertainty in planning time and uncertainty in message delivery over an unreliable network. The idea is simple: if the planner fails, double the time step (Line 7). If it succeeds, contract the time step (Line 5). The constant 2/3 that we use in the contraction strategy does not need to be chosen particularly carefully; resetting Δ_{k+1} to a small value works well too. Figure 3 illustrates one iteration of the protocol.

3.3 Completeness and Competitiveness

We can now state a basic theorem that guarantees that Algorithm 1 is probabilistically complete for static goals as long as the robot never reaches a state where the goal becomes unreachable.

Theorem 1. *If the environment is deterministic and perfectly modeled, and the goal is reachable from any state that is reachable from the start, then Algorithm 1 will find a solution trajectory in expected finite time.*

Proof. Let $\mathcal{R}$ be the set of states reachable from the start, and let $T(x)$ be the expected planning time for finding a solution trajectory starting at $x \in \mathcal{R}$. Because of the assumption in Section 3.1, $T(x)$ is finite, and so is the maximum of $T(x)$ over all $\mathcal{R}$, which we denote to be T_{max}. First we will show that the time until a plan update has a finite expected value.

Suppose Algorithm 1 has its first plan update on the k'th iteration after $k-1$ failed iterations (the possibility that no such k exists is vanishingly small). Because $k-1$ iterations have passed without an update, the planning cutoff on the k'th iteration is $2^k \Delta_0$. So the total time spent T over all the k iterations is a geometric series with sum $T = (2^{k+1} - 1)\Delta_0$. Let T_p be the random variable denoting the planning time necessary to find a global solution starting at $(y(t_k), t_k)$. If k were known, then T_p would be lie in the range $(2^{k-1}\Delta_0, 2^k \Delta_0]$, so that the inequality $T < 4T_p$ holds. But the inequality $E[T_p] \leq T_{max}$ holds for all k, x_k, and t_k, so $E[T] < E[4T_p] \leq 4T_{max}$ unconditionally.

The number N of plan updates needed to reach a global minimum is finite since the initial trajectory has finite cost, and each plan update reduces cost by a significant amount. So, the total expected running time of Algorithm 1 is bounded by $4NT_{max}$, which is finite. □

We remark that the bound $4NT_{max}$ is extremely loose, and seemingly poor compared to the performance bound T_{max} of simply planning from the initial state until a solution is found. In practice, most problems contain few planning queries of extremely high difficulty corresponding to escaping deep local minima of C, and the running time of Algorithm 1 will tend to be dominated by those queries. Smaller, greedy advances in $C(y)$ are often much quicker to plan.

By construction Algorithm 1 will never drive the robot to an inevitable collision state (ICS) as defined in [13], so it is equivalently "safe". But in which systems is it asymptotically complete? The key assumption of Theorem 1 is that the goal can be reached by all states in $\mathcal{R}$ (this can be slightly weakened to take $\mathcal{R}$ as those states actually reached by the robot during execution). For example, it holds in reversible systems. In general, however, Algorithm 1 might inadvertently drive the robot into a dead end from which it cannot escape — a condition we might call an *inevitable failure state*. In fact, non-reversible systems seem to prove quite troublesome to all replanning techniques because detecting dead ends requires sufficient global foresight that

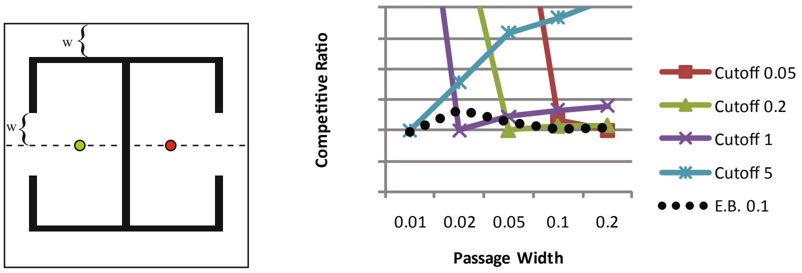


Fig. 4 (a) Our 2D benchmark problem. Passage width, and hence, difficulty, is parameterized by w. (b) The performance of constant cutoff strategies varies greatly across passage widths, and short cutoffs (0.05, 0.1, and 0.2) fail completely on difficult problems. The adaptive exponential backoff strategy (E.B.) achieves consistent performance across problem variations. Performs is measured by solution time, normalized by the time of the best constant cutoff for that problem.

might not be practical to achieve with limited computation time. These are fruitful directions for future work.

As a final note, we remark that Algorithm 1 is not necessarily complete in problems with time-varying potential V. For example, if the global minima of V might alternate quickly between two locally easy but globally difficult problems, then the algorithm will forever be able to make local progress and will thereby keep the time step short.

3.4 Completeness and Sensitivity Experiments

We evaluated the performance of the adaptive strategy against constant time stepping strategies on a static 2D benchmark across varying problem difficulties. Consider a unit square state space S where the state is subject to velocity constraints $||\dot{x}|| \leq 1$. Obstacles partition the feasible space $F(t)$ into two "rooms" with opposite-facing doorways, which are connected by hallways (see Figure 4). The state must travel from $(0.3, 0.5)$ to $(0.6, 0.5)$, and the potential function $V(x)$ simply measures the distance to the goal. For replanning we use a unidirectional RRT planner [9], which, like other sample-based planners, is sensitive to the presence of narrow passages in the feasible space. We control the difficulty of escaping local minima by varying a parameter w, and set the hallway widths to w and the doorway widths to $2w$.

We measure performance as the overall time taken by the robot to reach the goal, averaged over 10 runs with different random seeds. If it cannot reach the goal after 120 s, we terminate the run and record 120 s as the running time. Experiments compared performance over varying w for constant cutoffs 0.05, 0.1, 0.2, 0.5, 1, 2, and 5 s and the exponential backoff algorithm starting with $\Delta_1 = 0.1$ (performance was relatively insensitive to choice of Δ_1).

Figure 4 plots the performance ratios of several strategies. Performance ratio is measured relative to the best constant time step for that problem.

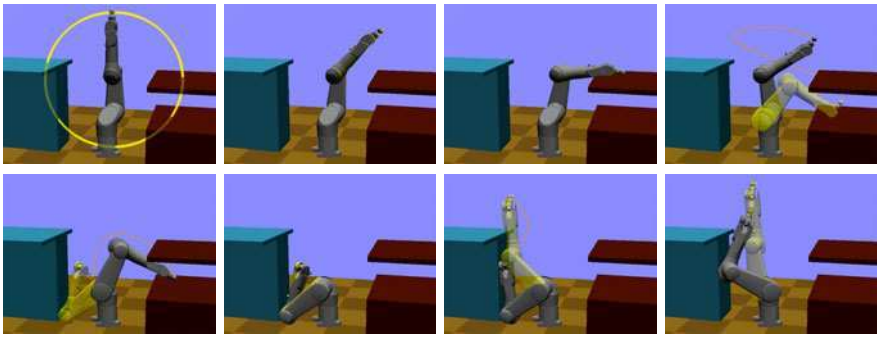


Fig. 5 A Staübli TX90L manipulator is commanded in real time to move its end effector in a clockwise circle in a cluttered environment. The robot responds reactively to the target's motion. Along the upper semicircle, rapid replanning with a short time step allows the target to be followed closely. When obstacles are encountered on the lower semicircle, planning becomes more difficult. Adaptive time stepping gives the planner sufficient time to enter and escape deep narrow passages. The current plan is drawn in orange, and its destination configuration is drawn transparently.

Shorter cutoffs are unreliable on hard problems because the planner is unable to construct paths that escape the local minimum of the initial room. On the other hand, longer cutoffs waste time on easier problems. The adaptive strategy delivers consistent performance, performing no worse than 1.4 times that of the best constant cutoff across all problems.

3.5 Assisted Teleoperation Experiments on a 6DOF Manipulator

Replanning interleaves planning and execution, so motion appears more fluid than a pre-planning approach. This is advantageous in human-robot interaction and assisted teleoperation applications where delays in the onset of motion may be viewed as unnatural. We implemented a teleoperation system for a dynamically simulated 6DOF Staübli TX90L manipulator that uses replanning for real-time obstacle avoidance in assisted control. The robot is able to reject infeasible commands, follow commands closely while near obstacles, and does not get stuck in local minima like potential field approaches.

In this system, an operator controls a 3D target point (for example, using a joystick or a laser pointer), and the robot is instructed to reach the point using its end effector. The robot's state space consists of configuration $\times$ velocity, and its acceleration and velocity are bounded. Its configuration are subject to joint limit and collision constraints. The objective function for the planner is an unpredictably time-varying function $V(x,t)$ which measures the distance from the end effector to the target point.

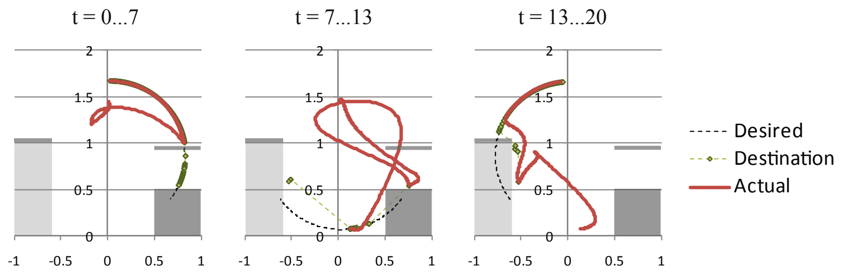


Fig. 6 Traces of the end effector's desired position (Desired), the position at the current plan's destination configuration (Destination), and actual position as executed by the robot (Actual) for the experiment in Figure 5.

Our underlying planner is a unidirectional variant of the SBL motion planner [15] that is adapted to produce dynamically feasible paths. We made the following adjustments to the basic algorithm:

- We extend the search tree by sampling extensions to stationary configurations sampled at random. The local planner constructs dynamically feasible trajectories that are optimal in obstacle free environments (a similar strategy was used in [5]). To do so, we use analytically computed trajectories that are time-optimal under the assumption of box-bounds on velocity and acceleration [6].
- For every randomly generated sample, we generate a second configuration using an inverse kinematics solver in order to get closer to the target.
- SBL uses a lazy collision checking mechanism that improves planning time by delaying edge feasibility checks, usually until a path to the goal is found. We delay edge checks until the planner finds a path that improves $C(y)$.
- To improve the fluidity of motion, we devote 20% of each time step to trajectory smoothing. We used the shortcutting heuristic described in [6] that repeatedly picks two random states on the trajectory, constructs a time-optimal segment between them, and replaces the intermediate portion of the trajectory if the segment is collision free.

The simulation environment is based on the Open Dynamics Engine rigid-body simulation package, where the robot is modeled as a series of rigid links controlled by a PID controller with feedforward gravity compensation and torque limits. The simulation does perform collision detection, but in our experiments the simulated robot did not collide with the environment.

We simulated a user commanding a target to follow a circular trajectory that passes through the robot and obstacles (Figure 5). The circle has radius 0.8 m and a period of 20 s. The upper semicircle is relatively unconstrained and can be followed exactly. Targets along the lower semicircle are significantly harder to reach; at several points they pass through obstacles, and at other points they require the robot to execute contorted maneuvers through narrow passages in the feasible space. Experiments show that replanning can reach a large portion of the lower semicircle while tracking the upper semicircle nearly perfectly (Figure 6).

4 Replanning in Unpredictable Environments

A conservative approach to uncertainty may be preferred in safety-critical applications like transportation and medical robotics. This section presents a real-time contingency planning algorithm that generates both optimistic (goal seeking) and pessimistic (safety seeking) trajectories to balance safety-seeking and goal-seeking behavior. Adaptive time stepping allows for a high probability of replanning before a certain time limit — the time to potential failure, or TTPF — in which safety is guaranteed. Experiments evaluate the system in a pursuit-evasion scenario.

4.1 Conservative Replanning Framework

We assume that we have access to conservative bounds on the uncertainty of the environment, and let E_k denote the environment model estimated by the robot's sensors at k'th time step. Let $F(t; E_k)$ denote the feasible set with the current model, and let $\tilde{F}(t; E_k)$ denote the set of states that is guaranteed to be feasible at time t under the conservative uncertainty bounds. For example, if obstacle velocities are bounded, then one can consider a conservative space-time "cone" of possible obstacle positions that grows as time increases.

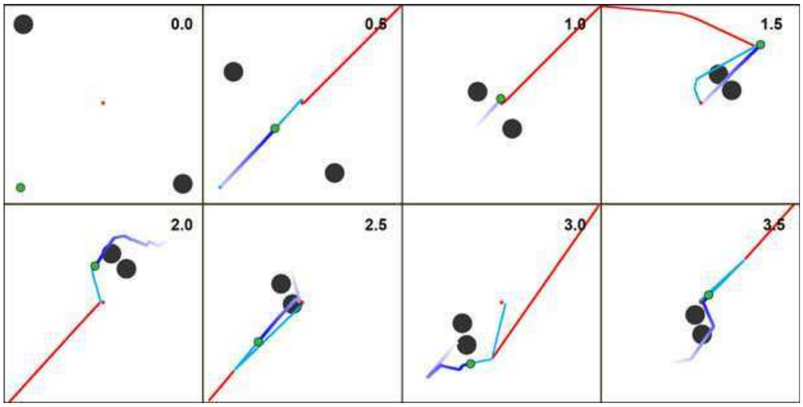


Fig. 7 Snapshots taken at half-second intervals from a pursuit-evasion experiment in the unit square. Two pursuers (grey circles) seek the evader (green) greedily at half the speed of the evader. The evader knows the pursuers' velocity bound but not their behavior. The evader replans a pessimistic path (red) to avoid the pursuers in the worst case, and replans an optimistic path (cyan) in order to reach the goal in the center of the room. Both paths share a common prefix. The trace of the robot between frames is drawn as a purple trail.

Consider a purely safety-seeking robot that uses the following scheme:

1. The current trajectory $y(t)$ has a *time to potential failure* (TTPF) T if it is safe for some duration T under conservative bounds on uncertainty. That is, $y(t) \in \tilde{F}(t; E_k)$ for all $t \in [t_k, t_k + T]$.
2. Replanning searches for a path $\hat{y}$ that increases the TTPF to $T + \Delta_k$ or some constant T_{min}, whichever is lower.

It is straightforward to use Algorithm 1 to implement such behavior simply by using the TTPF as an optimization criterion. The robot will remain safe unless replanning cannot improve the TTPF within the duration T (and even then, a constraint violation only happens in the worst case)[1]. A violation may occur if 1) no trajectory that improves the TTPF exists, in which case the planner can do nothing except hope that the potential hazard goes away, or 2) not enough planning time was devoted to finding a safe path.

The risk of condition (2) is somewhat mitigated by the selection of the parameter T_{min}, which governs an "acceptable" threshold for the TTPF. Below this threshold, the planner enforces that subsequent pessimistic plans must increase the TTPF. Naturally, if safety were the robot's only objective, the best approach is to set T_{min} to be infinite. In the below section, a finite T_{min} will allow it to make optimistic progress toward a target while being acceptably confident that safety will be ensured.

If the robot must also seek to optimize an objective $V(x)$, it must sacrifice some safety in order to do so. Below we describe a contingency planning framework where the robot's path has a similarly high probability of safety as the above scheme, but the planner seeks to simultaneously increase the TTPF and makes progress towards reducing $V(x)$.

4.2 A Contingency Replanning Algorithm

In our contingency planning algorithm, the robot maintains both an optimistic and a pessimistic trajectory that share a common prefix (Figure 7). The role of the pessimistic trajectory is to optimize the TTPF, while the role of the optimistic trajectory is to encourage consistent progress toward the goal.

Pseudocode is listed in Figure 8. The pessimistic trajectory $y(t)$ is maintained and followed by default. The optimistic trajectory $y^o(t)$, if it exists, is identical to $y(t)$ until the "junction" time t_j. Each iteration of the replanning loop begins by establishing time limits for the optimistic and the pessimistic planners, with sum Δ_k (Line 2). Then a top-level decision is made whether to initiate the new plan from the optimistic or the pessimistic trajectory:

[1] A major benefit of sample-based replanning is that holding TTPF constant, a factor n increase in computational speed results in a sharp reduction in failure rate from p to p^n.

```
Algorithm 2. Contingency Replanning with Adaptive Time Steps
Initialization:
0a. $y(t) \leftarrow$ an initial trajectory starting from $t = 0$.
0b. $y^o(t) \leftarrow$ nil.
0c. Junction time $t_j \leftarrow 0$

Repeat for $k = 1, \ldots$:
1. Measure the current time $t_k$.
2. Pick a pessimistic and optimistic time limit $\Delta_k^p$ and $\Delta_k^o$. Let $\Delta_k = \Delta_k^p + \Delta_k^o$
3. If $t_j > t_k + \Delta_k$ (branch the new plan from the optimistic path)
4.    Plan an improved optimistic path starting from $y_o(t_j)$.
5.    Plan a pessimistic path $\hat{y}$ starting from $y_o(t_j + \Delta_k)$.
6.    If Line 5 is successful, then
7.       Set $y(t) \leftarrow y_o(t)$ for $t \leq t_j + \Delta_k$, and $y(t) \leftarrow \hat{y}(t)$ for $t \geq t_j + \Delta_k$.
8.       Set $t_j \leftarrow t_j + \Delta_k$.
9.    End
10. Otherwise, (branch the new plan from the pessimistic path)
11.    Plan an optimistic path starting from $y(t_k + \Delta_k)$.
12.    If successful, then
13.       Plan a pessimistic path $\hat{y}$ starting from $y_o(t_k + 2\Delta_k)$.
14.       If successful, then
15.          Set $y(t) \leftarrow y_o(t)$ for $t_k + \Delta_k \leq t \leq t_k + 2\Delta_k$, and $y(t) \leftarrow \hat{y}(t)$ for
$t \geq t_k + 2\Delta_k$.
16.          Set $t_j \leftarrow t_k + 2\Delta_k$.
17.       End
18.    Otherwise,
19.       Plan a pessimistic path $\hat{y}$ starting from $y(t_k + \Delta_k)$.
20.       If successful, set $y(t) \leftarrow \hat{y}(t)$ for $t \geq t_k + \Delta_k$.
```

Fig. 8 Pseudocode for the contingency replanning algorithm.

- *From the optimistic trajectory* (Lines 4–9). To continue progress along y^o after time t_j, the robot must generate a pessimistic trajectory that branches out of y^o at some time after t_j. An improvement to the optimistic plan is attempted as well.
- *From the pessimistic trajectory* (Lines 11–20). To progress toward the target, the planner will attempt to branch a new pessimistic and optimistic pair out of the current pessimistic trajectory at time $t_k + \Delta_k$. The new junction time will be $t_k + 2\Delta_k$. If this fails, the planner attempts an extension to the pessimistic path.

To improve the optimistic path, the planner constructs a path in the optimistic feasible space $F(t; t_c)$ based on the current environment model. A query is deemed successful if, after time limit Δ_k^o, $C(y^o)$ is improved over the current optimistic path if it exists, or otherwise over the current pessimistic path. If the query fails, y^o is left untouched. Pessimistic queries are handled exactly as in the prior section.

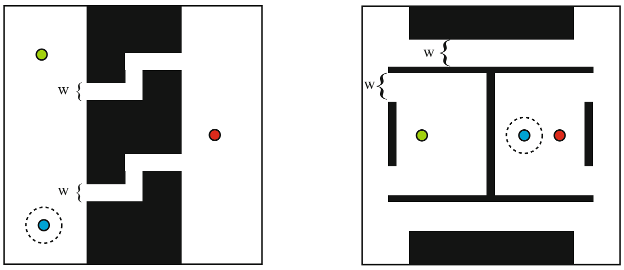


Fig. 9 Pursuit-evasion environments 1 and 2. Narrow passages, and hence, difficulty, are parameterized by w. The evader (green) must try to reach the target (red) within 10 s while avoiding the pursuer (blue), with capture radius 0.05.

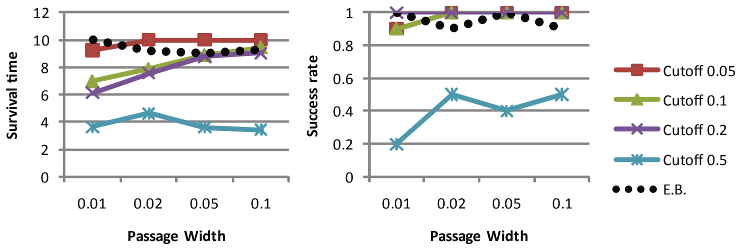


Fig. 10 (a) Survival time and (b) success rates for evader time-stepping strategies on problem 1. Results were averaged over ten trials on each passage width. The adaptive strategy (E.B.) performs as well as the best constant cutoff, and is more consistent across problem variations.

To choose planning times, we again use an adaptive time stepping scheme using the exponential backoff strategy of Section 3.2. Pessimistic and optimistic planning times are learned independently. We also make adjustments in case the candidate time step exceeds the finite TTPF of our paths. First, if we find that Δ_k exceeds the TTPF T of the pessimistic path, that is, failure may occur before planning is complete, we set $\Delta_k^p = T/2$ and $\Delta_k^o = 0$. Second, if we are attempting a modification to the optimistic trajectory, and the TTPF of the optimistic trajectory T^o is less than $t_j + \Delta_k$, then we scale Δ_k^p and Δ_k^o to attempt a replan before T^o (otherwise, the pessimistic replan is guaranteed to fail).

4.3 Experiments on a Pursuit-Evasion Example

Our experiments evaluate how contingency planning strategies affect an evader's performance in a planar pursuit-evasion scenario. The evader's goal is to reach a target within 10 s before being captured by a pursuer. The evader and pursuer move at maximum speeds 1 and 0.5, respectively. The evader treats the pursuer as an unpredictable obstacle with bounded

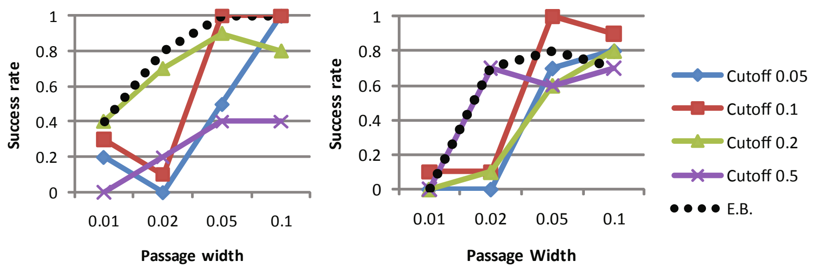


Fig. 11 Success rates for evader time-stepping strategies on problem 2 for a (a) nonadversarial and (b) adversarial pursuer behaviors. In the nonadversarial case the pursuer is allowed to pass through obstacles. A shorter time step (Cutoff 0.2) performs well in the nonadversarial case, but a longer time step (Cutoff 0.5) performs better in the adversarial case. The adaptive strategy works well in both cases.

velocity, and uses Algorithm 2 with $T_{min} = 1.0$. The evader's conservative model of $\tilde{F}(t, E_k)$ does not consider walls to be impediments to the pursuer's possible movement. The pursuer treats the evader as an unpredictably moving target, and uses Algorithm 1 to reach it.

Holding the pursuer's behavior constant, we varied the environment difficulty and evader's time stepping strategy on Problem 1 (Figure 9(a)). Here the pursuer begins in a room with the evader, which must escape through a narrow passage to reach the target in a second room. Figure 10 shows that narrow passage width does not affect the evader's survival much, but rather, responsiveness is more important to enable it to dance around an approaching pursuer. The adaptive time strategy appropriately finds short time steps.

Next, we considered a more difficult environment, Problem 2 (Figure 9(b)). Mere survival is not challenging (in all experiments it was over 90%), but reaching the target is; success requires the evader to choose a different hallway than the pursuer. We tested a nonadversarial pursuer behavior in which it "wanders" with velocity varying according to a random walk, and is allowed to pass through walls. Figure 11(a) shows that in this case, the success rate is highly dependent on problem difficulty, and no constant cutoff performs uniformly well across all width variations. Similar variations were found using an adversarial pursuer (Figure 11(b)). The adaptive strategy performed nearly as well as the best constant cutoff across all problem variations.

5 Conclusion

The runtime variance of planning queries has been a major impediment to the adoption of replanning techniques in real-time robot control. This paper addresses this problem by introducing two adaptive time-stepping algorithms – a simple one for deterministic environments, and a more complex one for nondeterministic environments – that tolerate run-time variance by learning a time step on-the-fly. Experiments on shared control for an industrial robot arm and on pursuit-evasion examples suggest that replanning may be a viable mechanism for real-time navigation and obstacle avoidance. Additional videos of our experiments can be found on the web at http://www.iu.edu/motion/realtime.html.

References

1. Allgöwer, F., Zheng, A.: Nonlinear Model Predictive Control (Progress in Systems and Control Theory). Birkhäuser, Basel (1956)
2. Anderson, S.J., Peters, S.C., Iagnemma, K.D., Pilutti, T.E.: A unified approach to semi-autonomous control of passenger vehicles in hazard avoidance scenarios. In: Proc. IEEE Int. Conf. on Systems, Man and Cybernetics, San Antonio, TX, USA, pp. 2032–2037 (2009)
3. Bekris, K., Kavraki, L.: Greedy but safe replanning under kinodynamic constraints. In: Proc. IEEE Int. Conference on Robotics and Automation (ICRA), Rome, Italy, pp. 704–710 (April 2007)
4. Bruce, J., Veloso, M.: Real-time randomized path planning for robot navigation. In: IEEE International Conference on Intelligent Robots and Systems (IROS), Lausanne, Switzerland (October 2002)
5. Feron, E., Frazzoli, E., Dahleh, M.: Real-time motion planning for agile autonomous vehicles. In: AIAA Conference on Guidance, Navigation and Control, Denver, USA (August 2000)
6. Hauser, K., Ng-Thow-Hing, V.: Fast smoothing of manipulator trajectories using optimal bounded-acceleration shortcuts. In: Proc. IEEE Int. Conference on Robotics and Automation (ICRA), Anchorage, USA (2010)
7. Hsu, D., Kindel, R., Latombe, J.-C., Rock, S.: Kinodynamic motion planning amidst moving obstacles. Int. J. Rob. Res. 21(3), 233–255 (2002)
8. Kallmann, M., Mataric, M.: Motion planning using dynamic roadmaps. In: IEEE Intl. Conf. on Robotics and Automation (ICRA) (April 2004)
9. LaValle, S., Kuffner, J.: Randomized kinodynamic planning. In: Proc. IEEE Intl. Conf. on Robotics and Automation, pp. 473–479 (1999)
10. Likhachev, M., Ferguson, D., Gordon, G., Stentz, A., Thrun, S.: Anytime dynamic a*: An anytime, replanning algorithm. In: Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS) (2005)
11. Mayne, D.Q., Rawlings, J.B., Rao, C.V., Scokaert, P.O.M.: Constrained model predictive control: Stability and optimality. Automatica 36, 789–814 (2000)
12. Musliner, D.J., Durfee, E.H., Shin, K.G.: Circa: A cooperative intelligent real-time control architecture. IEEE Transactions on Systems, Man, and Cybernetics 23, 1561–1574 (1993)
13. Petti, S., Fraichard, T.: Safe motion planning in dynamic environments. In: IEEE International Conference on Intelligent Robots and Systems (IROS), pp. 3726–3731 (2005)
14. Ross, I., Gong, Q., Fahroo, F., Kang, W.: Practical stabilization through real-time optimal control. In: American Control Conference, p. 6 (June 2006)
15. Sánchez, G., Latombe, J.-C.: On delaying collision checking in PRM planning: Application to multi-robot coordination. Int. J. of Rob. Res. 21(1), 5–26 (2002)

16. Stachniss, C., Burgard, W.: An integrated approach to goal-directed obstacle avoidance under dynamic constraints for dynamic environments. In: IEEE-RSJ Int. Conf. on Intelligent Robots and Systems (IROS), pp. 508–513 (2002)
17. Stentz, A.: The focussed d* algorithm for real-time replanning. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI) (1995)
18. van den Berg, J., Ferguson, D., Kuffner, J.: Anytime path planning and replanning in dynamic environments. In: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), pp. 2366–2371 (May 2006)
19. van den Berg, J., Overmars, M.: Roadmap-based motion planning in dynamic environments. IEEE Trans. Robot. 21(5), 885–897 (2005)
20. Zucker, M., Kuffner, J., Branicky, M.: Multipartite rrts for rapid replanning in dynamic environments. In: Proc. IEEE Int. Conf. Robotics and Automation (April 2007)

The Bayes Tree: An Algorithmic Foundation for Probabilistic Robot Mapping

Michael Kaess, Viorela Ila, Richard Roberts, and Frank Dellaert

Abstract. We present a novel data structure, the Bayes tree, that provides an algorithmic foundation enabling a better understanding of existing graphical model inference algorithms and their connection to sparse matrix factorization methods. Similar to a clique tree, a Bayes tree encodes a factored probability density, but unlike the clique tree it is directed and maps more naturally to the square root information matrix of the simultaneous localization and mapping (SLAM) problem. In this paper, we highlight three insights provided by our new data structure. First, the Bayes tree provides a better understanding of batch matrix factorization in terms of probability densities. Second, we show how the fairly abstract updates to a matrix factorization translate to a simple editing of the Bayes tree and its conditional densities. Third, we apply the Bayes tree to obtain a completely novel algorithm for sparse nonlinear incremental optimization, that combines incremental updates with fluid relinearization of a reduced set of variables for efficiency, combined with fast convergence to the exact solution. We also present a novel strategy for incremental variable reordering to retain sparsity. We evaluate our algorithm on standard datasets in both landmark and pose SLAM settings.

Keywords: graphical models, clique tree, probabilistic inference, sparse linear algebra, nonlinear optimization, smoothing and mapping, SLAM, iSAM.

1 Introduction

Probabilistic inference algorithms are important in robotics for a number of applications, ranging from simultaneous localization and mapping (SLAM) for building

Michael Kaess
CSAIL, MIT, Cambridge, Massachusetts
e-mail: kaess@mit.edu

Viorela Ila · Richard Roberts · Frank Dellaert
School of Interactive Computing, Georgia Institute of Technology, Atlanta, Georgia
e-mail: {vila,richard,frank}@cc.gatech.edu

geometric models of the world, to tracking people for human robot interaction. Our research is mainly in large-scale SLAM and hence we will use this as an example throughout the paper. SLAM is a core competency in mobile robotics, as it provides the necessary data for many other important tasks such as planning and manipulation, in addition to direct applications such as 3D modeling, exploration, and reconnaissance. The uncertainty inherent in sensor measurements makes probabilistic inference algorithms the favorite choice for SLAM. And because online operation is essential for most real applications, efficient incremental online algorithms are important and are at the focus of this paper.

Taking a graphical model perspective to probabilistic inference in SLAM has a rich history [2] and has especially led to several novel and exciting developments in the last years [27, 10, 13, 12, 11, 31]. Paskin proposed the thin junction tree filter (TJTF) [27], which provides an incremental solution directly based on graphical models. However, filtering is applied, which is known to be inconsistent when applied to the inherently nonlinear SLAM problem [20], i.e., the average taken over a large number of experiments diverges from the true solution. In contrast, *full SLAM* [34] retains all robot poses and can provide an exact solution, which does not suffer from inconsistency. Folkesson and Christensen presented Graphical SLAM [10], a graph-based full SLAM solution that includes mechanisms for reducing the complexity by locally reducing the number of variables. More closely related, Frese's Treemap [12] performs QR factorization within nodes of a tree that is balanced over time. Sparsification is applied to prevent nodes from becoming too large, introducing approximations by duplication of variables.

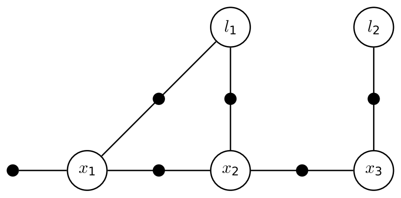
The sparse linear algebra perspective has been explored by Dellaert et al. [6, 7, 23] in Smoothing and Mapping (SAM), an approach that exploits the sparsity of the smoothing information matrix. The matrices associated with smoothing are typically very sparse, and one can do much better than the cubic complexity associated with factorizing a dense matrix [24]. Kaess et al. [22, 23] proposed incremental smoothing and mapping (iSAM), which performs *fast incremental updates* of the square root information matrix, yet is able to compute the full map and trajectory at any time. New measurements are added using matrix update equations [16, 15, 17], so that previously calculated components of the square root information matrix are reused. However, to remain efficient and consistent, iSAM requires periodic batch steps to allow for variable reordering and relinearization, which is expensive and detracts from the intended online nature of the algorithm.

To combine the advantages of the graphical model and sparse linear algebra perspective, **we propose a novel data structure, the Bayes tree.** Our approach is based on viewing matrix factorization as eliminating a factor graph into a Bayes net, which is the graphical model equivalent of the *square root information matrix*. Performing marginalization and optimization in Bayes nets is not easy in general. However, a Bayes net resulting from elimination/factorization is *chordal,* and it is well known that a chordal Bayes net can be converted into a tree-structured graphical model in which these operations are easy. The most well-known such data structure is the *clique tree* [30, 1], also known as the *junction tree* in the AI literature [4], which has already been exploited for distributed inference in SLAM [8, 27]. However, the new data structure we propose here, the Bayes tree, is *directed* and corresponds more naturally to the result of the QR factorization in linear algebra, allowing us to analyze it in terms of conditional probability densities in the tree. We further show that incremental inference corresponds to a simple editing of this tree, and present a novel incremental variable ordering strategy.

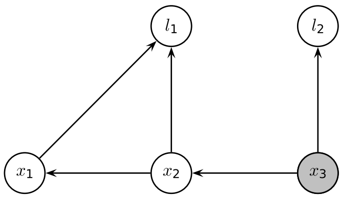
Exploiting this new data structure and the insights gained, we propose **a novel incremental exact inference method that allows for incremental reordering and just-in-time relinearization**. To the best of our knowledge this is a completely novel approach to providing an efficient and exact solution to a sparse nonlinear optimization problem in an incremental setting, with general applications beyond SLAM. While standard nonlinear optimization methods repeatedly solve a linear batch problem to update the linearization point, our Bayes tree-based algorithm allows fluid relinearization of a reduced set of variables which translates into higher efficiency, while retaining sparseness and full accuracy. We compare our new method to iSAM using multiple publicly available datasets in both landmark and pose SLAM settings.

2 Problem Statement

We use a *factor graph* [25] to represent the SLAM problem in terms of graphical models. Formally, a factor graph is a bipartite graph $G = (\mathscr{F}, \Theta, \mathscr{E})$ with two node types: *factor nodes* $f_i \in \mathscr{F}$ and *variable nodes* $\theta_j \in \Theta$. Edges $e_{ij} \in \mathscr{E}$ are always



$$A = \begin{array}{c} \begin{array}{ccccc} l_1 & l_2 & x_1 & x_2 & x_3 \end{array} \\ \begin{bmatrix} X & & X & & \\ X & & & X & \\ & X & & & X \\ & & X & & \\ & & X & X & \\ & & & X & X \end{bmatrix} \end{array}$$



$$R = \begin{array}{c} \begin{array}{ccccc} l_1 & l_2 & x_1 & x_2 & x_3 \end{array} \\ \begin{bmatrix} X & & X & X & \\ & X & & & X \\ & & X & X & \\ & & & X & X \\ & & & & X \end{bmatrix} \end{array}$$

Fig. 1 (top) The factor graph and the associated Jacobian matrix A for a small SLAM example, where a robot located at successive poses x_1, x_2, and x_3 makes observations on landmarks l_1 and l_2. In addition there is an absolute measurement on the pose x_1. (bottom) The chordal Bayes net and the associated square root information matrix R resulting from eliminating the factor graph using the elimination ordering l_1, l_2, x_1, x_2, x_3. Note that the root, the last variable to be eliminated, is shaded darker.

Algorithm 1. Eliminating a variable θ_j from the factor graph.

```
1. Remove from the factor graph all factors $f_i(\Theta_i)$ that are adjacent to $\theta_j$. Define the *separator* $S_j$ as all variables involved in those factors, excluding $\theta_j$.
2. Form the (unnormalized) joint density $f_{joint}(\theta_j, S_j) = \prod_i f_i(\Theta_i)$ as the product of those factors.
3. Using the chain rule, factorize the joint density $f_{joint}(\theta_j, S_j) = P(\theta_j|S_j) f_{new}(S_j)$. Add the conditional $P(\theta_j|S_j)$ to the Bayes net and the factor $f_{new}(S_j)$ back into the factor graph.
```

between factor nodes and variables nodes. A factor graph G defines the factorization of a function $f(\Theta)$ as

$$f(\Theta) = \prod_i f_i(\Theta_i) \tag{1}$$

where Θ_i is the set of variables θ_j adjacent to the factor f_i, and independence relationships are encoded by the edges e_{ij}: each factor f_i is a function of the variables in Θ_i. An example of a SLAM factor graph is shown in Fig. 1(top).

When assuming Gaussian measurement models

$$f_i(\Theta_i) \propto \exp\left(-\frac{1}{2}\left\|h_i(\Theta_i) - z_i\right\|^2_{\Sigma_i}\right) \tag{2}$$

as is standard in the SLAM literature [32, 3, 9], the factored objective function we want to maximize (1) corresponds to the nonlinear least-squares criterion

$$\arg\min_{\Theta}\left(-\log f(\Theta)\right) = \arg\min_{\Theta} \frac{1}{2}\sum_i \left\|h_i(\Theta_i) - z_i\right\|^2_{\Sigma_i} \tag{3}$$

where $h_i(\Theta_i)$ is a measurement function and z_i a measurement, and $\|e\|^2_\Sigma \overset{\Delta}{=} e^T \Sigma^{-1} e$ is defined as the squared Mahalanobis distance with covariance matrix Σ.

A crucial insight is that **inference can be understood as converting the factor graph to a Bayes net using the elimination algorithm.** Variable elimination [1, 4] originated in order to solve systems of linear equations, and was first applied in modern times by Gauss in the early 1800s [14].

In factor graphs, elimination is done via a *bipartite elimination game*, as described by Heggernes and Matstoms [19]. This can be understood as taking apart the factor graph and transforming it into a *Bayes net* [29]. One proceeds by eliminating one variable at a time, and converting it into a node of the Bayes net, which is gradually built up. After eliminating each variable, the reduced factor graph defines a density on the remaining variables. The pseudo-code for eliminating a variable θ_j is given in Algorithm 1. After eliminating all variables, the Bayes net density is defined by the product of the conditionals produced at each step:

$$P(\Theta) = \prod_j P(\theta_j|S_j) \tag{4}$$

The result of this process for the example in Fig. 1(top) is shown in Fig. 1(bottom).

3 The Bayes Tree

The Bayes net resulting from elimination/factorization is *chordal,* and it can be converted into a tree-structured graphical model in which optimization and marginalization are easy. In this paper we introduce a new data structure, *the Bayes tree*, to better capture the equivalence with linear algebra and enable new algorithms in recursive estimation. A Bayes tree is a directed tree where the nodes represent *cliques* C_k of the underlying chordal Bayes net. In this respect Bayes trees are similar to clique trees, but a Bayes tree is directed and is closer to a Bayes net in the way it encodes a factored probability density. In particular, we define one conditional density $P(F_k|S_k)$ per node, with the *separator* S_k as the intersection $C_k \cap \Pi_k$ of the clique C_k and its parent clique Π_k, and the *frontal variables* F_k as the remaining variables, i.e. $F_k \overset{\Delta}{=} C_k \setminus S_k$. We write $C_k = F_k : S_k$. This leads to the following expression for the joint density $P(\Theta)$ on the variables Θ defined by a Bayes tree,

$$P(\Theta) = \prod_k P(F_k|S_k) \tag{5}$$

where for the root F_r the separator is empty, i.e., it is a simple prior $P(F_r)$ on the root variables. The way Bayes trees are defined, the separator S_k for a clique C_k is always a subset of the parent clique Π_k, and hence the directed edges in the graph have the same semantic meaning as in a Bayes net: conditioning.

Every chordal Bayes net can be transformed into a tree by discovering its cliques. Discovering cliques in chordal graphs is done using the maximum cardinality search algorithm by Tarjan and Yannakakis [33], which proceeds in reverse elimination order to discover cliques in the Bayes net. The algorithm for converting a Bayes net into a Bayes tree is summarized in Algorithm 2 and the corresponding Bayes tree for the small SLAM example in Fig. 1 is shown in Fig. 2.

Gaussian Case. In practice one always considers a linearized version of problem (3). If the measurement models h_i in equation (2) are nonlinear and a good linearization point is not available, nonlinear optimization methods such as Gauss-Newton iterations or the Levenberg-Marquardt algorithm solve a succession of linear approximations to (3) in order to approach the minimum.

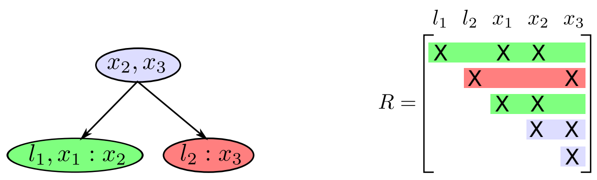


Fig. 2 The Bayes tree and the associated square root information matrix R describing the clique structure in the Bayes net from Fig. 1. A Bayes tree is similar to a clique tree, but is better at capturing the formal equivalence between sparse linear algebra and inference in graphical models. The association of cliques with rows in the R factor is indicated by color.

Algorithm 2. Creating a Bayes tree from the chordal Bayes net resulting from elimination (Algorithm 1).

```
For each conditional density $P(\theta_j|S_j)$ of the Bayes net, in *reverse* elimination order:
If no parent ($S_j = \{\}$)
    start a new root clique $F_r$ containing $\theta_j$
else
    identify parent clique $C_p$ that contains the first eliminated variable of $S_j$ as a frontal variable
    if nodes $F_p \cup S_p$ of parent clique $C_p$ are equal to separator nodes $S_j$ of conditional
        insert conditional into clique $C_p$
    else
        start new clique $C'$ as child of $C_p$ containing $\theta_j$
```

At each iteration of the nonlinear solver, we linearize around a linearization point Θ to get a new, *linear* least-squares problem in $\mathbf{x}$ with the objective function

$$-\log f(\mathbf{x}) = \frac{1}{2}\|\mathbf{A}\mathbf{x} - \mathbf{b}\|^2 \tag{6}$$

where $\mathbf{A} \in \mathbb{R}^{m \times n}$ is the measurement Jacobian consisting of m measurement rows and $\mathbf{x}$ is an n-dimensional tangent vector of a minimal representation [18]. Note that the covariances Σ_i have been absorbed into the corresponding block rows of $\mathbf{A}$, making use of $\|\mathbf{x}\|_\Sigma^2 = \mathbf{x}^T\Sigma^{-1}\mathbf{x} = \mathbf{x}^T\Sigma^{-\frac{T}{2}}\Sigma^{-\frac{1}{2}}\mathbf{x} = \left\|\Sigma^{-\frac{1}{2}}\mathbf{x}\right\|^2$. The matrix $\mathbf{A}$ above is a sparse block-matrix, and its graphical model counterpart is a *Gaussian* factor graph with exactly the same structure as the nonlinear factor graph, see Fig. 1. The probability density on $\mathbf{x}$ defined by this factor graph is the normal distribution

$$P(\mathbf{x}) \propto e^{-\log f(\mathbf{x})} = \exp\left\{-\frac{1}{2}\|\mathbf{A}\mathbf{x} - \mathbf{b}\|^2\right\} \tag{7}$$

In Gaussian factor graphs, elimination is equivalent to sparse QR factorization of the measurement Jacobian. In Gaussian factor graphs, the chain rule $f_{joint}(\theta_j, S_j) = P(\theta_j|S_j)f_{new}(S_j)$ in step 3 of Algorithm 1 can be implemented using Householder reflections or a Gram-Schmidt orthogonalization, in which case the entire elimination algorithm is equivalent to QR factorization of the entire measurement matrix $\mathbf{A}$. To see this, note that, for $x_j \in \mathbb{R}$ and $\mathbf{s}_j \in \mathbb{R}^l$ (the set of variables S_j combined in a vector of length l), the factor $f_{joint}(x_j, \mathbf{s}_j)$ defines a Gaussian density

$$f_{joint}(x_j, \mathbf{s}_j) \propto \exp\left\{-\frac{1}{2}\left\|\mathbf{a}x_j + \mathbf{A}_S\mathbf{s}_j - \mathbf{b}\right\|^2\right\} \tag{8}$$

where the dense, but small matrix $\mathbf{A}_j = [\mathbf{a}|\mathbf{A}_S]$ is obtained by concatenating the vectors of partial derivatives of all factors connected to variable x_j. Note that $\mathbf{a} \in \mathbb{R}^k$, $\mathbf{A}_S \in \mathbb{R}^{k \times l}$ and $\mathbf{b} \in \mathbb{R}^k$, with k the number of measurement rows of all factors connected to x_j. The desired conditional $P(x_j|\mathbf{s}_j)$ is obtained by evaluating the joint (8) for a given value of $\mathbf{s}_j$, yielding

$$P(x_j|\mathbf{s}_j) \propto \exp\left\{-\frac{1}{2}(x_j + \mathbf{r}\mathbf{s}_j - d)^2\right\} \tag{9}$$

with $\mathbf{r} \overset{\Delta}{=} \mathbf{a}^\dagger \mathbf{A}_S$ and $d \overset{\Delta}{=} \mathbf{a}^\dagger \mathbf{b}$, where $\mathbf{a}^\dagger \overset{\Delta}{=} \left(\mathbf{a}^T\mathbf{a}\right)^{-1}\mathbf{a}^T$ is the *pseudo-inverse* of $\mathbf{a}$. The new factor $f_{new}(\mathbf{s}_j)$ is obtained by substituting $x_j = d - \mathbf{r}\mathbf{s}_j$ back into (8):

$$f_{new}(\mathbf{s}_j) = \exp\left\{-\frac{1}{2}\left\|\mathbf{A}'\mathbf{s}_j - \mathbf{b}'\right\|^2\right\} \tag{10}$$

where $\mathbf{A}' \overset{\Delta}{=} \mathbf{A}_S - \mathbf{a}\mathbf{r}$ and $\mathbf{b}' \overset{\Delta}{=} \mathbf{b} - \mathbf{a}d$. The above is one step of Gram-Schmidt, interpreted in terms of densities, and the sparse vector $\mathbf{r}$ and scalar d can be recognized as specifying a single joint conditional density in the Bayes net, or alternatively a single row in the sparse square root information matrix as indicated in Fig. 2.

Solving. The optimal assignment $\mathbf{x}^*$ of the linear least-squares solution is the one that maximizes the joint density $P(\mathbf{x})$ from (7). The optimal assignment $\mathbf{x}^*$ can be computed in dynamic programming style in one pass from the leaves up to the root of the tree to define all functions, and then one pass down to retrieve the optimal assignment for all frontal variables, which together make up the variables $\mathbf{x}$. The first pass is already performed during construction of the Bayes tree, and is represented by the conditional densities associated with each clique. The second pass recovers the optimal assignment starting from the root based on (9) by solving

$$x_j = d - \mathbf{r}\mathbf{s}_j \tag{11}$$

for every variable x_j, which is equivalent to backsubstitution in sparse linear algebra.

4 Incremental Inference

We show that incremental inference corresponds to a simple editing of the Bayes tree, which also provides a better explanation and understanding of the otherwise abstract incremental matrix factorization process. In particular, we will now store and compute the square root information matrix R in the form of a Bayes tree $\mathscr{T}$. Incremental factorization/inference is performed by reinterpreting the top part of the Bayes tree again as a factor graph, adding to this the new factors, creating with a new elimination order a new Bayes tree from this “top”, then reattaching to it the unaffected subtrees. When a new measurement is added, for example a factor $f'(x_j, x_{j'})$, only the paths between the cliques containing x_j and $x_{j'}$ (respectively) and the root are affected. The sub-trees below these cliques are unaffected, as are any other sub-trees not containing x_j or $x_{j'}$. Fig. 3 shows how these steps are applied to our small SLAM example (originally in Fig. 2). The upper-left shows that adding

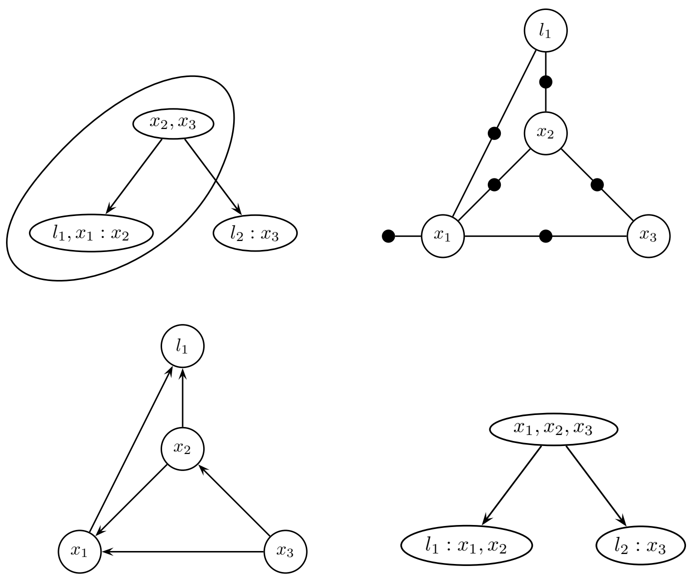


Fig. 3 Updating a Bayes tree with a new factor, based on the example in Fig. 2. (top left) The affected part of the Bayes tree is highlighted for the case of adding a new factor between x_1 and x_3. Note that the right branch is not affected by the change. (top right) The factor graph generated from the affected part of the Bayes tree. (bottom left) The chordal Bayes net resulting from eliminating the factor graph. (bottom right) The Bayes tree created from the chordal Bayes net, with the unmodified right "orphan" subtree from the original Bayes tree added back in.

Algorithm 3. Updating the Bayes tree with new factors $\mathscr{F}'$.

```
In: Bayes tree $\mathscr{T}$, new linear factors $\mathscr{F}'$
Out: modified Bayes tree $\mathscr{T}$'

1. Remove top of Bayes tree and re-interpret it as a factor graph:
    a. For each affected variable, remove the corresponding clique and all parents up to the root.
    b. Store orphaned sub-trees $\mathscr{T}_{orph}$ of removed cliques.
2. Add the new factors $\mathscr{F}'$ into the resulting factor graph.
3. Re-order and eliminate the factor graph into a Bayes net (Algorithm 1), and re-assemble into a new Bayes tree (Algorithm 2).
4. Insert the orphans $\mathscr{T}_{orph}$ back into the new Bayes tree.
```

the new factor between x_1 and x_3 only affects the left branch of the tree. The entire process of updating the Bayes tree with a new factor is described in Algorithm 3.

To understand why only the top part of the tree is affected, we look at two important properties of the Bayes tree. These directly arise from it encoding information flow during elimination. First, during elimination, variables in each clique collect information *from their child cliques* via the elimination of these children. Thus, information in any clique propagates *only upwards* to the root. Second, the information from a factor enters elimination only when the first variable of that factor is eliminated.

Combining these two properties, we see that a new factor cannot influence any other variables that are not successors of the factor's variables. However, a factor on variables having different (i.e. independent) paths to the root means that these paths must now be re-eliminated to express the new dependency between them.

5 Incremental Reordering

Choosing the right variable ordering is essential for the efficiency of a sparse matrix solution, and this also holds for the Bayes tree approach. An optimal ordering minimizes the fill-in, which refers to additional entries in the square root information matrix that are created during the elimination process. In the Bayes tree, fill-in translates to larger clique sizes, and consequently slower computations. Fill-in can usually not be completely avoided, unless the original Bayes net already is chordal. Finding the variable ordering that leads to the minimal fill-in is NP-hard. One typically uses heuristics such as the column approximate minimum degree (COLAMD) algorithm by Davis et al. [5], which provide close to optimal orderings for many problems.

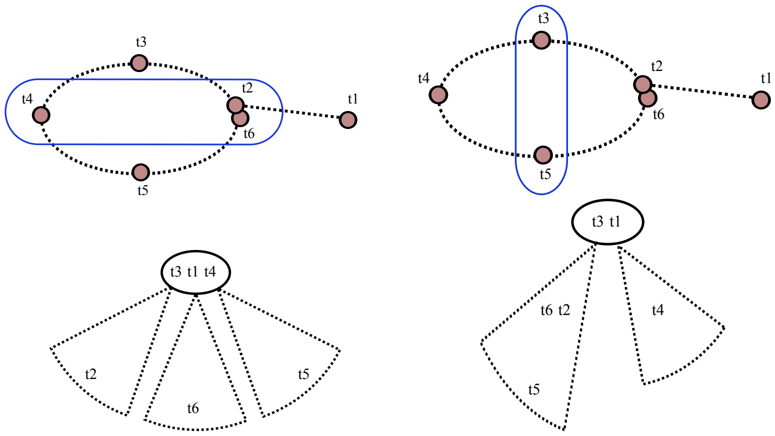


Fig. 4 For a trajectory with loop closing, two different optimal variable orderings based on nested dissection are shown in the top row, with the corresponding Bayes tree structure in the bottom row. For the incremental setting the left choice is preferable, as the most recent variables end up in the root, minimizing work in future updates.

While performing incremental inference in the Bayes tree, variables can be reordered at every incremental update, eliminating the need for periodic batch reordering. This was not understood in [23], because this is only obvious within the graphical model framework, but not for matrices. Reordering is only performed for the variables affected by the new factors. Finding an optimal ordering for this subset of variables does not necessarily provide an optimal overall ordering. However, we have observed that some incremental orderings provide good solutions, comparable to batch application of COLAMD.

One particularly good ordering forces the affected variables to be eliminated last. This strategy provides a good ordering because new measurements almost always connect to recently observed variables. In particular, odometry measurements always connect to the previous pose. In the exploration mode it is clear that if the most recent variables end up in the root, only a small part of the tree (optimally only the root) has to be reorganized in the next step. The more difficult case of a loop closure is visualized in Fig. 4. In the case of a simple loop, nested dissection provides the optimal ordering. The first cut can either (a) include the root, or (b) not include the root, and both solutions are equivalent in terms of fill-in. However, there is a significant difference in the incremental case: For the horizontal cut that does not include the most recent variable t_6, that variable will end up further down in the tree, requiring larger parts of the tree to change in the next update step. The vertical cut, on the other hand, includes the last variable in the first cut, pushing it into the root, and therefore leading to smaller, more efficient changes in the next step. In order to deal with more general topologies than this simple example, we use a constrained version of the COLAMD algorithm, that allows keeping the last variables in the root while still obtaining a good overall ordering.

6 Exact Incremental Inference with Fluid Relinearization

In this section we use the Bayes tree in a novel algorithm for optimizing a set of nonlinear factors that grows over time, which is directly applicable to online mapping. We have already shown how the Bayes tree is updated with new linear factors. We now discuss how to perform relinearization where needed, a process that we call *fluid relinearization*. Then we present a combined algorithm for adding nonlinear factors over time, while keeping the Bayes tree and the estimate up-to-date.

The goal of our algorithm is to obtain an estimate Θ for the variables (map and trajectory), given a set of nonlinear constraints that expands over time, represented by nonlinear factors $\mathscr{F}$. New factors $\mathscr{F}'$ can arrive at any time and may add new variables Θ' to the estimation problem. We take the most recent estimate Θ as linearization point to solve a linearized system as a subroutine in an iterative nonlinear optimization scheme. The linearized system is represented by the Bayes tree $\mathscr{T}$.

Solving. When solving in a nonlinear setting, we obtain a delta vector Δ that is used to update the linearization point Θ, as shown in Algorithm 4. Updates are often local operations that do not affect the solution of other parts of the map. Therefore we will consider variables unchanged for which the recovered delta changes by less

Algorithm 4. Solving the Bayes tree in the nonlinear case returns an update Δ that can be added to the current linearization point Θ to obtain the current estimate for all variables $\Theta + \Delta$.

```
In: Bayes tree $\mathscr{T}$
Out: update $\Delta$
Starting from the root clique $C_r = F_r$:

1. For current clique $C_k = F_k : S_k$
   compute update $\Delta_k$ of frontal variables $F_k$ using already computed values of parents $S_k$
   and the local conditional density $P(F_k|S_k)$.
2. For all variables $\Delta_{k,j}$ in $\Delta_k$ that change by more than a threshold $\alpha$:
   recursively process each descendant containing such a variable.
```

Algorithm 5. Fluid relinearization: The linearization points of select variables are updated based on the current delta Δ.

```
In: nonlinear factors $\mathscr{F}$, linearization point $\Theta$, Bayes tree $\mathscr{T}$, delta $\Delta$
Out: updated Bayes tree $\mathscr{T}$, updated linearization point $\Theta$

1. Mark variables in $\Delta$ above threshold $\beta$: $J = \{\Delta_j \in \Delta | \Delta_j \geq \beta\}$.
2. Update linearization point for marked variables: $\Theta_J := \Theta_J + \Delta_J$.
3. Mark all cliques that involve marked variables $\Theta_J$ and all their ancestors.
4. From the leaves to the top, if a clique is marked:
   a. Relinearize the original factors in $\mathscr{F}$ associated with the clique.
   b. Add cached marginal factors from any unmarked children.
   c. Re-eliminate.
```

than a small threshold α. For a clique that does not contain any variables that are considered changed, the subtrees will not be traversed. To be exact, the different units of variables have to be taken into account, but one simple solution is to take the minimum over all thresholds.

Fluid Relinearization. The idea behind just-in-time or fluid relinearization is to keep track of the validity of the linearization point for each variable, and only relinearize when needed. This represents a departure from the conventional linearize/solve approach that currently represents the state of the art, and can be viewed as a completely new algorithm for nonlinear optimization. For a variable that is chosen to be relinearized, all relevant information has to be removed from the Bayes tree and replaced by relinearizing the corresponding original nonlinear factors. Cliques that are re-eliminated have to take into account also the marginal factors that get passed up from subtrees. We cache those marginals during elimination to avoid having to re-eliminate unmarked cliques to obtain them. Algorithm 5 shows the overall fluid relinearization process.

Now we have all components for a fully incremental nonlinear optimization algorithm that allows exact incremental inference for sparse nonlinear problems such as SLAM. The algorithm is summarized in Algorithm 6, and we provide a brief

Algorithm 6. Nonlinear iteration with incremental variable reordering and fluid relinearization.

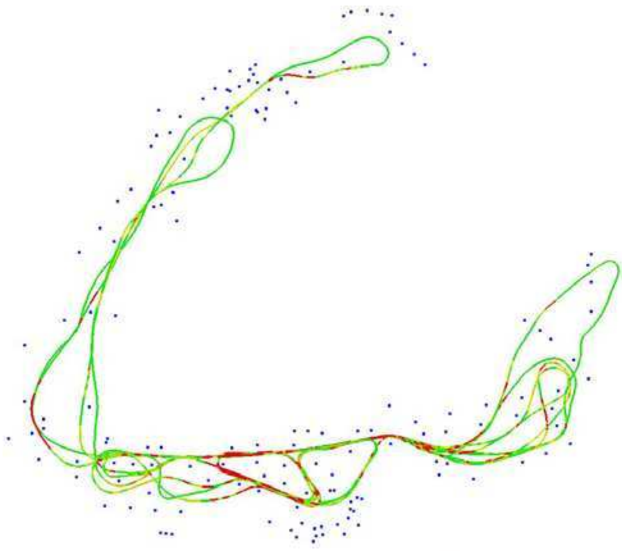
```
In / out: Bayes tree $\mathscr{T}$, linearization point $\Theta$, nonlinear factors $\mathscr{F}$
In: new nonlinear factors $\mathscr{F}'$, new variables $\Theta'$
Initialization: $\mathscr{T} = \emptyset$, $\Theta = \emptyset$, $\mathscr{F} = \emptyset$

1. Add any new factors $\mathscr{F} := \mathscr{F} \cup \mathscr{F}'$.
2. Initialize any new variables $\Theta'$ and add $\Theta := \Theta \cup \Theta'$.
3. Linearize new factors $\mathscr{F}'$ to obtain $\mathscr{F}'_{lin}$.
4. Linear update step, applying Algorithm 3 to $\mathscr{F}'_{lin}$.
5. Solve for delta $\Delta$ with Algorithm 4.
6. Iterate Algorithm 5 until no more relinearizations occur.
```

discussion of its complexity here. We assume here that initialization is available and it is close enough to the global minimum to allow convergence - that is a general requirement of any direct solver method. The number of iterations needed to converge is typically fairly small, in particular because of the quadratic convergence properties of our algorithm near the minimum. For exploration tasks with a constant number of constraints per pose, the complexity is $O(1)$. In the case of loop closures the situation becomes more difficult, and the most general bound is that for full factorization, $O(n^3)$, where n is the number of variables (poses and landmarks if present). Under certain assumptions that hold for many SLAM problems, the complexity is bounded by $O(n^{1.5})$ [24]. It is important to note that this bound does not depend on the number of loop closings. It should also be noted that our incremental algorithm is often much faster than a full factorization, as we show below.

7 Experimental Results

This section describes the experiments that validate the presented approach, using both synthetic and real datasets that are publicly available. We compare our estimation and timing results with a state of the art incremental algorithm [23] in order to highlight the advantages of fluid relinearization and incremental reordering. We have implemented the batch and iSAM algorithms using the same Bayes tree library to provide a comparison of the algorithms, rather than a comparison of different implementations. All results are obtained with a research C++ implementation, running single-threaded on a laptop with Intel Core 2 Duo 2.2 GHz processor, and using the COLAMD algorithm by Davis et al. [5]. We use the thresholds $\alpha = 0.005$ and $\beta = 0.05$ for solving and relinearization, respectively.

We evaluate the timing of our Bayes tree algorithm on the Victoria Park dataset, an often-used benchmark SLAM dataset [23, 28] courtesy of H. Durrant-Whyte and E. Nebot. This dataset includes 6969 laser scans with the corresponding odometry readings. The laser scans are processed to detect the trunks of the trees in the park, which are used as landmarks. Fig. 5 shows the final trajectory estimate together with the detected landmarks. In this figure the trajectory is colored according to



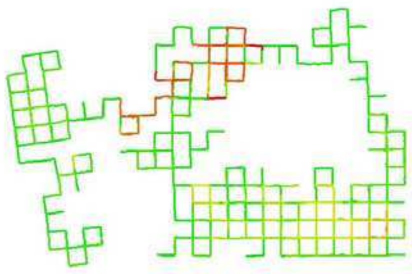
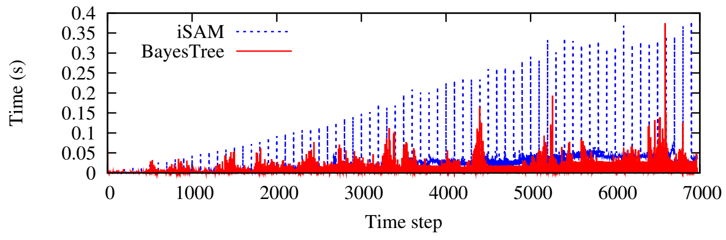


Fig. 5 The Victoria Park dataset (top) and the simulated Manhattan world dataset (bottom) after optimization, color coded with the number of variables that are updated for every step along the trajectory. Green corresponds to a low number of variables, red to a high number.

the number of variables our algorithm had to recalculate at each step, where green represents a small number of variables (order of 10), yellow a moderate number, and red finally a large number (order of hundreds of variables). A relatively small portion of the trajectory is colored red, mainly the part at the bottom where the vehicle closed loops multiple times, re-visiting the same location up to eight times. In Fig. 6, we compare per-step timing on the Victoria Park dataset between our algorithm and the original iSAM algorithm [23] (both implemented using the same library as noted above). The results show that our fully incremental algorithm does not suffer from the periodic spikes in iSAM. Our algorithm also performs better in cumulative time, while providing the additional advantage of continuously updating the linearization point of all variables having significant changes.

To evaluate the accuracy of our algorithm, we use the simulated Manhattan world from [26], courtesy of E. Olson. Fig. 7 shows that the normalized chi-square values follow the least-squares batch solution, providing a nearly exact solution in every



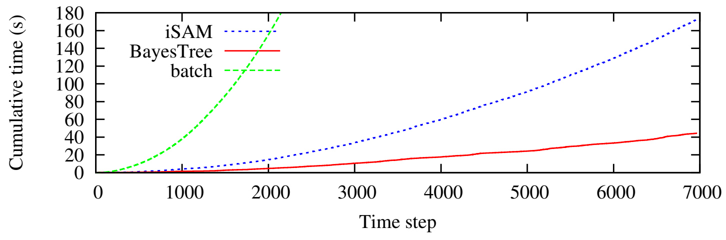


Fig. 6 Timing comparison for the Victoria Park dataset. The top row shows per step timing and the bottom row shows the cumulative time. Our new algorithm (red) provides an improvement in speed over the original iSAM algorithm (blue), in addition to its advantages of eliminating periodic batch factorization and performing fluid relinearization. The original iSAM algorithm included a batch step every 100 iterations, which is clearly visible from the spikes.

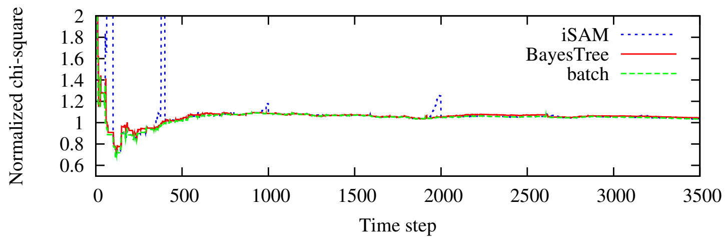


Fig. 7 Comparison of normalized χ^2 for the simulated Manhattan world. iSAM shows some spikes where it deviates from the least squares solution because relinearization is only performed every 100 steps. The Bayes tree solution is always very close to the least squares solution because of the fluid relinearization ($\beta = 0.05$).

step. While iSAM also converges to the exact solution, it shows some spikes related to relinearization only being performed in the periodic batch steps. Final cumulative times for providing a full solution in every step are $19.4s$ and $47.6s$ for our algorithm and iSAM, respectively. Fig. 5 shows the estimated trajectory for the simulated Manhattan world, again using the same color coding for the number of variables that had to be recalculated in each step.

Finally, we evaluated timing results on the Intel dataset, courtesy of D. Haehnel and D. Fox. This dataset was preprocessed by laser scan-matching, resulting in a pose graph formulation without landmarks, containing about 4000 poses. The final cumulative times are 44.4*s* and 172.6*s* for our algorithm and iSAM, respectively.

8 Conclusion

We have presented a novel data structure, the Bayes tree, that provides an algorithmic foundation which enables new insights into existing graphical model inference algorithms and sparse matrix factorization methods. These insights have led us to a fully incremental algorithm for nonlinear least-squares problems as they occur in mobile robotics. We have used SLAM as an example application, even though the algorithm is also suitable for other incremental inference problems, such as object tracking and sensor fusion. Our novel graph-based algorithm should also allow for better insights into the recovery of marginal covariances, as we believe that simple recursive algorithms in terms of the Bayes tree are formally equivalent to the dynamic programming methods described in [21]. The graph based structure also provides a starting point for exploiting parallelization that is becoming available in newer processors.

Acknowledgements. This work was partially funded under NSF grant 0713162 (RI: Inference in Large-Scale Graphical Models) and under ONR grant N00014-06-1-0043. V. Ila has been partially supported by the Spanish MICINN under the *Programa Nacional de Movilidad de Recursos Humanos de Investigación*. M. Kaess would like to thank J. Leonard for his support of this work.

References

1. Blair, J., Peyton, B.: An introduction to chordal graphs and clique trees. In: George, J., Gilbert, J., Liu, J.H. (eds.) Graph Theory and Sparse Matrix Computations, IMA Volumes in Mathematics and its Applications, vol. 56, pp. 1–27. Springer, Heidelberg (1993)
2. Brooks, R.: Visual map making for a mobile robot. In: IEEE Intl. Conf. on Robotics and Automation (ICRA), vol. 2, pp. 824–829 (1985)
3. Castellanos, J., Montiel, J., Neira, J., Tardós, J.: The SPmap: A probabilistic framework for simultaneous localization and map building. IEEE Trans. Robot. Automat. 15(5), 948–953 (1999)
4. Cowell, R.G., Dawid, A.P., Lauritzen, S.L., Spiegelhalter, D.J.: Probabilistic Networks and Expert Systems. In: Statistics for Engineering and Information Science. Springer, Heidelberg (1999)
5. Davis, T., Gilbert, J., Larimore, S., Ng, E.: A column approximate minimum degree ordering algorithm. ACM Trans. Math. Softw. 30(3), 353–376 (2004)
6. Dellaert, F.: Square Root SAM: Simultaneous location and mapping via square root information smoothing. In: Robotics: Science and Systems, RSS (2005)
7. Dellaert, F., Kaess, M.: Square Root SAM: Simultaneous localization and mapping via square root information smoothing. Intl. J. of Robotics Research 25(12), 1181–1203 (2006)

8. Dellaert, F., Kipp, A., Krauthausen, P.: A multifrontal QR factorization approach to distributed inference applied to multi-robot localization and mapping. In: Proc. 22nd AAAI National Conference on AI, Pittsburgh, PA (2005)
9. Dissanayake, M., Newman, P., Durrant-Whyte, H., Clark, S., Csorba, M.: A solution to the simultaneous localization and map building (SLAM) problem. IEEE Trans. Robot. Automat. 17(3), 229–241 (2001)
10. Folkesson, J., Christensen, H.: Graphical SLAM - a self-correcting map. In: IEEE Intl. Conf. on Robotics and Automation (ICRA), vol. 1, pp. 383–390 (2004)
11. Folkesson, J., Christensen, H.: Closing the loop with Graphical SLAM. IEEE Trans. Robotics 23(4), 731–741 (2007)
12. Frese, U.: Treemap: An $O(\log n)$ algorithm for indoor simultaneous localization and mapping. Autonomous Robots 21(2), 103–122 (2006)
13. Frese, U., Larsson, P., Duckett, T.: A multilevel relaxation algorithm for simultaneous localisation and mapping. IEEE Trans. Robotics 21(2), 196–207 (2005)
14. Gauss, C.: Theoria Motus Corporum Coelestium in Sectionibus Conicis Solem Mabientium [Theory of the Motion of the heavenly Bodies Moving about the Sun in Conic Sections]. In: Perthes and Besser, Hamburg, Germany (1809), English translation available at `http://name.umdl.umich.edu/AGG8895.0001.001`
15. Gentleman, W.: Least squares computations by Givens transformations without square roots. IMA J. of Appl. Math. 12, 329–336 (1973)
16. Gill, P., Golub, G., Murray, W., Saunders, M.: Methods for modifying matrix factorizations. Mathematics and Computation 28(126), 505–535 (1974)
17. Golub, G., Loan, C.V.: Matrix Computations, 3rd edn. Johns Hopkins University Press, Baltimore (1996)
18. Hall, B.: Lie Groups, Lie Algebras, and Representations: An Elementary Introduction. Springer, Heidelberg (2000)
19. Heggernes, P., Matstoms, P.: Finding good column orderings for sparse QR factorization. In: Second SIAM Conference on Sparse Matrices (1996)
20. Julier, S., Uhlmann, J.: A counter example to the theory of simultaneous localization and map building. In: IEEE Intl. Conf. on Robotics and Automation (ICRA), vol. 4, pp. 4238–4243 (2001)
21. Kaess, M., Dellaert, F.: Covariance recovery from a square root information matrix for data association. Journal of Robotics and Autonomous Systems 57, 1198–1210 (2009), doi:10.1016/j.robot.2009.06.008
22. Kaess, M., Ranganathan, A., Dellaert, F.: Fast incremental square root information smoothing. In: Intl. Joint Conf. on AI (IJCAI), Hyderabad, India, pp. 2129–2134 (2007)
23. Kaess, M., Ranganathan, A., Dellaert, F.: iSAM: Incremental smoothing and mapping. IEEE Trans. Robotics 24(6), 1365–1378 (2008)
24. Krauthausen, P., Dellaert, F., Kipp, A.: Exploiting locality by nested dissection for square root smoothing and mapping. In: Robotics: Science and Systems, RSS (2006)
25. Kschischang, F., Frey, B., Loeliger, H.A.: Factor graphs and the sum-product algorithm. IEEE Trans. Inform. Theory 47(2) (2001)
26. Olson, E., Leonard, J., Teller, S.: Fast iterative alignment of pose graphs with poor initial estimates. In: IEEE Intl. Conf. on Robotics and Automation (ICRA) (2006)
27. Paskin, M.: Thin junction tree filters for simultaneous localization and mapping. In: Intl. Joint Conf. on AI (IJCAI) (2003)
28. Paz, L.M., Pinies, P., Tardós, J.D., Neira, J.: Large scale 6DOF SLAM with stereo-in-hand. IEEE Transactions on Robotics 24(5), 946–957 (2008)
29. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann, San Francisco (1988)

30. Pothen, A., Sun, C.: Distributed multifrontal factorization using clique trees. In: Proc. of the Fifth SIAM Conf. on Parallel Processing for Scientific Computing, pp. 34–40. Society for Industrial and Applied Mathematics, Philadelphia (1992)
31. Ranganathan, A., Kaess, M., Dellaert, F.: Loopy SAM. In: Intl. Joint Conf. on AI (IJCAI), Hyderabad, India, pp. 2191–2196 (2007)
32. Smith, R., Self, M., Cheeseman, P.: A stochastic map for uncertain spatial relationships. In: Int. Symp on Robotics Research (1987)
33. Tarjan, R., Yannakakis, M.: Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs and selectively reduce acyclic hypergraphs. SIAM J. Comput. 13(3), 566–579 (1984)
34. Thrun, S., Burgard, W., Fox, D.: Probabilistic Robotics. The MIT press, Cambridge (2005)

Monte Carlo Value Iteration for Continuous-State POMDPs

Haoyu Bai, David Hsu, Wee Sun Lee, and Vien A. Ngo

Abstract. Partially observable Markov decision processes (POMDPs) have been successfully applied to various robot motion planning tasks under uncertainty. However, most existing POMDP algorithms assume a discrete state space, while the natural state space of a robot is often continuous. This paper presents *Monte Carlo Value Iteration* (MCVI) for continuous-state POMDPs. MCVI samples both a robot's state space and the corresponding belief space, and avoids inefficient a priori discretization of the state space as a grid. Both theoretical results and preliminary experimental results indicate that MCVI is a promising new approach for robot motion planning under uncertainty.

1 Introduction

A challenge in robot motion planning and control is the uncertainty inherent in robots' actuators and sensors. Incorporating uncertainty into planning leads to much more reliable robot operation.

Partially observable Markov decision processes (POMDPs) provide a principled general framework for modeling uncertainty and planning under uncertainty. Although POMDPs are computationally intractable in the worst case [13], point-based approximation algorithms have drastically improved the speed of POMDP planning in recent years [12, 14, 19, 20]. Today, the fastest POMDP algorithms, such as HSVI [19] and SARSOP [12], can solve moderately complex POMDPs with hundreds of thousands states in reasonable time. POMDPs have been used successfully to model a variety of robotic tasks, including navigation [3, 18], grasping [8], target tracking [10, 14], and exploration [19]. Most of the existing point-based POMDP algorithms, however, assume a discrete state space, while the natural state space for a robot is often continuous. Our primary goal is to develop a principled and practical POMDP algorithm for robot motion planning in continuous state spaces.

Haoyu Bai · David Hsu · Wee Sun Lee · Vien A. Ngo
Department of Computer Science, National University of Singapore, Singapore
e-mail: {haoyu,dyhsu,leews,ngoav}@comp.nus.edu.sg

If the state space S is continuous, one common way of using existing POMDP algorithms would be to place a regular grid over S and construct a discrete POMDP model first. The difficulty with this approach is that the number of states grow exponentially with the robot's degrees of freedom (DoFs), resulting in the "curse of dimensionality" well known in geometric motion planning (without uncertainty). The effect of a large number of states is in fact aggravated in POMDP planning. Due to uncertainty, the robot's state is not known exactly and is modeled as a *belief*, which can be represented as a probability distribution over S. We plan in the *belief space* $\mathcal{B}$, which consists of all possible beliefs. The result of POMDP planning is a *policy*, which tells the robot how to act at any belief $b \in \mathcal{B}$. A standard belief representation is a vector b, in which an entry $b(s)$ specifies the probability of the robot being in the discretized state $s \in S$. The *dimensionality* of $\mathcal{B}$ is then equal to the number of states in the discrete POMDP model.

Probabilistic sampling is a powerful idea for attacking the curse of dimensionality [23]. In geometric motion planning, the idea of probabilistically sampling a robot's configuration space led to tremendous progress in the last two decades [5]. Similarly, in POMDP planning, a key idea of point-based algorithms is to sample a small set of points from the belief space $\mathcal{B}$ as an approximate representation of $\mathcal{B}$ rather than represent $\mathcal{B}$ exactly. However, this is not enough, if the robot's state space S is continuous. To compute a policy, we need to evaluate the effect of executing a sequence of actions with an initial belief b. Conceptually, we apply the sequence of actions to *each* state $s \in S$ and average the execution results with probabilistic weights $b(s)$. It is clearly impossible to perform this computation exactly in finite time, as there are infinitely many states in a continuous state space S.

In this paper, we propose *Monte Carlo Value Iteration* (MCVI) for continuous state POMDPs. MCVI samples both a robot's state space S and the corresponding belief space $\mathcal{B}$, and avoids inefficient a priori discretization of the state space as a grid. The main technical innovation of MCVI is to use Monte Carlo sampling in conjunction with dynamic programming to compute a policy represented as a finite state controller. We show that, under suitable conditions, the computed policy approximates the optimal policy with a guaranteed error bound. We also show preliminary results of the algorithm applied to several distinct robotic tasks, including navigation, grasping, and exploration.

In the following, we start with some preliminaries on POMDPs and related work (Section 2). Next, we describe the main idea of MCVI and the algorithmic details (Section 3). We then present experimental results (Section 4). Finally, we conclude with some remarks on future research directions.

2 Background

2.1 Preliminaries on POMDPs

A POMDP models an agent taking a sequence of actions under uncertainty to maximize its total reward. In each time step, the agent takes an action $a \in A$ and moves from a state $s \in S$ to $s' \in S$, where S and A are the agent's state space and action space, respectively. Due to the uncertainty in actions, the end state s' is represented as a conditional probability function $T(s,a,s') = p(s'|s,a)$, which gives the probability that the agent lies in s', after taking action a in state s. The agent then takes an observation $o \in O$, where O is the observation space. Due to the uncertainty in observations, the observation result is also represented as a conditional probability function $Z(s',a,o) = p(o|s',a)$ for $s' \in S$ and $a \in A$. To elicit desirable agent behavior, we define a reward function $R(s,a)$. In each time step, the agent receives a real-valued reward $R(s,a)$, if it is in state $s \in S$ and takes action $a \in A$. The agent's goal is to maximize its expected total reward by choosing a suitable sequence of actions. When the sequence of actions has infinite length, we typically specify a discount factor $\gamma \in (0,1)$ so that the total reward is finite and the problem is well defined. In this case, the expected total reward is given by $\mathrm{E}\big(\sum_{t=0}^{\infty} \gamma^t R(s_t,a_t)\big)$, where s_t and a_t denote the agent's state and action at time t, respectively.

The goal of POMDP planning is to compute an *optimal policy* π^* that maximizes the agent's expected total reward. In the more familiar case where the agent's state is fully observable, a policy prescribes an action, given the agent's current state. However, a POMDP agent's state is partially observable and modeled as a belief, i.e., a probability distribution over S. A POMDP policy $\pi\colon \mathcal{B} \to A$ maps a belief $b \in \mathcal{B}$ to the prescribed action $a \in A$.

A policy π induces a *value function* $V_\pi\colon \mathcal{B} \to \mathrm{R}$. The *value* of b with respect to π is the agent's expected total reward of executing π with initial belief b:

$$V_\pi(b) = \mathrm{E}\Big(\sum_{t=0}^{\infty} \gamma^t R(s_t,a_t) \;\Big|\; \pi, b\Big). \tag{1}$$

If the action space and the observation spaces of a POMDP are discrete, then the optimal value function V^* can be approximated arbitrarily closely by a piecewise-linear, convex function [15]:

$$V(b) = \max_{\alpha \in \Gamma} \int_{s \in S} \alpha(s) b(s)\, ds, \tag{2}$$

where each $\alpha \in \Gamma$ is a function over S and commonly called an α-function. If the state space is also discrete, we can represent beliefs and α-functions as vectors and replace the integral in (2) by a sum. For each fixed α, $h(b) = \sum_{s \in S} \alpha(s) b(s)$ then defines a hyperplane over $\mathcal{B}$, and $V(b)$ is the maximum over a finite set of hyperplanes at b. In this case, it is clear why $V(b)$ is piecewise-linear and convex.

POMDP policy computation is usually performed offline, because of its high computational cost. Given a policy π, the control of the agent's actions is performed online in real time. It repeatedly executes two steps. The first step is action selection. If the agent's current belief is b, it takes the action $a = \pi(b)$, according to the given policy π. The second step is belief update. After taking an action a and receiving an observation o, the agent updates its belief:

$$b'(s') = \tau(b,a,o) = \eta Z(s',a,o)\int_{s\in S} T(s,a,s')b(s)\,ds, \tag{3}$$

where η is a normalizing constant.

More information on POMDPs is available in [11, 22].

2.2 Related Work

POMDPs provide a principled general framework for planning under uncertainty, but they are often avoided in robotics, because of their high computational complexity. In recent years, point-based POMDP algorithms made significant progress in computing approximate solutions to discrete POMDPs [12, 14, 19, 20]. Their success hinges on two main ideas. First, they sample a small set of points from the belief space $\mathcal{B}$ and use it as an approximate representation of $\mathcal{B}$. Second, they approximate the optimal value function as a set of α-vectors. The α-vectors allow partial policies computed at one belief point to be used for other parts of $\mathcal{B}$ when appropriate, thus bringing substantial gain in computational efficiency.

In comparison, progress on continuous POMDPs has been much more limited, partly due to the difficulty of representing beliefs and value functions for POMDPs when high-dimensional, continuous state spaces are involved. As mentioned earlier, discretizing the state space with a regular grid often results in an unacceptably large number of states. One idea is to restrict beliefs and value functions to a particular parametric form, e.g., a Gaussian [3, 16] or a linear combination of Gaussians [4, 15]. For robots in complex geometric environments with many obstacles, uni-modal distributions, such as the Gaussian, are often inadequate. In theory, a linear combination of Gaussians can partially address this inadequacy. However, when the environment geometry contains many "discontinuities" due to obstacles, the number of Gaussian components required often grows too fast for the approach to be effective in practice. Other algorithms, such as MC-POMDP [21] and Perseus [15], use particle filters to represent beliefs. Perseus still uses a linear combination of Gaussians for value function representation and thus suffers some of the same shortcomings mentioned above. MC-POMDP represents a value function by storing its values at the sampled belief points and interpolating over them using Gaussians as kernel functions and KL divergence as the distance function. Interpolation in a belief space is not easy. KL divergence does not satisfy the metric properties, making it difficult to understand the interpolation error. Furthermore, choosing suitable parameter values for the Gaussian kernels involves some of the same difficulties as those in choosing an a priori discretization of the state space.

MCVI also uses the particle-based belief representation, but it exploits one key successful idea of point-based discrete POMDP algorithms: the α-vectors. It captures the α-functions implicitly as a policy graph [6, 11] and retains their main benefits by paying a computational cost. To construct the policy graph, MCVI makes use of approximate dynamic programming by sampling the state space and performing Monte Carlo (MC) simulations. Approximate dynamic programming has also been used in policy search for Markov decision processes (MDPs) and POMDPs without exploiting the benefits of α-functions [1].

MCVI takes the approach of offline policy computation. An alternative is to perform online search [17, 7]. These two approaches are complementary and can be combined to deal with challenging planning tasks with long time horizons.

3 Monte Carlo Value Iteration

In this paper, we focus on the main issue of continuous state spaces and make the simplifying assumption of discrete action and observation spaces.

3.1 Policy Graphs

One way of representing a policy is a *policy graph* G, which is a directed graph with labeled nodes and edges. Each node of G is labeled with an action $a \in A$, and each edge of G is labeled with an observation $o \in O$. To execute a policy π_G represented this way, we use a finite state controller whose states are the nodes of G. The controller starts in a suitable node v of G, and a robot, with initial belief b, performs the associated action a_v. If the robot then receives an observation o, the controller transitions from v to a new node v' by following the edge (v, v') with label o. The process then repeats. The finite state controller does not maintain the robot's belief explicitly, as in (3). It encodes the belief implicitly in the controller state based on the robot's initial belief b and the sequence of observations received.

For each node v of G, we may define an α-function α_v. Let $\pi_{G,v}$ denote a partial policy represented by G, when the controller always starts in node v of G. The value $\alpha_v(s)$ is the expected total reward of executing $\pi_{G,v}$ with initial robot state s:

$$\alpha_v(s) = \mathrm{E}\Big(\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\Big) = R(s, a_v) + \mathrm{E}\Big(\sum_{t=1}^{\infty} \gamma^t R(s_t, a_t)\Big) \tag{4}$$

Putting (4) together with (1) and (2), we define the value of b with respect to π_G as

$$V_G(b) = \max_{v \in G} \int_{s \in S} \alpha_v(s) b(s) ds. \tag{5}$$

So V_G is completely determined by the α-functions associated with the nodes of G.

3.2 MC-Backup

The optimal POMDP value function V^* can be computed with *value iteration* (VI), which is based on the idea of dynamic programming [2]. An iteration of VI is commonly called a *backup*. The backup operator H constructs a new value function V_{t+1} from the current value function V_t:

$$V_{t+1}(b) = HV_t(b) = \max_{a \in A} \Big\{ R(b, a) + \gamma \sum_{o \in O} p(o|b, a) V_t(b') \Big\}, \tag{6}$$

where $R(b,a) = \int_{s\in S} R(s,a)b(s)ds$ is the robot's expected immediate reward and $b' = \tau(b,a,o)$ is the robot's next belief after it takes action a and receives observation o. At every $b \in \mathcal{B}$, the backup operator H looks ahead one step and chooses the action that maximizes the sum of the expected immediate reward and the expected total reward at the next belief. Under fairly general conditions, V_t converges to the unique optimal value function V^*.

Representing a value function as a set of α-functions has many benefits, but storing and computing α-functions over high-dimensional, continuous state spaces is difficult (Section 2.2). We do not represent a value function explicitly as a set of α-functions, but instead represent it implicitly as a policy graph. Let V_G denote the value function for the current policy graph G. Substituting (5) into (6), we get

$$HV_G(b) = \max_{a\in A}\Big\{\int_{s\in S} R(s,a)b(s)ds + \gamma \sum_{o\in O} p(o|b,a) \max_{v\in G} \int_{s\in S} \alpha_v(s)b'(s)ds\Big\}. \quad (7)$$

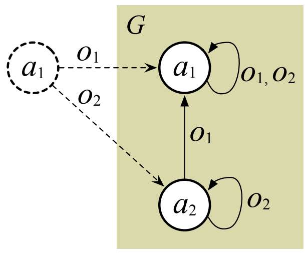


Fig. 1 Backup a policy graph G. The dashed lines indicate the new node and edges.

Let us first evaluate (7) at a particular point $b \in \mathcal{B}$ and construct the resulting new policy graph G', which contains a new node u and a new edge from u for each $o \in O$ (Fig. 1). Since we do not maintain α-functions explicitly, it seems difficult to compute the integral $\int_{s\in S} \alpha_v(s)b'(s)ds$. However, the definition of α_v in (4) suggests computing the integral by MC simulation: repeatedly sample a state s with probability $b'(s)$ and then simulate the policy $\pi_{G,v}$. Pushing further on this idea, we can in fact evaluate the entire right-hand side of (7) via sampling and MC simulation, and construct the new policy graph G'. We call this MC-backup of G at b (Algorithm 1).

Conceptually, Algorithm 1 considers all possible ways of generating G'. The new node u in G' has $|A|$ possible labels, and each outgoing edge from u has $|G|$ possible end nodes in G, where $|G|$ denotes the number of nodes in G (Fig. 1). Thus, there are $|A||G|^{|O|}$ candidates for G'. Each candidate graph G' defines a new policy $\pi_{G',u}$. We draw N samples to estimate the value of b with respect each candidate $\pi_{G',u}$. For each sample, we pick s from the state space S with probability $b(s)$. We run an MC simulation under $\pi_{G',u}$, starting from s, for L steps and calculate the total reward $\sum_{t=0}^{L} \gamma^t R(s_t,a_t)$. The simulation length L is selected to be is sufficiently large so that the error due to the finite simulation steps is small after discounting. We then choose the candidate graph with the highest average total reward. Unfortunately, this naive procedure requires an exponential number of samples.

Algorithm 1 computes the same result, but is more efficient, using only $N|A||G|$ samples. The loop in line 3 matches the maximization over actions $a \in A$ in (7). The loop in line 4 matches the first integral over states $s \in S$ and the sum over observations $o \in O$. The loop in line 8 matches the maximization over nodes $v \in G$. The three nested loops generate the simulation results and store them in $V_{a,o,v}$ for

Algorithm 1. Backup a policy graph G at a belief b with N samples.

```
MC-BACKUP($G, b, N$)
 1. For each action $a \in A$, $R_a \leftarrow 0$.
 2. For each action $a \in A$, each observation $o \in O$, and each node $v \in G$, $V_{a,o,v} \leftarrow 0$.
 3. for each action $a \in A$ do
 4.    for $i = 1$ to $N$ do
 5.       Sample a state $s_i$ with probability distribution $b(s_i)$.
 6.       Simulate taking action $a$ in state $s_i$. Generate the new state $s'_i$ by sampling from
          the distribution $T(s_i, a, s'_i) = p(s'_i|s_i, a)$. Generate the resulting observation $o_i$ by
          sampling from the distribution $Z(s'_i, a, o_i) = p(o_i|s'_i, a)$.
 7.       $R_a \leftarrow R_a + R(s_i, a)$.
 8.       for each node $v \in G$ do
 9.          Set $V'$ to be the expected total reward of simulating the policy represented by $G$,
             with initial controller state $v$ and initial robot state $s'_i$.
10.          $V_{a,o_i,v} \leftarrow V_{a,o_i,v} + V'$.
11.    for each observation $o \in O$ do
12.       $V_{a,o} \leftarrow \max_{v \in G} V_{a,o,v}$,   $v_{a,o} \leftarrow \arg\max_{v \in G} V_{a,o,v}$.
13.    $V_a \leftarrow (R_a + \gamma \sum_{o \in O} V_{a,o})/N$.
14. $V^* \leftarrow \max_{a \in A} V_a$,   $a^* \leftarrow \arg\max_{a \in A} V_a$.
15. Create a new policy graph $G'$ by adding a new node $u$ to $G$. Label $u$ with $a^*$. For each
    $o \in O$, add the edge $(u, v_{a^*,o})$ and label it with $o$.
16. return $G'$.
```

$a \in A, o \in O,$ and $v \in G$. Using $V_{a,o,v}$, one can compare the values at b with respect to any candidate policy graphs and choose the best one (lines 11–14).

Interestingly, a relatively small number of samples are sufficient for MC-backup to be effective. Let $\hat{H}_b V_G$ denote the value function for the improved policy graph resulting from MC-backup of G at b. With high probability, $\hat{H}_b V_G$ approximates HV_G well at b, with error decreasing at the rate $\mathcal{O}(1/\sqrt{N})$. For simplicity, we assume in our analysis below that the simulation length L is infinite. Taking the finite simulation length into account adds another error term that decreases exponentially with L.

Theorem 1. *Let $R_{\max}$ be an upper bound on the magnitude of $R(s,a)$ over $s \in S$ and $a \in A$. Given a policy graph G and a point $b \in \mathcal{B}$,* MC-BACKUP(G,b,N) *produces an improved policy graph such that for any $\tau \in (0,1)$,*

$$\left|HV_G(b) - \hat{H}_b V_G(b)\right| \leq \frac{2R_{\max}}{1-\gamma}\sqrt{\frac{2\big(|O|\ln|G| + \ln(2|A|) + \ln(1/\tau)\big)}{N}},$$

with probability at least $1-\tau$.

Proof. There are $|A||G|^{|O|}$ candidates for the improved policy graph. Effectively MC-BACKUP uses N samples to estimate the value at b with respect to each candidate and chooses the best one.

First, we calculate the probability that all the estimates have small errors. Let σ_i be a random variable representing the total reward of the ith simulation under a candidate policy. Define $\sigma = \sum_{i=1}^{N} \sigma_i/N$. Using Hoeffding's inequality, we have $p(|\sigma - \mathrm{E}(\sigma)| \geq \epsilon) \leq 2\mathrm{e}^{-N\epsilon^2/2C^2}$, where $C = R_{\max}/(1-\gamma)$ and ϵ is a small positive constant. Let E_i denote the event that the estimate for the ith candidate policy has error greater than ϵ. Applying the union bound $p(\bigcup_i E_i) \leq \sum_i p(E_i)$, we conclude that the estimate for any of the candidate policy graphs has error greater than ϵ with probability at most $\tau = 2|A||G|^{|O|}\mathrm{e}^{-N\epsilon^2/2C^2}$. So we set $\epsilon = C\sqrt{\frac{2(|O|\ln|G|+\ln(2|A|)+\ln(1/\tau))}{N}}$.

Next, let G^* denote the best candidate policy graph and G^*_{MC} denote the candidate graph chosen by MC-BACKUP. Let σ^* and σ^*_{MC} be the corresponding estimates of the value at b in MC-BACKUP. Then,

$$\begin{aligned} HV_G(b) - \hat{H}_b V_G(b) &= \mathrm{E}(\sigma^*) - \mathrm{E}(\sigma^*_{\text{MC}}) \\ &= \mathrm{E}(\sigma^*) - \sigma^* + \sigma^* - \mathrm{E}(\sigma^*_{\text{MC}}) \\ &\leq \mathrm{E}(\sigma^*) - \sigma^* + \sigma^*_{\text{MC}} - \mathrm{E}(\sigma^*_{\text{MC}}), \end{aligned}$$

The inequality in the last line follows, as MC-BACKUP always chooses the candidate policy graph with the highest estimate. Thus $\sigma^* \leq \sigma^*_{\text{MC}}$. Finally, the result in the previous paragraph implies that $|\sigma^* - \mathrm{E}(\sigma^*)| \leq \epsilon$ and $|\sigma^*_{\text{MC}} - \mathrm{E}(\sigma^*_{\text{MC}})| \leq \epsilon$, with probability at least $1-\tau$. Hence, $|HV_G(b) - \hat{H}_b V_G(b)| \leq 2\epsilon$, and the conclusion follows. □

Now we combine MC-backup, which samples the state space S, and point-based POMDP planning, which samples the belief space $\mathcal{B}$. Point-based POMDP algorithms use a set B of points sampled from $\mathcal{B}$ as an approximate representation of $\mathcal{B}$. Let $\delta_B = \sup_{b\in\mathcal{B}} \min_{b'\in B} \|b - b'\|_1$ be the maximum L_1 distance from any point in $\mathcal{B}$ to the closest point in B. We say that B covers $\mathcal{B}$ well if δ_B is small. Suppose that we are given such a set B. In contrast to the standard VI backup operator H, which performs backup at every point in $\mathcal{B}$, the operator $\hat{H}_B$ applies MC-BACKUP(G, b, N) on a policy graph G at every point in B. Each invocation of MC-BACKUP(G, b, N) returns a policy graph with one additional node added to G. We take a union of the policy graphs from all the invocations over $b \in B$ and construct a new policy graph G'. Let V_0 be value function for some initial policy graph and $V_{t+1} = \hat{H}_B V_t$.

The theorem below bounds the approximation error between V_t and the optimal value function V^*.

Theorem 2. *For every* $b \in \mathcal{B}$ *and every* $\tau \in (0, 1)$,

$$\begin{aligned} |V^*(b) - V_t(b)| \leq \frac{2R_{\max}}{(1-\gamma)^2}\sqrt{\frac{2\big((|O|+1)\ln(|B|t) + \ln(2|A|) + \ln(1/\tau)\big)}{N}} \\ + \frac{2R_{\max}}{(1-\gamma)^2}\delta_B + \frac{2\gamma^t R_{\max}}{(1-\gamma)}, \end{aligned}$$

with probability at least $1-\tau$.

To keep the proof simple, the bound is not tight. The objective here is to identify the main sources of approximation error and quantify their effects. The bound consists of three terms. The first term depends on how well MC-backup samples S (Algorithm 1, line 5). It decays at the rate $\mathcal{O}(1/\sqrt{N})$. We can reduce this error by taking a suitably large number of samples from S. The second term, which contains δ_B, depends on how well B covers $\mathcal{B}$. We can reduce δ_B by sampling a sufficiently large set B to cover $\mathcal{B}$ well. The last term arises from a finite number t of MC-backup iterations and decays exponentially with t. Note that although MC-backup is performed over points in B, the error bound holds for every $b \in \mathcal{B}$.

To prove the theorem, we need a Lipschitz condition on value functions:

Lemma 1. *Suppose that a POMDP value function V can be represented as or approximated arbitrarily closely by a set of α-functions. For any $b, b' \in \mathcal{B}$, if $\|b - b'\|_1 \leq \delta$, then $|V(b) - V(b')| \leq \frac{R_{\max}}{1-\gamma}\delta$.*

We omit the proof of Lemma 1, as it is similar to an earlier proof [9] for the special case $V = V^*$. We are now ready to prove Theorem 2.

Proof (Theorem 2). Let $\epsilon_t = \max_{b \in B} |V^*(b) - V_t(b)|$ be the maximum error of $V_t(b)$ over $b \in B$. First, we bound the maximum error of $V_t(b)$ over any $b \in \mathcal{B}$ in terms of ϵ_t. For any point $b \in \mathcal{B}$, let b' be the closest point in B to b. Then

$$|V^*(b) - V_t(b)| \leq |V^*(b) - V^*(b')| + |V^*(b') - V_t(b')| + |V_t(b') - V_t(b)|$$

Applying Lemma 1 twice to V^* and V_t, respectively, and using $|V^*(b') - V_t(b')| \leq \epsilon_t$, we get

$$|V^*(b) - V_t(b)| \leq \frac{2R_{\max}}{1-\gamma}\delta_B + \epsilon_t. \tag{8}$$

Next, we bound the error ϵ_t. For any $b' \in B$,

$$\begin{aligned} |V^*(b') - V_t(b')| &\leq |HV^*(b') - \hat{H}_{b'}V_{t-1}(b')| \\ &\leq |HV^*(b') - HV_{t-1}(b')| + |HV_{t-1}(b') - \hat{H}_{b'}V_{t-1}(b')|, \end{aligned} \tag{9}$$

The inequality in the first line holds, because by definition, $V^*(b') = HV^*(b')$, $V^*(b') \geq V_t(b')$, and $V_t(b') \geq \hat{H}_{b'}V_{t-1}(b')$. It is well known that the operator H is a contraction: $\|HV - HV'\|_\infty \leq \gamma\|V - V'\|_\infty$ for any value functions V and V', where $\|\cdot\|_\infty$ denotes the L_∞ norm. The contraction property and (8) imply

$$|HV^*(b') - HV_{t-1}(b')| \leq \gamma\Big(\frac{2R_{\max}}{1-\gamma}\delta_B + \epsilon_{t-1}\Big). \tag{10}$$

Theorem 1 guarantees small approximation error with high probability for a single MC-backup operation. To obtain V_t, we apply $\hat{H}_B$ for t times and thus have $|B|t$ MC-backup operations in total. Suppose that each MC-backup fails to achieve small error with probability at most $\tau/|B|t$. Applying the union bound together with Theorem 1, every backup operation $\hat{H}_{b'}$ achieves

$$|HV_{t-1}(b') - \hat{H}_{b'}V_{t-1}(b')| \leq \frac{2R_{\max}}{1-\gamma}\sqrt{\frac{2\big(|O|\ln(|B|t) + \ln(2|A|) + \ln(|B|t/\tau)\big)}{N}}. \tag{11}$$

with probability at least $1 - \tau$. We then substitute the inequalities (9–11) into the definition of ϵ_t and derive a recurrence relation for ϵ_t. For any initial policy graph, the error ϵ_0 can be bounded by $2R_{\max}/(1-\gamma)$. Solving the recurrence relation for ϵ_t and substituting it into (8) gives us the final result. □

3.3 Algorithm

Theorem 2 suggests that by performing MC-backup over a set B of suitably sampled beliefs, we can approximate the optimal value function with a bounded error. To complete the algorithm, we need to resolve a few remaining issues. First, we need a method for sampling from the belief space and obtaining B. Next, $\hat{H}_B$ performs backup at every point in B, but for computational efficiency, we want to perform backup only at beliefs that lead to significant improvement in the value function approximation. Both issues occur in discrete POMDP algorithms as well and have been addressed in earlier work. Finally, we use particle filters [22] to represent beliefs over continuous state spaces. Particle filtering can be implemented very efficiently and has been used with great success in important robotic tasks, such as localization and SLAM [22].

We now give a short description of the algorithm. It shares the same basic structure with our SARSOP algorithm [12] for discrete POMDPs; however, it uses MC-backup and particle filtering to handle continuous state spaces.

Overview. The algorithm computes an approximation to an optimal policy by updating a policy graph G. To improve G, it samples beliefs incrementally and performs backup at selected sampled beliefs.

Let $\mathcal{R} \subseteq \mathcal{B}$ be a subset of points reachable from a given initial belief $b_0 \in \mathcal{B}$ under arbitrary sequences of actions and observations. Following the recent point-based POMDP planning approach, our algorithm samples a set of beliefs from this reachable space $\mathcal{R}$ rather than $\mathcal{B}$ for computational efficiency, as $\mathcal{R}$ is often much smaller than $\mathcal{B}$. The sampled beliefs form a tree $T_{\mathcal{R}}$. Each node of $T_{\mathcal{R}}$ represents a sampled belief $b \in \mathcal{R}$, and the root of $T_{\mathcal{R}}$ is the initial belief b_0. If b is a node of $T_{\mathcal{R}}$ and b' is a child of b in $T_{\mathcal{R}}$, then $b' = \tau(b, a, o)$ for some $a \in A$ and $o \in O$. By definition, the belief associated with every node in $T_{\mathcal{R}}$ lies in $\mathcal{R}$.

To sample new beliefs, our algorithm updates $T_{\mathcal{R}}$ by performing a search in $\mathcal{R}$. At each node b of $T_{\mathcal{R}}$, it maintains both upper and lower bounds on $V^*(b)$. We start from the root of $T_{\mathcal{R}}$ and traverse a single path down until reaching a leaf of $T_{\mathcal{R}}$. At a node b along the path, we choose action a that leads to the child node with the highest upper bound and choose observation o that leads to the child node making the largest contribution to the gap between the upper and lower bounds at the root of $T_{\mathcal{R}}$. These heuristics are designed to bias sampling towards regions that likely lead to improvement in value function approximation. If b is a leaf node, then we use the same criteria to choose a belief b' among all beliefs reachable from b with an action

$a \in A$ and an observation $o \in O$. We compute $b' = \tau(b, a, o)$ using particle filtering and create a new node for b' in $T_{\mathcal{R}}$ as a child of b. The sampling path terminates when it reaches a sufficient depth to improve the bounds at the root of $T_{\mathcal{R}}$. We then go back up this path to the root and perform backup at each node along the way to update the policy graph as well as to improve the upper and lower bound estimates. We repeat the sampling and backup procedures until the gap between the upper and lower bounds at the root of $T_{\mathcal{R}}$ is smaller than a pre-specified value.

Policy and lower bound backup. The lower bound at a tree node b is computed from the policy graph G. As G always represents a valid policy, $V_G(b)$ is a lower bound of $V^*(b)$. We initialize G with a simple default policy, e.g., always performing a single fixed action. To update the lower bound at b, we perform MC-backup on G at b. As a result, we obtain an updated policy graph G' and an MC estimate of the value at b with respect to G' as an improved lower bound.

Upper bound backup. To obtain the initial upper bound at a node b, one general approach is to apply standard relaxation techniques. Assuming that a robot's state variables are all fully observable, we can solve a corresponding MDP, whose value function provides an upper bound on the POMDP value function. By assuming that a robot's actions are deterministic, we can further relax to a deterministic planning problem. To update the upper bound at b, we use the standard backup operator.

The upper and lower bounds in our algorithm are obtained via sampling and MC simulations, and are thus approximate. The approximation errors decrease with the number of samples and simulations. Since the bounds are only used to guide belief space sampling, the approximation errors do not pose serious difficulties.

For lack of space, our algorithm description is quite brief. Some additional details that improve computational efficiency are available in [12], but they are independent of the use of MC-backup and particle filtering to deal with continuous state spaces.

4 Experiments

We implemented MCVI in C++ and evaluated it on three distinct robot motion planning tasks: navigation, grasping, and exploration. In each test, we used MCVI to compute a policy. We estimated the expected total reward of a policy by running a sufficiently large number of simulations and averaging the total rewards, and used the estimate as a measure of the quality of the computed policy. As MCVI is a randomized algorithm, we repeated each test 10 times and recorded the average results. All the computation was performed on a computer with a 2.66 GHz Intel processor under the Linux operating system.

4.1 Navigation

This 1-D navigation problem first appeared in the work on Perseus [15], an earlier algorithm for continuous POMDPs. A robot travels along a corridor with four doors (Fig. 2*a*). The robot's goal is to enter the third door from the left. The robot has

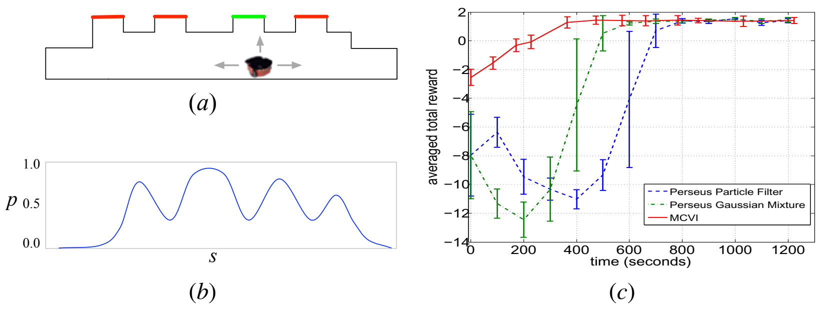


Fig. 2 Navigation in a corridor. (*a*) The environment. (*b*) The observation function for the observation CORRIDOR. (*c*) Estimated expected total rewards of computed policies.

three actions: MOVE-LEFT, MOVE-RIGHT, and ENTER. The robot does not know its exact location, but can gather information from four observations: LEFT-END, RIGHT-END, DOOR, and CORRIDOR, which indicate different locations along the corridor. Both the actions and observations are noisy. The robot receives a positive reward if it enters the correct door, and a negative reward otherwise.

For comparison with Perseus, we use the same model as that in [15]. Perseus requires that all the transition functions, observation functions, and reward functions are modeled as a combination of Gaussians. See Fig. 2*b* for an illustration of the observation function for the observation CORRIDOR. Details of the model are available in [15]. It is important to note that representing the entire model with Gaussians imposes a severe restriction. Doing so for more complex tasks, such as grasping and obstructed rock sample in the following subsections, is impractical.

We ran MCVI with 600 particles for belief representation and $N = 400$ for MC-BACKUP. We also ran Perseus using the original authors' Matlab program, with parameter settings suggested in [15]. There are two versions of Perseus using different belief representations. One version uses Gaussian mixture, and the other one uses particle filtering. The results are plotted in Fig. 2*c*. The horizontal axis indicates the time required for policy computation. The vertical axis indicates the average total reward of a computed policy. Each data point is the average over 10 runs of each algorithm. The error bars indicate the 95% confidence intervals.

Since MCVI is implemented in C++ and Perseus is implemented in Matlab, the running times are not directly comparable. However, the plot indicates that MCVI reaches the same performance level as Perseus, even though MCVI does not require a Gaussian model and does not take advantage of it. Also, the smaller error bars for MCVI indicate that it is more robust, especially when the planning time is short.

The main purpose of this test is to compare with Perseus, a well-known earlier algorithm for continuous POMDPs. As one would expect, the task is relatively simple. We can construct a discrete POMDP model for it and compute a policy efficiently using discrete POMDP algorithms.

4.2 Grasping

In this simplified grasping problem [8], a robot hand with two fingers tries to grasp a rectangular block on a table and lift it up (Fig. 3). The fingers have contact sensors at the tip and on each side. Thus, each observation consists of outputs from all the contact sensors. The observations are noisy. Each contact sensor has a 20% probability of failing to detect contact, when there is contact, but 0% probability of mistakenly detecting contact, when there is none. Initially, the robot is positioned randomly above the block. Its movement is restricted to a 2-D vertical plane containing both the hand and the block. The robot's actions include four compliant guarded moves: MOVE-LEFT, MOVE-RIGHT, MOVE-UP, and MOVE-DOWN. Each action moves the robot hand until a contact change is detected. The robot also has OPEN and CLOSE actions to open and close the fingers as well as a LIFT action to lift up the block. If the robot performs LIFT with the block correctly grasped, it is considered a success, and the robot receives a positive reward. Otherwise, the robot receives a negative reward. In this problem, uncertainty comes from the unknown initial position of the robot hand and noisy observations.

We ran MCVI with 150 particles for belief representation and $N = 500$ for MC-BACKUP. On the average, the planning time is 160 seconds, and the computed policy has a success rate of 99.7%. For comparison, we manually constructed a open-loop policy: MOVE-LEFT $\rightarrow$ MOVE-DOWN $\rightarrow$ MOVE-RIGHT $\rightarrow$ MOVE-UP $\rightarrow$ MOVE-RIGHT $\rightarrow$ MOVE-DOWN $\rightarrow$ CLOSE $\rightarrow$ LIFT. The success rate of this policy is only 77.2%. Many of the failures occur because the manually constructed policy does not adequately reason about noisy observations.

Fig. 3 shows a simulation run of the computed policy. In one MOVE-LEFT action (Fig. 3*f*), the tip contact sensor of the left finger fails to detect the top surface of the block. At a result, the robot does not end the MOVE-LEFT action in the proper position, but it recovers from the failure when the tip contact sensor of the right finger correctly detects contact (Fig. 3*h*).

The grasping problem can also be modeled as a discrete POMDP [8]. However, this requires considerable efforts in analyzing the transition, observation, and reward functions. Although the resulting discrete POMDP model is typically more compact than the corresponding continuous POMDP model, the discretization process may be difficult to carry out, especially in complex geometric environments. In contrast, MCVI operates on continuous state spaces directly and is much easier to use.

4.3 Obstructed Rock Sample

The original Rock Sample problem [19] is a benchmark for new discrete POMDP algorithms. In this problem, a planetary rover explores an area and searches for rocks with scientific value. The rover always knows its own position exactly, as well as those of the rocks. However, it does not know which rocks are valuable. It uses the SENSE action to take noisy long-range sensor readings on the rocks. The accuracy of the readings depends on the distance between the rover and the rocks. The rover can also apply the SAMPLE action on a rock in the immediate vicinity and receive

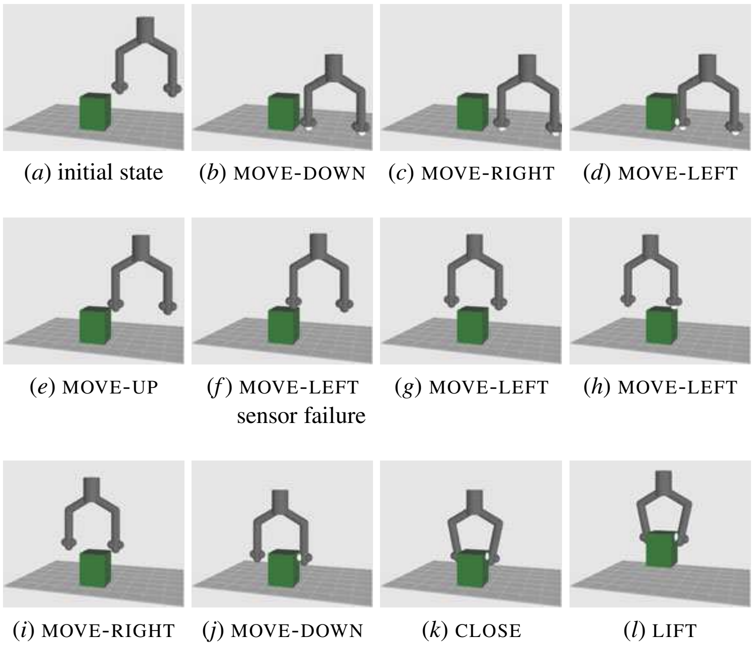


Fig. 3 A simulation run of the simplified grasping task. The spheres at the tip and the sides of the fingers indicate contact sensors. They turn white when contact is detected.

a positive or negative reward, depending on whether the sampled rock is actually valuable. The robot's goal is to find as many valuable rocks as possible quickly and then move to the right boundary of the environment to report the results.

We extended Rock Sample in several ways to make it more realistic. We introduced obstructed regions, which the rover cannot travel through. Furthermore, the rover's movement is now noisy. In each time step, the rover can choose to move in any of eight equally spaced directions with two speed settings. Finally, the rover does not always know its own location exactly. It can only localize in the immediate vicinity of a rock, which serves as a landmark. We call this extended version *Obstructed Rock Sample* (ORS).

We created three models of ORS by varying the noise levels for the rover's movements and long-range rock sensor. We ran MCVI on each model. The average planning time ranges from 5 minutes for the low-noise model to a maximum of 2 hours.

Fig. 4 shows a simulation run for each computed policy. For the low-noise model (Fig. 4*a*), the rover first moves towards the top-left rock. It senses the rock and decides to sample it. It also senses the lower-left rock, but cannot determine whether the rock is valuable, because the rock is far away and the sensor reading is too noisy. The rover then approaches the lower-left rock and senses it again. Together the two sensor readings indicate that the rock is likely bad. So the rover does not sample it. Along the way, the rover also senses the top-right rock twice and decides that

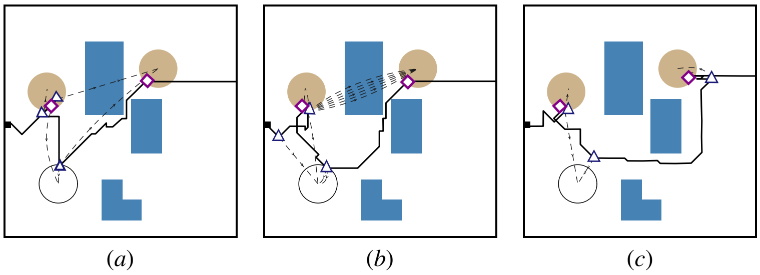


Fig. 4 Simulations runs for three ORS models: (*a*) low noise in sensing and movements, (*b*) higher sensor noise, and (*c*) higher movement noise. Shaded polygons indicate obstructed regions. Shaded and white discs indicate the regions in which the rover may perform the SAMPLE action. The rocks are located at the center of the discs. Shaded discs represent valuable rocks, and white discs represent bad rocks. Solid black curves indicates the rover's trajectories. Each "◇" marks a location where the rover performs a SAMPLE action. Each "△" marks a location where the rover performs a SENSE action, and the corresponding dashed line indicates the rock being sensed.

the rock is likely valuable. As the movement noise is low, the rover chooses to go through the narrow space between two obstacles to reach the rock and sample it. It then takes a shortest path to the right boundary. We do not have a good way of determining how well the computed policy approximates an optimal one. In this simulation run, the jaggedness in the rover's path indicates some amount of sub-optimality. However, the rover's overall behavior is reasonable. When the sensor noise in the model is increased (Fig. 4*b*), the rover maintains roughly the same behavior, but it must perform many more sensing actions to determine whether a rock is valuable. When the movement noise is increased (Fig. 4*c*), the rover decides that it is too risky to pass through the narrow space between obstacles and takes an alternative safer path.

A standard discrete POMDP model of Rock Sample uses a grid map of the environment. Typically discrete POMDP algorithms can handle a 10×10 grid in reasonable time. This is inadequate for complex geometric environments. The environment shown in Fig. 4, which consists of relatively simple geometry, requires a grid of roughly 50×50, due to closely spaced obstacles. A discrete POMDP model of this size requires about 4 GB of memory before any computation is performed. MCVI avoids this difficulty completely.

4.4 Discussion

While the experimental results are preliminary, the three different examples indicate that MCVI is flexible and relatively easy to use. It does not require artificial discretization of a continuous state space as a grid. It also does not impose restriction on the parametric form of the model.

Our current implementation of MCVI uses fixed values for the number of particles, M, for belief representation and the parameter N in MC-BACKUP. Our experimental results show that MC-BACKUP typically takes around 99% of the total running time and is the dominating factor. To improve efficiency, we may use the sample variance of the simulations to set N adaptively and stop the simulations as soon as the variance becomes sufficiently small. We may over-estimate M, as this does not affect the total running time significantly.

5 Conclusion

POMDPs have been successfully applied to various robot motion planning tasks under uncertainty. However, most existing POMDP algorithms assume a discrete state space, while the natural state space of a robot is often continuous. This paper presents Monte Carlo Value Iteration for continuous-state POMDPs. MCVI samples both a robot's state space and the corresponding belief space, and computes a POMDP policy represented as a finite state controller. The use of Monte Carlo sampling enables MCVI to avoid the difficulty of artificially discretizing a continuous state space and make it much easier to model robot motion planning tasks under uncertainty using POMDPs. Both theoretical and experimental results indicate that MCVI is a promising new approach for robot motion planning under uncertainty.

We are currently exploring several issues to improve MCVI. First, the running time of MCVI is dominated by MC simulations in MC-backup. We may group similar states together and avoid repeated MC simulations from similar states. We may also parallelize the simulations. Parallelization is easy here, because all the simulations are independent. Second, the size of a policy graph in MCVI grows over time. We plan to prune the policy graph to make it more compact [6]. Finally, an important issue is to deal with not only continuous state spaces, but also continuous observation and action spaces.

Acknowledgments. This work is supported in part by MoE AcRF grant R-252-000-327-112 and MDA GAMBIT grant R-252-000-398-490.

References

1. Bagnell, J.A., Kakade, S., Ng, A., Schneider, J.: Policy search by dynamic programming. In: Advances in Neural Information Processing Systems (NIPS), vol. 16 (2003)
2. Bellman, R.E.: Dynamic Programming. Princeton University Press, Princeton (1957)
3. Brooks, A., Makarendo, A., Williams, S., Durrant-Whyte, H.: Parametric POMDPs for planning in continuous state spaces. Robotics & Autonomous Systems 54(11), 887–897 (2006)
4. Brunskill, E., Kaelbling, L., Lozano-Perez, T., Roy, N.: Continuous-state POMDPs with hybrid dynamics. In: Int. Symp. on Artificial Intelligence & Mathematics (2008)
5. Choset, H., Lynch, K.M., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, L.E., Thrun, S.: Principles of Robot Motion: Theory, Algorithms, and Implementations, vol. ch. 7. The MIT Press, Cambridge (2005)

6. Hansen, E.A.: Solving POMDPs by searching in policy space. In: Proc. AAAI Conf. on Artificial Intelligence, pp. 211–219 (1998)
7. He, R., Brunskill, E., Roy, N.: PUMA: Planning under uncertainty with macro-actions. In: Proc. AAAI Conf. on Artificial Intelligence (2010)
8. Hsiao, K., Kaelbling, L.P., Lozano-Pérez, T.: Grasping POMDPs. In: Proc. IEEE Int. Conf. on Robotics & Automation, pp. 4485–4692 (2007)
9. Hsu, D., Lee, W.S., Rong, N.: What makes some POMDP problems easy to approximate? In: Advances in Neural Information Processing Systems (NIPS) (2007)
10. Hsu, D., Lee, W.S., Rong, N.: A point-based POMDP planner for target tracking. In: Proc. IEEE Int. Conf. on Robotics & Automation, pp. 2644–2650 (2008)
11. Kaelbling, L.P., Littman, M.L., Cassandra, A.R.: Planning and acting in partially observable stochastic domains. Artificial Intelligence 101(1-2), 99–134 (1998)
12. Kurniawati, H., Hsu, D., Lee, W.S.: SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In: Proc. Robotics: Science and Systems (2008)
13. Papadimitriou, C., Tsisiklis, J.N.: The complexity of Markov decision processes. Mathematics of Operations Research 12(3), 441–450 (1987)
14. Pineau, J., Gordon, G., Thrun, S.: Point-based value iteration: An anytime algorithm for POMDPs. In: Proc. Int. Jnt. Conf. on Artificial Intelligence, pp. 477–484 (2003)
15. Porta, J.M., Vlassis, N., Spaan, M.T.J., Poupart, P.: Point-based value iteration for continuous POMDPs. J. Machine Learning Research 7, 2329–2367 (2006)
16. Prentice, S., Roy, N.: The belief roadmap: Efficient planning in linear pomdps by factoring the covariance. In: Proc. Int. Symp. on Robotics Research (2007)
17. Ross, S., Pineau, J., Paquet, S., Chaib-Draa, B.: Online planning algorithms for POMDPs. J. Artificial Intelligence Research 32(1), 663–704 (2008)
18. Roy, N., Thrun, S.: Coastal navigation with mobile robots. In: Advances in Neural Information Processing Systems (NIPS), vol. 12, pp. 1043–1049 (1999)
19. Smith, T., Simmons, R.: Point-based POMDP algorithms: Improved analysis and implementation. In: Proc. Uncertainty in Artificial Intelligence (2005)
20. Spaan, M.T.J., Vlassis, N.: A point-based POMDP algorithm for robot planning. In: Proc. IEEE Int. Conf. on Robotics & Automation (2004)
21. Thrun, S.: Monte carlo POMDPs. In: Advances in Neural Information Processing Systems (NIPS). The MIT Press, Cambridge (2000)
22. Thrun, S., Burgard, W., Fox, D.: Probabilistic Robotics. The MIT Press, Cambridge (2005)
23. Traub, J.F., Werschulz, A.G.: Complexity and Information. Cambridge University Press. Cambridge (1998)

Randomized Belief-Space Replanning in Partially-Observable Continuous Spaces

Kris Hauser

Abstract. We present a sample-based replanning strategy for driving partially-observable, high-dimensional robotic systems to a desired goal. At each time step, it uses forward simulation of randomly-sampled open-loop controls to construct a belief-space search tree rooted at its current belief state. Then, it executes the action at the root that leads to the best node in the tree. As a node quality metric we use Monte Carlo simulation to estimate the likelihood of success under the QMDP belief-space feedback policy, which encourages the robot to take information-gathering actions as needed to reach the goal. The technique is demonstrated on target-finding and localization examples in up to 5D state spacess.

1 Introduction

Many robotics problems involve planning in uncertain, partially-observable domains, which requires reasoning about how hypothetical state distributions, *belief states*, change over time as the robot acts upon the world and gathers information with its sensors. Although this has been studied heavily in discrete domains, most realistic robotics problems have continuous, high-dimensional state spaces with nonlinear dynamics, which places them far out of the reach of tractability for state-of-the-art planners built for discrete systems. Although recent techniques have made progress in addressing continuous systems assuming Gaussian process and observation noise [3, 19, 21], the more general case of nonlinear and multi-modal belief states have proven to be much more challenging, in large part because of the difficulty of representing policies over an infinite-dimensional belief space [20, 22].

Kris Hauser
School of Informatics and Computing, Indiana University
e-mail: `hauserk@indiana.edu`

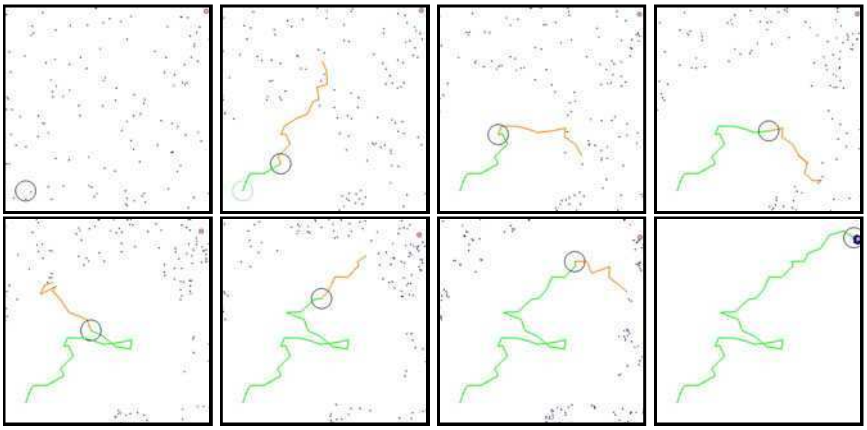


Fig. 1 An execution trace of a robot (large circle) searching for a wandering target (pink circle) in the unit square. The robot's sensor has a 0.25 unit range. The current belief state is represented by 100 particles (dots) and the current plan (orange) is updated by replanning.

We present a Randomized Belief-Space Replanning (RBSR) technique that addresses an undiscounted and cost-free version of the continuous partially-observable Markov decision process (POMDP) formulation, where the belief state must be driven to a goal region. Rather than solving the POMDP once and using the solution as a lookup table, RBSR repeatedly generates coarse plans, executes the first step, and uses sensor feedback to refine future plans (Figure 1). Much like a receding-horizon controller or model predictive controller, its success rate and computation time depend on the exploration strategy used to generate the search trees, and the evaluation function used to pick the "best" plan.

The RBSR exploration strategy performs a forward search in belief space by randomly sampling open-loop actions, and for an evaluation function we estimate the success rate of a QMDP-like policy. QMDP is a heuristic strategy that descends the cost-to-go function of the underlying MDP, averaged over the belief state [16], and it works well when state uncertainty is low, but with high uncertainty it may fall into local minima because it fails to perform active information-gathering. Hence, RBSR's random exploration strategy discourages information loss and encourages information-gathering actions because they improve the likelihood that QMDP succeeds.

RBSR employs random sampling approaches at multiple points in the procedure — random belief-space exploration strategies, particle filtering for state estimation, probabilistic roadmap-like approaches in the QMDP policy evaluation, and Monte-Carlo simulation in the evaluation function — making it highly parallelizable and applicable to high-dimensional spaces. In preliminary experiments, we applied RBSR to a simulated target pursuit problem with a 4-D state space and localization problems in up to 5-D (Figure 1). In our tests, RBSR computes each replanning step in seconds, and drives the belief state to a solution with high probability. Though our current implementation is promising, it is not as reliable in 6D or higher because it becomes much more difficult to maintain accurate belief estimates over time. Nevertheless, we anticipate that future implementations of RBSR will be capable of solving many real-world robotics problems.

2 Related Work

Optimal planning in partially-observable problems is extremely computationally complex and is generally considered intractable even for small discrete state spaces [17]. Approximate planning in discrete spaces is a field of active research, yielding several techniques based on the point-based algorithms devised by Kearns et al [12] and Pineau et al (2003) [18]. For example, the SARSOP algorithm developed by Kurniawati et al (2008) has solved problems with thousands of discrete states in seconds [14].

Hypothetically, these algorithms can be applied to continuous problems by discretizing the space. But because of the "curse of dimensionality", any regular discretization of a high-dimensional space will requires an intractably large number of states. Porta et al (2006) has made progress in extending point-based value iteration to the continuous setting by representing belief states as particles or mixtures of Gaussians [20]. Thrun (2000) presented a technique that also works with continuous spaces by combining particle filtering with reinforcement learning on belief states [22]. For both of these methods, the need to approximate the value function over the infinite-dimensional belief space (either using alpha-vector or Q-value representations, respectively) comes at a high computational and memory expense. We use similar representations, but because we use replanning to avoid explicit policy representation, our approach sacrifices near-optimality for reduction in computational expense.

Several recently developed algorithms attempt to address continuous spaces by leveraging the success of probabilistic roadmaps (PRMs) in motion planning [11], which build a network of states sampled at random from the configuration space. Alterovitz et al (2007) present a Stochastic Motion Roadmap planner for continuous spaces with motion uncertainty, which solves an MDP using the discretization of state space induced by a PRM [1]. The techniques of Burns and Brock (2007) and Guibas et al (2008) augment roadmaps with edge costs for motions that have high probability of being in collision, and respectively address the problems of localization errors and environment sensing errors [4, 7]. Huang and Gupta (2009) address planning for manipulators under base uncertainty by associating probabilistic roadmaps with particles representing state hypotheses and searching for a short path that is likely to be collision free [10].

Another set of related approaches use assumptions of Gaussian observation and process noise, which makes planning much faster because probabilistic

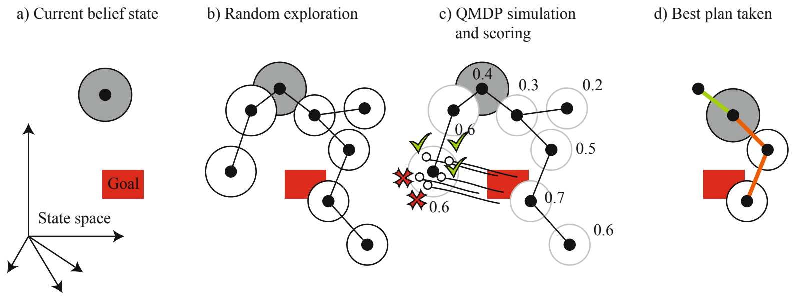


Fig. 2 Illustrating the replanning steps. (a) A belief-space search tree is initialized with the current belief state. (b) The tree is grown using random exploration of open-loop motions. (c) Nodes in the tree are scored with estimates of the likelihood of success under the QMDP policy. Traces of 5 belief-state particles under QMDP simulation are depicted. (d) The best plan is executed. (If the best node is the root, QMDP is executed by default).

inference can be performed in closed form. The Belief Roadmap technique of Prentice and Roy (2009) computes a roadmap of belief states under both motion and sensing uncertainty, under the assumptions of Gaussian uncertainty and linear transition and observation functions [21]. van den Berg et al (2010) consider path planning while optimizing the likelihood that a path is collision-free, under the assumption that a Linear-Quadratic-Gaussian feedback controller is used to follow the path. Platt et al (2010) and du Toit and Burdick (2010) construct plans using a maximum-likelihood observation assumption, and correcting for observation errors by replanning [6, 19]. RBSR also uses a replanning strategy, but uses a particle-based uncertainty representation that is better at handling nonlinear and multi-modal distributions, and makes no assumptions on the type of observations received.

The Randomized Path Planner (RPP) was an early approach in path planning in high-dimensional spaces that uses the principle that reactive policies often work well when the system is near the goal or when a space is relatively free of obstacles [2]. RPP plans by alternating steps of potential field descent and random walks to escape local minima, and was surprisingly effective at solving path planning problems that were previously considered intractable. RBSR shares a similar philosophy, but addresses problems with partial observablility.

3 Problem Definition

RBSR is given a POMDP-like model of the problem as input, and it interleaves planning and execution steps much like a receding-horizon controller. Each iteration performs the following steps:

1. The current sensor input is observed, and the robot's belief state is updated using a particle filter.
2. The planner generates a truncated search tree rooted at the current belief state. (Figure 2.a–b)
3. The robot executes the action associated with the "best" branch out of the root node. (Figure 2.c–d)

This section describes the POMDP formulation, particle filtering belief state update, and the QMDP policy that is used to evaluate the quality of nodes in the tree.

3.1 POMDP Modeling

The problem is formalized as an undiscounted partially-observable Markov decision process (POMDP) over a set of states S, actions A, and observations O. S, A, and O are treated as subsets of Cartesian space, although this is not strictly necessary. A *belief state* is defined to be a probability distribution over S. We address the setting where the robot starts at an initial belief state b_{init} and wishes to reach a goal set $\mathcal{G} \subseteq S$ with high probability. We treat obstacles by moving all colliding states to a special absorbing state. At discrete time steps the robot performs an action, which changes its (unobserved) state, and it receives an observation.

Although most POMDP formulations are concerned with optimizing rewards and action costs, we treat a somewhat simpler problem of simply maximizing the probability of reaching the goal. We also do not consider discounting. Discounting is numerically convenient and has a natural interpretation in economics, but in many respects is inappropriate for robotics problems because it gives preference to short term rewards.

The dynamics of the system are specified in the *transition model* $\mathcal{T} : s, a \to s'$ that generates a new state, given an existing state and an action. The sensor model is specified in the *sensor model* $\mathcal{O} : s \to o$ that generates an observation given a state. These models are stochastic, and we let the notation $s' \leftarrow \mathcal{T}(s, a)$ and $o \leftarrow \mathcal{O}(s)$ denote sampling at random from the posterior distributions of $\mathcal{T}$ and $\mathcal{O}$, respectively.

3.2 Simulation and Filtering with Belief Particles

To approximate the distribution over state hypotheses, we represent a belief state b as a weighted set of n particles $\{(w^{(1)}, s^{(1)}), \ldots, (w^{(n)}, s^{(n)})\}$, where n is a parameter, and the weights sum to 1. Using such a representation, we can easily simulate an observation $o \leftarrow \mathcal{O}(s^{(k)})$ by sampling particle $(w^{(k)}, s^{(k)})$ proportional to its weight. We also define functions that compute the successor belief state after executing action a:

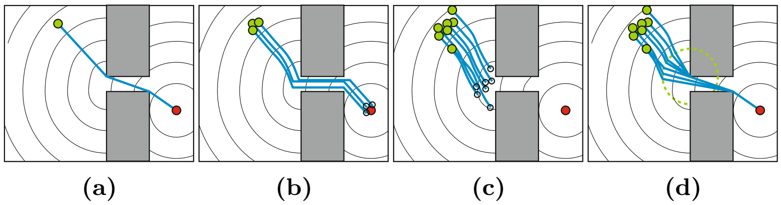


Fig. 3 The QMDP policy will succeed for well-localized belief states (a,b), but it may fall into local minima for a poorly localized belief state (c). On the other hand, QMDP allows the robot to incorporate new information during execution. So, if it could sense the corner of the obstacle as a landmark, then QMDP will also reach the goal (d).

$$\mathcal{T}(b,a) = \{(w^{(i)}, \mathcal{T}(s^{(i)}, a))\}_{i=1}^{n} \tag{1}$$

And the posterior belief state after observing o:

$$\texttt{Filter}(b,o) = \{(\frac{1}{Z} w^{(i)} Pr(O(s^{(i)}) = o), s^{(i)})\}_{i=1}^{n} \tag{2}$$

where Z is a normalization factor that ensures weights sum to 1.

We assume the robot performs state estimation using a particle filter, which have many variants that are beyond the scope of this work. We refer the reader to the survey in [5] for details. Most of these techniques address the problem of sample impoverishment that arises when few particles in b are consistent with a given sequence of observations. For the remainder of this paper, we will assume that the chosen filter is robust enough to maintain sufficiently representative belief states.

3.3 QMDP Policy

As an endgame strategy, RBSR uses the incomplete QMDP policy that is quite successful in practice for highly-localized belief states or when information can be gathered quickly to localize the state (see Figure 3). QMDP is also used in RBSR to define a function $f(b)$ that measure the quality of hypothetical belief states by simulating how well QMDP makes progress toward the goal.

The QMDP policy essentially takes the optimal action assuming full observability is attained on the next step [16]. Suppose we are given a complete cost-to-go function C_{fo} the fully-observable version of the problem. We will put aside the question of how to compute such a function until Section 3.4, and currently we describe how to use C_{fo} to derive a QMDP controller for the partially-observable belief space.

The belief-space policy $\pi_{\text{QMDP}}(b)$ is defined to descend the expected value of C_{fo} over the distribution of states in b. More precisely, we define

$$C(b) \equiv E_{s \sim b}[C_{fo}(s)] \approx \sum_{i=1}^{n} w^{(i)} C_{fo}(s^{(i)}), \tag{3}$$

and define π_{QMDP} to pick the action that descends $C(b)$ as quickly as possible:

$$\pi_{\text{QMDP}}(b) \equiv \arg\min_{a} C(\mathcal{T}(b,a)). \tag{4}$$

If the expected value of the resulting belief state does not reduce $C(b)$, we define $\pi_{\text{QMDP}}(b)$ to return "terminate". Collision states are assigned infinite potential. In practice, we compute the arg min in (4) by sampling many actions and picking the one that minimizes the RHS.

The QMDP policy alternates state estimation and potential field descent using the following feedback controller:

```
QMDP
Input: belief state $b_0$.
1. For $t = 1, 2, \ldots$, do:
2.     Sense observation $o_t$
3.     $b_o \leftarrow$ Filter($b_{t-1}, o_t$)
4.     $a_t = \pi_{\text{QMDP}}(b_o)$
5.     If $a_t =$"terminate," stop. Otherwise, execute $a_t$.
6.     $b_t \leftarrow \mathcal{T}(b_o, a_t)$
```

QMDP is also used in RBSR to measure quality of future belief states. We define a belief-state evaluation function $f(b)$ that uses Monte-Carlo simulation of QMDP on a holdout set of m particles $\{s^{(1)}, \ldots s^{(m)}\}$ from b which are used to simulate "ground truth". The complement of the holdout set b' is used as the initial belief state. For each test sample $s^{(i)}$, QMDP is invoked from the initial belief state $b_0 = b'$, and $s_0 = s^{(i)}$ is used for simulating the "true" observation $\mathcal{O}(s^{(i)})$ (Line 2). It is also propagated forward along with the belief state using the transition model $s_t \leftarrow \mathcal{T}(s_{t-1}, a_t)$ (Line 6). This continues until termination.

To enforce an ordering on $f(b)$ (with higher values better), we incorporate two results of the QMDP simulation: s, the fraction of terminal states s_t that lie in the goal $\mathcal{G}$, and c, the average QMDP value function evaluated at the terminal belief states $C(b_t)$. We prioritize success rate s over the value function c by letting $f(b)$ return a tuple $(s, -c)$. To compare the tuples $f(b_1) = (s_1, -c_1)$ and $f(b_2) = (s_2, -c_2)$ we use lexicographical order; that is, the value function is used only break ties on success rate. (This usually occurs when all locally reachable belief states have zero success rate.)

3.4 Computing Value Functions for the Fully-Observable Problem

Let us now return to the question of how one might provide a potential field C_{fo} for the fully-observable version of the input POMDP. The policy that descends C_{fo} is assumed to be complete, that is, if state is fully observable, then a descent of C_{fo} is guaranteed to reach the goal. Although in discrete POMDPs such a function can be computed using value iteration on the underlying MDP, the problem is more difficult in continuous POMDPs.

In problems with no motion uncertainty, C_{fo} is simply a cost function of the underlying motion planning problem. This can sometimes be computed analytically; e.g., for a robot with unit bounds on velocity in a convex workspace, C_{fo} is the straight-line distance to the goal. For more complex problems with high-dimensional or complex state spaces, approximate methods may be needed. In our examples we use a Probabilistic Roadmap (PRM) [11] embedded in $\mathcal{S}$, where each point in space is identified with its closest vertex in the roadmap, and the shortest distance from each vertex to the goal is computed using Dijkstra's algorithm. We build the PRM with a sufficiently large number of samples such that shortest paths in the roadmap approximate shortest paths in $\mathcal{S}$. By caching the shortest distance for each PRM vertex, computing $C_{fo}(s)$ runs in logarithmic time using a K-D tree to lookup the vertex closest to s.

This PRM-based potential field assumes that velocities can be chosen in any direction and with unit cost, but can be adapted to handle differentially-constrained systems using other sample-based motion planners. If actions are stochastic, a potential based on the Stochastic Motion Roadmap [1] might yield better results. We leave such extensions to future work.

4 Randomized Belief Space Replanning

The replanning algorithm used by RBSR grows a *belief tree* T whose nodes are belief states $b \in \mathcal{B}$, and the edges store open-loop actions.

```
Randomized Belief Space Replanning
Input: Current belief state $b_0$, current plan $a_1, \ldots, a_t$.
1. Initialize $T$ with the belief states in the plan starting from $b_0$.
2. For $i = 1, \ldots, N$, do:
3.     Pick a node $b$ in $T$ and an action $a$.
4.     If $b' = \mathcal{T}(b, a)$ is feasible, add $b'$ to $T$ as a child of $b$.
5. End
6. Sort the nodes in $T$ in order of decreasing $EIG(b)$.
7. For the $M$ best nodes in $T$, evaluate $f(b)$.
8. Return the plan leading to the node with the highest $f(b)$.
```

In Line 3 we use a Voronoi exploration strategy to quickly distribute nodes across belief space. Lines 6–7 are used to avoid evaluating f on all nodes on the tree, because it is an relatively expensive operation. We use an *expected information gain* score $EIG(b)$ to restrict the evaluations of f to a small subset of nodes $M << N$. Because $EIG(b)$ is less expensive than f to compute, this strategy leads to major speed gains. These strategies are described in greater detail below.

4.1 Voronoi-Biased Exploration Strategy

The exploration strategy is designed to cover the space of reachable open-loop motions quickly, and we use a Voronoi-biasing strategy much like the Rapidly-Exploring Random Tree (RRT) motion planner [15]. To expand the tree, we sample a random target point s_{tgt} from the state space S, and sample a set of representative particles from all belief states in the tree $R = \{s | b \in T, s \sim b\}$. Then, we find the closest point s from R to s_{tgt}. We then find a control a action that brings s closer to s_{tgt}.

4.2 Expected Information Gain Scoring Strategy

We use an expected information gain strategy to avoid running expensive evaluations of f on belief states that are unlikely to yield improvements in f. The intuition is that information gain is a sort of proxy score for QMDP favorability because it measures the spread of a belief state distribution, and QMDP tends to succeed more when states are localized. We compute the expected information gain for a belief state b as follows. The information gain of the observation o is the Kullback-Leibler divergence between the posterior distribution $b_o \equiv Pr(s|o, b)$ and the prior $b \equiv Pr(s|b)$:

$$I(b_o||b) = \int_{s \in S} Pr(s|o, b) \log \frac{Pr(s|o, b)}{Pr(s|b)}. \qquad (5)$$

Given a particle representation of belief states b_o and b, we replace the distribution $Pr(s|b)$ using a kernel density estimator with Gaussian kernels centered on the particles in b, and approximate the integral by the weighted sum over the particles $s^{(i)}$ in b_o.

The expected information gain is simply the expectation of (5) over o:

$$EIG(b) = \int_{o \in O} Pr(o|b) I(b_o||b) \qquad (6)$$

To compute this, we compute the observation $o^{(i)} \equiv \mathcal{O}(s^{(i)})$ for each particle in b, perform particle filtering $b_o = Filter(b, o^{(i)})$, and then compute the weighted average of (5) over all particles. Although EIG is an $O(n^3)$

computation, in our experiments it is typically orders of magnitude faster than computing f, and this scoring stage leads to major speedups.

4.3 Complexity and Convergence

The time complexity of RBSR depends on several parameters: the number of belief space particles n, the number of holdout particles for QMDP simulation m, the number of exploration steps N, and the number of nodes retained for QMDP evaluation M. Assume that an evaluation of $\pi_{\text{QMDP}}(b)$ (4) takes time $O(n)$. Then, the exploration stage takes time $O(nN^2)$, the EIG scoring takes time $O(n^3N)$, and the evaluation stage takes time $O(TnmM)$ where T is the average number of steps taken by QMDP before it terminates. But the running time of the evaluation stage hides a higher constant factor because it uses more expensive operations such as state and path collision checking, and in our experiments it dominates running time. Space complexity is $O((n+m)N)$.

The parameter n affects how accurately RBSP tracks and predicts belief states using the particle filter, and should be set high enough to attain a desired accuracy. In our experiments we do a small amount of tuning to find a reasonable parameter. m affects how accurately RBSP predicts the success rate of the QMDP policy, and $f(b)$ may be quite noisy for low m. Specifically, the variance of the success rate estimate p is bounded by $p(1-p)/m$, and m should be chosen to achieve a desired accuracy. Parameters N and M affect the chance that RBSR makes progress toward the goal in a single time step. We used parameters $N = 100$ and $M = 10$ in our experiments.

5 Experimental Results

We performed experiments on two scenarios: a 2D pursuit scenario with a 4D state space, as well as a localization scenario that has tunable dimension. Although these problems are not difficult to solve using special-purpose strategies, they pose a challenge for general-purpose planners to solve in a

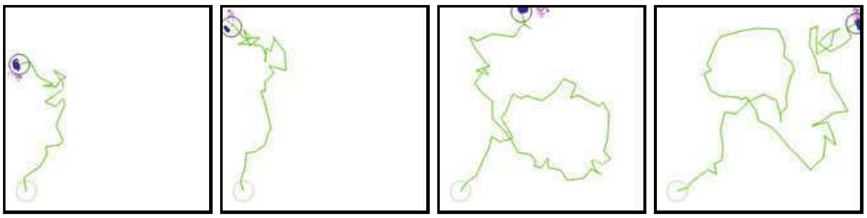


Fig. 4 Execution traces of the pursuit example for four different initial target locations (purple circles). The robot uses a distance sensor with maximum range 0.25.

reasonable amount of time and memory. For example, the SARSOP planner [14] can approximately solve a coarsely discretized version of the pursuit scenario in a few minutes, but it exhausts our test machine's 2Gb of memory once the resolution of the workspace grid exceeds 15x15.

5.1 Pursuit Scenario

Our first set of experiments consider a pursuit scenario in the unit square where the robot must reach a slower target that moves at random (Figure 1). The position of the robot is observable and controlled precisely, but it cannot sense the target outside a circle of radius 0.25. The target's position is a uniform distribution in the initial belief state, and the goal condition is to

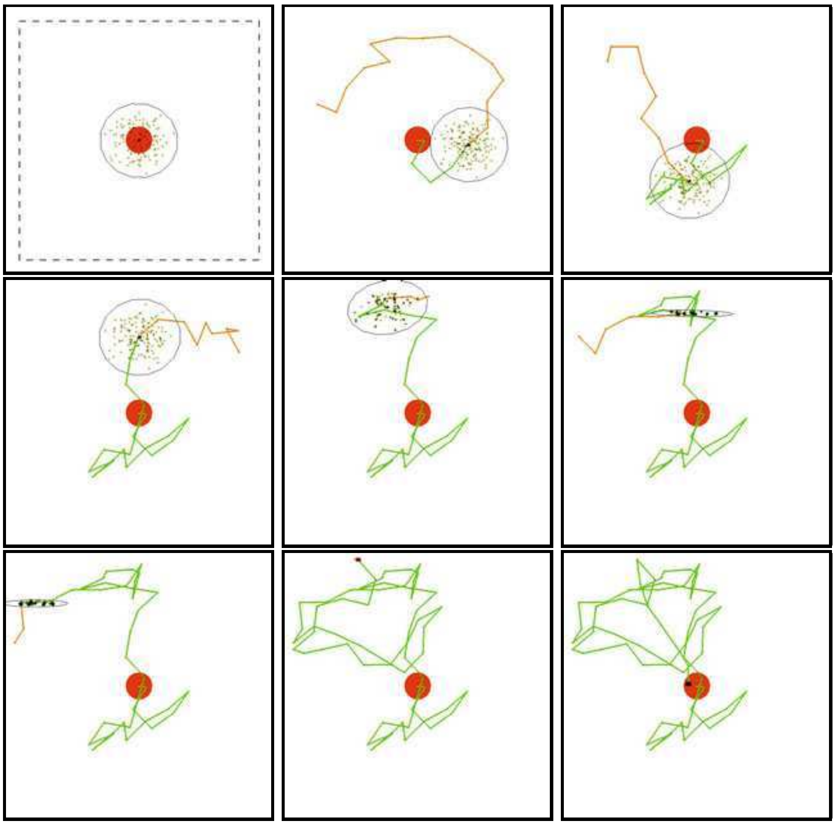


Fig. 5 An execution trace of a robot localizing itself to reach the red circle with high probability. Its sensor measures the distance to the walls, and has maximum range 0.05 (dashed lines). The current belief state is represented by 100 particles (dots) with a covariance ellipsoid, and the current plan (orange) is updated by replanning.

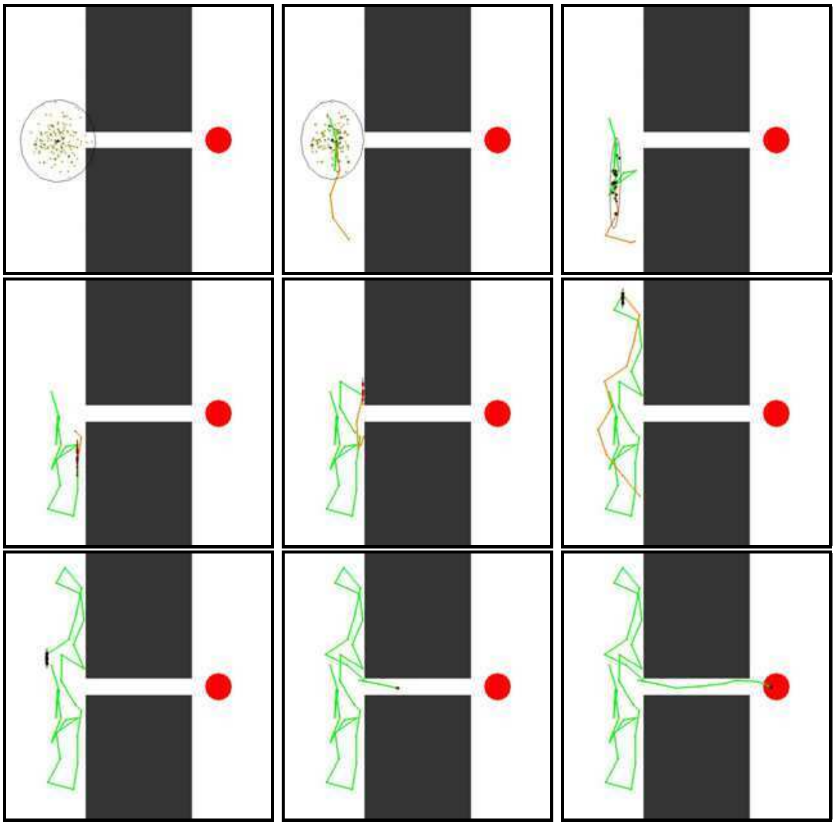


Fig. 6 A robot localizing itself using a proximity sensor in a space with obstacles.

achieve a distance of 0.05 to the target. We tested three sensor models: 1) a position sensor that reports the target's x, y position relative to the robot, 2) a direction sensor that reports only direction and not distance, and 3) a distance sensor that does not report direction.

Using preliminary experiments we tuned the number of particles in the belief state needed for accurate particle filtering, and found that 100 particles were sufficient for the position and direction sensor, and 200 particles were needed for the proximity sensor. So, we used $m = 50$ particles as a holdout set, and $n = 150$ and $n = 250$, respectively, for the position/direction sensors and the proximity sensor. In 25 trials on each of these problems, with random target start states, RBSP never failed to reach the target. Several execution traces for different initial target positions are drawn in Figure 4. Average path length is approximately 1.7, which is close to the expected path length computed by SARSOP on a 15 x 15 grid, but is still suboptimal. Each replanning iteration took about 15 s on average, with standard deviation $\tilde{1}0$ s.

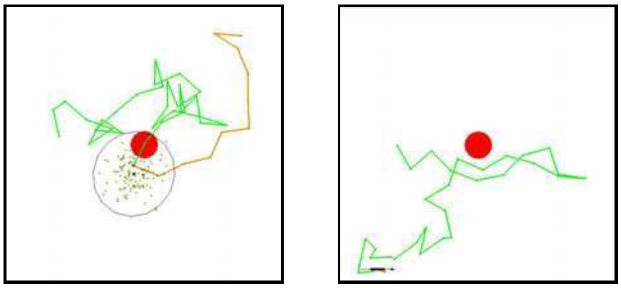


Fig. 7 Left: a partial localization execution using only 5 holdout particles. Because the evaluation function is noisy, the plan is often drastically revised and the walls have not yet been sensed after 30 steps. Right: by initiating replanning only when information gain exceeds a threshold, the path is smoother and two walls have been sensed within 30 steps.

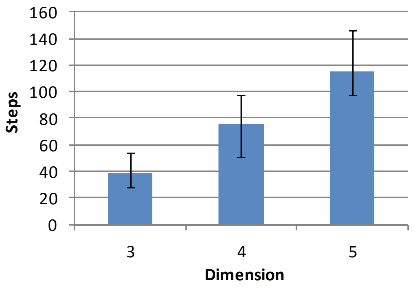


Fig. 8 In localization problems up to 5 dimensions the number of replanning steps scales roughly linearly. Columns report average, minimum, and maximum steps over 10 trials.

5.2 Localization Scenario

In our second scenario a robot is in an unknown configuration in a known d-dimensional environment and must localize itself and reach a small goal by measuring the distance to obstacles. The sensor has a limited range, which requires that the robot perform several steps of active sensing before reaching the goal. The optimal strategy is to proceed toward a wall until the sensor returns a reading, and then proceed to an adjacent wall until a closer reading is obtained, and so on until it achieves d readings from d linearly independent walls. Note that RBSR does not have a "proceed until" action in its action set, so instead it must approximate such a policy by a sequence of conditional movement actions and sensing actions.

In the first experiment (Figure 5), we set $d = 2$, $\mathcal{S}$ is the unit square, the initial belief state is a circular Gaussian distribution with standard deviation 0.1, and the goal radius and the sensing radius are both set to 0.05. To represent belief states we used 150 particles with a holdout set of 50. We also tested a space containing obstacles (Figure 6). In both examples, RBSR performs localization by moving close to obstacle boundaries, in somewhat random fashion, until it senses nearby walls. This continues until sufficient data is gathered to reach the goal.

We tested the effects of reducing the size of the holdout set m, and what we found was that the resulting executions tend to be much more noisy due to spurious noise in $f(b)$. In such cases, we found better results when replanning is initiated only when the current belief state experiences a large information gain due to an incoming observation (Figure 7). We hope to explore this strategy further in future work.

Our final set of experiments tested scalability with respect to dimension. Figure 8 plots the number of replanning steps taken by RBSR in problems from $d = 3$ to $d = 5$ in the unit hypercube. We increased the number of particles to 500, but kept all other parameters unchanged from the experiment in Figure 5. These experiments suggest that the number of replanning steps is roughly linear in dimension. Running time per timestep is roughly linear in dimension as well, ranging from approximately 6 s in the 3D case up to approximately 14 s in the 5D case. In higher dimensions, the accuracy of the particle filter dropped off sharply. In future work we hope to explore more sophisticated belief state representations, like Gaussian mixture models, that can maintain accuracy with a manageable number of particles.

6 Discussion: Exploration Strategies

The experiments above are preliminary but promising, and in future work we would like to study RBSR's theoretical performance in the face of approximate belief states and randomization in the exploration strategy. In this section we argue why we expect that RBSR will work well in a broader class of problems; particularly those in which 1) random walks in belief space have a significant probability of finding useful information, and 2) in the process of information-gathering, uncertainty is not significantly increased.

Under these assumptions, RBSR is roughly a belief-space analogue to the Randomized Path Planner (RPP) [2], which addresses path planning in a deterministic, fully-observable environment by alternating potential field descent with random walks to escape local minima. RBSR is, however, better than RPP for two reasons: 1) it perform many walks in simulation only and then picks the best one for execution, and 2) it performs many walks in parallel using the Voronoi bias heuristic, which is more efficient at exploring belief space than a random walk. So, we should be able to show that RBSR performs at least as well as RPP, which is probabilistically complete.

Another interpretation is that RBSR uses *macro-actions* to make planning more efficient. The idea of macro-actions have existed for some time in the discrete POMDP literature as a way to reduce the exploration breadth and depth in large robotics problems [16]. For example, Hsiao et. al. addressed a robot grasping problem using specially constructed macro-actions that either provide information or seek the goal [9]. They demonstrate that if uncertainty grows slowly during information-gathering, then forward planning can be limited to depth one. RBSR can also be interpreted as depth-one forward planning, using the QMDP policy as a goal-seeking macro-action and belief-space sampling to produce information-gathering macro-actions on the fly. Two other recent works have also tackled the problem of constructing macro-actions automatically and with increasing granularity during forward planning [8, 13]. These approaches are limited to macro-actions that reach various states as subgoals, and we suspect that RBSR constructs better information-gathering macro-actions using belief space criteria; on the other hand we also suspect that the approaches in [8, 13] construct more optimal plans by searching to a greater depth. (Note that our current presentation of RBSR does not incorporate action costs; future implementations may incorporate path cost during the selection of information-gathering paths.) It remains an open question whether these varied approaches will yield problem-independent principles for generating and exploiting macro-actions in both discrete and continuous POMDPs.

7 Conclusion

This paper presented preliminary work in a Randomized Belief-Space Replanning (RBSR) technique for partially-observable problems in continuous state spaces. It constructs partial plans by sampling open-loop actions at random, and by evaluating the quality of future belief states by simulating a QMDP-like policy that performs well when the state is well-localized. By iteratively incorporating sensor feedback from plan execution and replanning, RBSR avoids having to compute a policy over large belief spaces. Experiments show that it solves a target pursuit problem with a 4D state space and a localization problem in 2D–5D state spaces relatively efficiently.

Future work should attempt to formally characterize convergence rates of RBSR and perform experimental comparisons against established techniques for discrete POMDPs. Future benchmark development for partially observable continuous problems would aid the empirical study of planner sensitivity to dimensionality and other belief space properties. We also intend to address improving path optimality and using sensing more efficiently in the RBSR framework, because randomization yields somewhat jerky plans. With additional refinements, RBSR-like approaches may lead to breakthroughs in planning under partial observability in realistic robotic systems.

References

1. Alterovitz, R., Simeon, T., Goldberg, K.: The stochastic motion roadmap: A sampling framework for planning with markov motion uncertainty. In: Robotics: Science and Systems (June 2007)

2. Barraquand, J., Latombe, J.-C.: Robot motion planning: A distributed representation approach. Int. J. Rob. Res. 10(6), 628–649 (1991)
3. Berg, J.V.D., Abbeel, P., Goldberg, K.: Lqg-mp: Optimized path planning for robots with motion uncertainty and imperfect state information. In: Proc. Robotics: Science and Systems (2010)
4. Burns, B., Brock, O.: Sampling-based motion planning with sensing uncertainty. In: Proc. IEEE Int. Conf. on Robotics and Automation (2007)
5. Doucet, A., Godsill, S., Andrieu, C.: On sequential monte carlo sampling methods for bayesian filtering. Statistics and Computing 10(3), 197–208 (2000)
6. du Toit, N., Burdick, J.: Robotic motion planning in dynamic, cluttered, uncertain environments. In: IEEE Int. Conf. on Robotics and Automation (2010)
7. Guibas, L.J., Hsu, D., Kurniawati, H., Rehman, E.: Bounded uncertainty roadmaps for path planning. In: Workshop on the Algorithmic Foundations of Robotics, Guanajuato, Mexico (2008)
8. He, R., Brunskill, E., Roy, N.: Puma: Planning under uncertainty with macro-actions. In: Proc. Twenty-Fourth Conf. on Artificial Intelligence (AAAI) (2010)
9. Hsiao, K., Lozano-Perez, T., Kaelbling, L.P.: Robust belief-based execution of manipulation programs. In: Eighth International Workshop on the Algorithmic Foundations of Robotics (WAFR) (2008)
10. Huang, Y., Gupta, K.: Collision-probability constrained prm for a manipulator with base pose uncertainty. In: Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, Piscataway, NJ, USA, 2009, pp. 1426–1432. IEEE Press, Los Alamitos (2009)
11. Kavraki, L.E., Svetska, P., Latombe, J.-C., Overmars, M.: Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Trans. Robot. and Autom. 12(4), 566–580 (1996)
12. Kearns, M., Mansour, Y., Ng, A.Y.: Approximate planning in large pomdps via reusable trajectories. In: Advances in Neural Information Processing Systems, vol. 12. MIT Press, Cambridge (2000)
13. Kurniawati, H., Du, Y., Hsu, D., Lee, W.: Motion planning under uncertainty for robotic tasks with long time horizons. In: Proc. Int. Symp. on Robotics Research (2009)
14. Kurniawati, H., Hsu, D., Lee, W.: Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces. In: Proc. Robotics: Science and Systems (2008)
15. LaValle, S.M., Kuffner Jr., J.J.: Rapidly-exploring random trees: progress and prospects. In: WAFR (2000)
16. Littman, M., Cassandra, A.R., Kaelbling, L.P.: Learning policies for partially observable environments: Scaling up. In: Proc. 12th Int. Conf. on Machine Learning, pp. 362–370. Morgan Kaufmann, San Francisco (1995)
17. Littman, M.L., Goldsmith, J., Mundhenk, M.: The computational complexity of probabilistic planning. Journal of Artificial Intelligence Research 9, 1–36 (1998)
18. Pineau, J., Gordon, G., Thrun, S.: Point-based value iteration: An anytime algorithm for pomdps. In: International Joint Conference on Artificial Intelligence (IJCAI), Acapulco, Mexico, pp. 1025–1032 (August 2003)
19. Platt, R., Tedrake, R., Kaelbling, L., Lozano-Perez, T.: Belief space planning assuming maximum likelihood observations. In: Proc. Robotics: Science and Systems (2010)

20. Porta, J.M., Vlassis, N., Spaan, M.T.J., Poupart, P.: Point-based value iteration for continuous pomdps. Journal of Machine Learning Research 7, 2329–2367 (2006)
21. Prentice, S., Roy, N.: The Belief Roadmap: Efficient Planning in Belief Space by Factoring the Covariance. The International Journal of Robotics Research 28(11-12), 1448–1465 (2009)
22. Thrun, S.: Monte carlo pomdps. In: Advances in Neural Information Processing Systems 12 (NIPS-1999), MIT Press, Cambridge (2000)

GPU-Based Parallel Collision Detection for Real-Time Motion Planning

Jia Pan and Dinesh Manocha

Abstract. We present parallel algorithms to accelerate collision queries for sample-based motion planning. Our approach is designed for current many-core GPUs and exploits the data-parallelism and multi-threaded capabilities. In order to take advantage of high number of cores, we present a clustering scheme and collision-packet traversal to perform efficient collision queries on multiple configurations simultaneously. Furthermore, we present a hierarchical traversal scheme that performs workload balancing for high parallel efficiency. We have implemented our algorithms on commodity NVIDIA GPUs using CUDA and can perform $500,000$ collision queries/second on our benchmarks, which is 10X faster than prior GPU-based techniques. Moreover, we can compute collision-free paths for rigid and articulated models in less than 100 milliseconds for many benchmarks, almost 50-100X faster than current CPU-based planners.

1 Introduction

Motion planning is one of the fundamental problems in algorithmic robotics. The goal is to compute collision-free paths for robots in complex environments. Some of the widely used algorithms for high-DOF (degree-of-freedom) robots are based on randomized sampling. These include algorithms based on PRMs [12] and RRTs [14]. These methods tend to capture the topology of the free configuration space of the robot by generating a high number of random configurations and connecting nearby collision-free configurations (i.e. milestones) using local planning methods. The resulting algorithms are probabilistically complete and have been successfully used to solve many challenging motion planning problems.

In this paper, we address the problem of designing fast and almost real-time planning algorithms for rigid and articulated models. The need for such algorithms arises

Jia Pan · Dinesh Manocha
University of North Carolina, Chapel Hill, NC, USA
e-mail: {panj,dm}@cs.unc.edu
http://gamma.cs.unc.edu/gplanner

not only from virtual prototyping and character animation, but also task planning for physical robots. Current robots (such as Willow Garage's PR2) tend to use live sensor data to generate a reasonably accurate model of the objects in the physical world. Some tasks, such as robot navigation or grasping, need to compute a collision-free path for the manipulator in real-time to handle dynamic environments. Moreover, many high-level task planning algorithms perform motion planning and subtask execution in an interleaved manner, i.e. the planning result of one subtask is used to construct the formulation of the following subtask [27]. A fast and almost real-time planning algorithm is important for these applications.

It is well known that a significant fraction (e.g. 90% or more) of randomized sampling algorithms is spent in collision checking. This includes checking whether a given configuration is in free-space or not as well as connecting two free-space configurations using a local planning algorithm. While there is extensive literature on fast intersection detection algorithms, some of the recent planning algorithms are exploiting the computational power and massive parallelism of commodity GPUs (graphics processing units) for almost real-time computation [23, 22]. Current GPUs are high-throughput many-core processors, which offer high data-parallelism and can simultaneously execute a high number of threads. However, they have a different programming model and memory hierarchy as compared to CPUs. As a result, we need to design appropriate parallel collision and planning algorithms that can map well to GPUs.

Main Results. We present a novel, parallel algorithm to perform collision queries for sample-based motion planning. Our approach exploits parallelism at two levels: it checks multiple configurations simultaneously whether they are in free space or not and performs parallel hierarchy traversal for each collision query. Similar techniques are also used for local planning queries. We present clustering techniques to appropriately allocate the collision queries to different cores, Furthermore, we introduce the notion of *collision-packet traversal*, which ensures that all the configurations allocated to a specific core result in similar hierarchical traversal patterns. The resulting approach also exploits fine-grained parallelism corresponding to each bounding volume overlap test to balance the workload.

The resulting algorithms have been implemented on commodity NVIDIA GPUs. In practice, we are able to process about $500,000$ collision queries per second on a $400 NVIDIA GeForce 480 desktop GPU, which is almost 10X faster than prior GPU-based collision checking algorithms. We also apply our collision checking algorithm for GPU-based motion planners to high DOF rigid and articulated robots. The resulting planner can compute collision-free paths in less than 100 milliseconds for various benchmarks and appears to be 50-100X faster than CPU-based planners.

The rest of the paper is organized as follows. We survey related work on real-time motion planning and parallel collision-checking algorithms in Section 2. Section 3 gives an overview of our approach and we present our new parallel algorithm for parallel collision queries in Section 4. We highlight the performance of our algorithm on different benchmarks in Section 5.

2 Previous Work

In this section, we give a brief overview of prior work in real-time motion planning and parallel algorithms for collision detection.

2.1 Real-Time Motion Planning

An excellent survey of various motion planning algorithms is given in [17]. Many parallel algorithms have also been proposed for motion planning by utilizing the properties of configuration spaces [20]. The distributed representation [5] can be easily parallelized. In order to deal with very high dimensional or difficult planning problems, distributed sampling-based techniques have been proposed [25].

The computational power of many-core GPUs has been used for many geometric and scientific computations [21]. The rasterization capabilities of a GPU can be used for real-time motion planning of low DOF robots [10, 26] or improve the sample generation in narrow passages [24, 7]. Recently, GPU-based parallel motion planning algorithms have been proposed for rigid models [23, 22].

2.2 Parallel Collision Queries

Some of the widely used algorithms for collision query are based on *bounding volume hierarchies* (BVH), such as *k*-DOP trees, OBB trees, AABB trees, etc [18]. Recent developments include parallel hierarchical computations on multi-core CPUs [13, 28] and GPUs [16]. CPU-based approaches tend to rely on fine-grained communication between processors, which is not suited for current GPU-like architectures. On the other hand, GPU-based algorithms [16] use work queues to parallelize the computation on the multiple cores. All these approaches are primarily designed to parallelize a single collision query for sample-based motion planning.

The capability to perform a high number of collision queries efficiently is essential in motion planning algorithms, e.g. for parallel collision queries in milestone computation and local planning. Some of the prior algorithms perform parallel queries in a simple manner: each thread handles a single collision query in an independent manner [23, 22, 3, 2]. As current multi-core CPUs have the capability to perform *multiple-instruction multiple-data* (MIMD) computations, these simple strategies can work well on CPUs. On the other hand, current GPUs offer high data parallelism and the ability to execute a high number of threads in parallel to overcome the high memory latency. As a result, we need different parallel collision detection algorithms to fully exploit their capabilities.

3 Overview

In this section, we first provide some background on current GPU architectures. Next, we address some issues in designing efficient parallel algorithms to perform collision queries.

3.1 GPU Architectures

In recent years, the focus in processor architectures has shifted from increasing clock rate to increasing parallelism. Commodity GPUs such as NVIDIA Fermi[1] have theoretical peak performance of Tera-FLOP/s for single precision computation and hundreds of Giga-FLOP/s for double precision computations. This peak performance is significantly higher as compared to current multi-core CPUs, thus outpacing CPU architectures [19] at relatively modest cost of $300 to $400. However, GPUs have different architectural characteristics and memory hierarchy, that impose some constraints in terms of designing appropriate algorithms. First, GPUs usually have a high number of independent cores (e.g. the newest generation GTX 480 has 15 cores and each core has 32 streaming processors resulting in total of 480 processors while GTX 280 has only 240 processors). Each of the individual cores is a vector processor capable of performing the same operation on several elements simultaneously (e.g. 32 elements for current GPUs). Secondly, the memory hierarchy on GPUs is quite different from that of CPUs and cache sizes on the GPUs are considerably smaller. Moreover, each GPU core can handle several separate tasks in parallel and switch between them in the hardware when one of them is waiting for a memory operation to complete. This hardware multithreading approach is thus designed to hide the memory access latency. Thirdly, all GPU threads are logically grouped in blocks with a per-block shared memory, which provides a weak synchronization capability between the GPU cores. Overall, shared memory is a limited resource on GPUs: increasing the shared memory distributed for each thread can limit the extent of parallelism. Finally, the threads are physically processed in chunks in SIMT (single-instruction, multiple-thread). This is different from SIMD (single-instruction multiple-data) and each thread can execute independent instructions. The GPU's performance can reduce significantly when threads in the same chunk diverge considerably, because these diverging portions are executed in a serial manner for all the branches. As a result, threads with coherent branching decisions (e.g. threads traversing the same paths in the BVH) are preferred on GPUs in order to obtain higher performance [8]. All of these characteristics imply that – unlike CPUs – achieving high performance in current GPUs depends on several factors: (1) generating a sufficient number of parallel tasks so that all the cores are highly utilized; (2) developing parallel algorithms such that the total number of threads is even higher than the number of tasks, so that each core has enough work to perform while waiting for data from relatively slow memory accesses; (3) assigning appropriate size for shared memory to accelerate memory accesses and not reduce

[1] `http://www.nvidia.com/object/fermi_architecture.html`

the level of parallelism; (4) performing coherent or similar branching decisions for each parallel thread within a given chunk. These requirements impose constraints in terms of designing appropriate collision query algorithms.

3.2 Notation and Terminology

We define some terms and highlight the symbols used in the rest of the paper.

chunk. The minimum number of threads that GPUs manage, schedule and execute in parallel, which is also called *warp* in the GPU computing literatures. The size of chunk (*chunk-size* or *warp-size*) is 32 on current NVIDIA GPUs (e.g. GTX 280 and 480).

block. The collection of GPU threads that will be executed on the same GPU core. These threads synchronize by using barriers and communicate via a small high-speed low-latency *shared memory*.

BVH_a. The *bounding volume hierarchy* (BVH) tree for model a. It is a binary tree with L levels, whose nodes are ordered in the breadth-first order starting from the root node. Each node is denoted as $\mathrm{BVH}_a[i]$ and its children nodes are $\mathrm{BVH}_a[2i]$ and $\mathrm{BVH}_a[2i+1]$ with $1 \leq i \leq 2^{L-1}-1$. The nodes at the l-th level of a BVH tree are represented as $\mathrm{BVH}_a[k], 2^l \leq k \leq 2^{l+1}-1$ with $0 \leq l < L$. The inner nodes are also called *bounding volumes* (BV) and the leaf nodes also have a link to the primitive triangles that are used to represent the model.

$\mathrm{BVTT}_{a,b}$. The *bounding volume test tree* (BVTT) represents recursive collision query traversal between two objects a, b. It is a 4-ary tree, whose nodes are ordered in the breadth-first order starting from the root node. Each node is denoted as $\mathrm{BVTT}_{a,b}[i] \equiv (\mathrm{BVH}_a[m], \mathrm{BVH}_b[n])$ or simply (m,n), which checks the BV or primitive overlap between nodes $\mathrm{BVH}_a[m]$ and $\mathrm{BVH}_b[n]$, while $m = \lfloor i - \frac{4^M+2}{3} \rfloor + 2^M$, $n = \{i - \frac{4^M+2}{3}\} + 2^M$ and $M = \lfloor log_4(3i-2) \rfloor$. BVTT node (m,n)'s children are $(2m, 2n)$, $(2m, 2n+1)$, $(2m+1, 2n)$, $(2m+1, 2n+1)$.

$\mathbf{q}$. A configuration of the robot, which is randomly sampled within the configuration space $\mathscr{C}$-Space. $\mathbf{q}$ is associated with the transformation $\mathbf{T_q}$. The BVH of a model a after applying such a transformation is given as $\mathrm{BVH}_a(\mathbf{q})$.

3.3 Collision Queries: Hierarchical Traversal

Collision queries between the geometric models are usually accelerated with hierarchical techniques based on BVHs, which correspond to traversing the BVTT related with the BVHs of the models [15]. The simplest parallel algorithms to perform multiple collision queries are based on each thread traversing the BVTT and checking whether a given configuration is in free space or not. Such a simple parallel algorithm is highlighted in Algorithm 1. This strategy is easy to implement and has been used in previous parallel planning algorithms based on multi-core or multiple CPUs. But it may not result in high parallel efficiency on current GPUs due to the following reasons. First, each thread needs a local traverse stack on the shared memory which may not be effective for complex models with thousands of polygons. Second, different threads may traverse the BVTT tree with incoherent patterns: there are many branching decisions performed during the traversal (e.g. **loop**, **if**, **return** in the pseudo-code) and the traversal flow of the hierarchy in different threads diverges quickly. Finally, different threads can have varying workloads; some may be busy with the traversal while the others may have finished the traversal early due to no overlap and are idle. These factors can affect the performance of the parallel algorithm.

The problems of low parallel efficiency in Algorithm 1 become more severe in complex or articulated models. For such models, there are longer traversal paths in the hierarchy and the difference between these paths can be large for different configurations. As a result, differences in the workloads between different threads can be high. For articulated models, each thread checks the collision status of all the links and stops when a collision is detected for any link. Therefore, more branching decisions are performed within each thread and this can lead to more incoherence. Similar issues also arise during local planning when each thread determines whether two milestones can be joined by a collision-free path by checking the collisions along the trajectory connecting them.

Algorithm 1. Simple parallel collision checking; Such approaches are frequently used on multi-core CPUs

```
1: Input: $N$ random configurations $\{\mathbf{q}_i\}_{i=1}^N$, $\mathrm{BVH}_a$ for the robot and $\mathrm{BVH}_b$ for the obstacles
2: Output: return whether one configuration is in free space or not
3: $t_{id} \leftarrow$ thread id of current thread
4: $\mathbf{q} \leftarrow \mathbf{q}_{t_{id}}$
5: $\lhd$ traverse stack $S[]$ is initialized with root nodes
6: shared $S[] \equiv$ local traversal stack
7: $S[] \leftarrow \mathrm{BVTT}[1] \equiv (\mathrm{BVH}_a(\mathbf{q})[1], \mathrm{BVH}_b[1])$
8: $\lhd$ traverse BVTT for $\mathrm{BVH}_a(\mathbf{q})$ and $\mathrm{BVH}_b$
9: loop
10:     $(x,y) \leftarrow pop(S)$.
11:     if $overlap(\mathrm{BVH}_a(\mathbf{q})[x], \mathrm{BVH}_b[y])$ then
12:         $S[] \leftarrow (2x,2y),(2x,2y+1),(2x+1,2y),(2x+1,2y+1)$ if $!isLeaf(x)$ && $!isLeaf(y)$
13:         $S[] \leftarrow (2x,2y),(2x,2y+1)$ if $isLeaf(x)$ && $!isLeaf(y)$
14:         $S[] \leftarrow (2x,2y),(2x+1,2y)$ if $!isLeaf(x)$ && $isLeaf(y)$
15:         return $collision$ if $isLeaf(x)$ && $isLeaf(y)$ && $exactIntersect(\mathrm{BVH}_a(\mathbf{q})[x], \mathrm{BVH}_b[y])$
16:     end if
17: end loop
18: return $collision\text{-}free$
```

4 Parallel Collision Detection on GPUs

In this section, we present two novel algorithms for efficient parallel collision checking on GPUs between rigid or articulated models. Our methods can be used to check whether a configuration lies in the free space or to perform local planning computations. The first algorithm uses clustering and fine-grained packet-traversal to improve the coherence of BVTT traversal for different threads. The second algorithm uses queue-based techniques and lightweight workload balancing to achieve higher parallel performance on the GPUs. In practice, the first method can provide 30%-50% speed up. Moreover, it preserves the per-thread per-query structure of the naive parallel strategy. Therefore, it is easy to implement and is suitable for cases where we need to perform some additional computations (e.g. retraction for handling narrow passages [29]). The second method can provide 5-10X speed up, but is relatively more complex to implement.

4.1 Parallel Collision-Packet Traversal

Our goal is to ensure that all the threads in a block performing BVTT-based collision checking have similar workloads and coherent branching patterns. This approach is motivated by recent developments related to interactive ray-tracing on GPUs for visual rendering. Each collision query traverses the BVTT and performs node-node or primitive-primitive intersection tests. In contrast, ray-tracing algorithms traverse the BVH tree and perform ray-node or ray-primitive intersections. Therefore, parallel ray-tracing algorithms on GPUs also need to avoid incoherent branches and varying workloads to achieve higher performance.

In real-time ray tracing, one approach to handle the varying workloads and incoherent branches is the use of ray-packets [8, 1]. In ray-tracing terminology, packet traversal implies that a group of rays follows exactly the same traversal path in the hierarchy. This is achieved by sharing the traversal stack (similar to the BVTT traversal stack in Algorithm 1) among the rays in the same warp-sized packet (i.e. threads that fit in one chunk on the GPU), instead of each thread using an independent stack for a single ray. This implies that the same additional nodes in the hierarchy may be visited during ray intersection tests, even though there are no intersections between the rays and those nodes. But the resulting traversal is coherent for different rays, because each node is fetched only once per packet. In order to reduce the number of computations (i.e. unnecessary node intersection tests), all the rays in one packet should be similar to one another, i.e. have similar traversal paths with few differing branches. We extend this idea to parallel collision checking and refer to our algorithm as *multiple configuration-packet* method.

The first challenge is to cluster similar collision queries or the configurations into groups. In some cases, the sampling scheme (e.g. the adaptive sampling for lazy PRM) can provide natural group partitions. However, in most cases we need suitable algorithms to compute these clusters. Clustering algorithms are natural choices for such a task, which aims at partitioning a set $\mathscr{X}$ of N data items $\{\mathbf{x}_i\}_{i=1}^N$ into K groups $\{C_k\}_{k=1}^K$ such that the data items belonging to the same group are more "similar" than the data items in different groups. The clustering algorithm used to group the configurations needs to satisfy some additional constraints: $|C_k| = \textit{chunk-size}, 1 \leq k \leq K$, i.e. each cluster should fit in one chunk on GPUs, except for the last cluster and $K = \lceil \frac{N}{\textit{chunk-size}} \rceil$. Using the formulation of k-means, the clustering problem can be formally described as: compute $K = \lceil \frac{N}{\textit{chunk-size}} \rceil$ items $\{\mathbf{c}_k\}_{k=1}^K$ that minimizes

$$\sum_{i=1}^{N} \sum_{k=1}^{K} \mathbf{1}_{\mathbf{x}_i \in C_k} \|\mathbf{x}_i - \mathbf{c}_k\|, \tag{1}$$

with constraints $|C_k| = \textit{chunk-size}, 1 \leq k \leq K$. To our knowledge, there are no clustering algorithms designed for this specific problem. One possible solution is *clustering with balancing constraints* [4], which has additional constraints $|C_k| \geq m, 1 \leq k \leq K$, where $m \leq \frac{N}{K}$.

Instead of solving Equation (1) exactly, we use a simpler clustering scheme to compute an approximate solution. First, we use k-means algorithm to cluster the N queries into C clusters, which can be implemented efficiently on GPUs [6]. Next, for k-th cluster of size S_k, we divide it into $\lceil \frac{S_k}{\textit{chunk-size}} \rceil$ sub-clusters, each of which corresponds to a *configuration-packet*. This simple method has some disadvantages. For example, the number of clusters is $\sum_{k=1}^{C} \lceil \frac{S_k}{\textit{chunk-size}} \rceil \geq K = \lceil \frac{N}{\textit{chunk-size}} \rceil$ and therefore Equation (1) may not result in an optimal solution. However, as shown later, even this simple method can improve the performance of parallel collision queries.

Next we map each configuration-packet to a single chunk. Threads within one packet will traverse the BVTT synchronously, i.e. the algorithm works on one BVTT node (x,y) at a time and processes the whole packet against the node. If (x,y) is a leaf node, an exact intersection test is performed for each thread. Otherwise, the algorithm loads its children nodes and tests the BVs for overlap to determine the remaining traversal order, i.e. to select one child (x_m, y_m) as the next BVTT node to be traversed for the entire packet. We select (x_m, y_m) in a greedy manner: it corresponds to the child node that is classified as overlapping by the most threads in the packet. We also push other children into the packet's traversal stack. In case no BV overlap is detected in all the threads or (x,y) is a leaf node, (x_m, y_m) would be the top element in the packet's traversal stack. The traversal step is repeated recursively, until the stack is empty. Compared to Algorithm 1, all the threads in one chunk share one traversal stack in shared memory, instead of using one stack for each thread. Therefore, the size of shared memory used is reduced by *chunk-size* times and results in higher parallel efficiency.

The traversal order described above is a greedy heuristic that tries to minimize the traversal path of the entire packet. For one BVTT node (x,y), if the overlap is not detected in any of the threads, it implies that these threads will not traverse the sub-tree rooted at (x,y). Since all the threads in the packet are similar and traverse the BVTT in nearly identical order, this implies that other threads in the same packet might not traverse the sub-tree either. We define the probability that the sub-tree rooted at (x,y) will be traversed by one thread as $p_{x,y} = \frac{\#\textit{overlap threads}}{\textit{packet-size}}$. For any traverse pattern P for BVTT, the probability that it is carried on by BVTT traversal will be $p_P = \prod_{(x,y) \in P} p_{x,y}$. As a result, our new traversal strategy guarantees that the traversal pattern with higher traverse probability will have a shorter traversal length, and therefore minimizes the overall path for the packet.

The decision about which child node is the candidate for next traversal step is computed using sum reduction [9], which can compute the sum of n items in parallel with $O(\log(n))$ complexity. Each thread writes a 1 in its own location in the shared memory if it detects overlap in one child and 0 otherwise. The sum of the memory locations is computed in 5 steps for a size 32 chunk. The packet chooses the child node with the maximum sum. The complete algorithm for configuration-packet computation is described in Algorithm 2.

Algorithm 2. Multiple Configuration-Packet Traversal

```
 1: Input: $N$ random configurations $\{\mathbf{q}_i\}_{i=1}^N$, BVH$_a$ for the robot and BVH$_b$ for the obstacles
 2: $t_{id} \leftarrow$ thread id of current thread
 3: $\mathbf{q} \leftarrow \mathbf{q}_{t_{id}}$
 4: **shared** $CN[] \equiv$ shared memory for children node
 5: **shared** $TS[] \equiv$ local traversal stack
 6: **shared** $SM[] \equiv$ memory for sum reduction
 7: **return if** $overlap(\mathrm{BVH}_a(\mathbf{q})[1], \mathrm{BVH}_b[1])$ is **false** for all threads in chunk
 8: $(x,y) = (1,1)$
 9: **loop**
10:     **if** $isLeaf(x)$ && $isLeaf(y)$ **then**
11:         update collision status of **q** **if** $exactIntersect(\mathrm{BVH}_a(\mathbf{q})[x], \mathrm{BVH}_b[y])$
12:         **break**, **if** $TS$ is empty
13:         $(x,y) \leftarrow pop(TS)$
14:     **else**
15:         $\triangleleft$ decide the next node to be traversed
16:         $CN[] \leftarrow (x,y)$'s children nodes
17:         **for all** $(x_c, y_c) \in CN$ **do**
18:             $\triangleleft$ compute the number of threads that detect overlap at node $(x_c, y_c)$
19:             write $overlap(\mathrm{BVH}_a(\mathbf{q})[x_c], \mathrm{BVH}_b[y_c])$ (0 or 1) into $SM[t_{id}]$ accordingly
20:             compute local summation $s_c$ in parallel by all threads in chunk
21:         **end for**
22:         **if** $\max_c s_c > 0$ **then**
23:             $\triangleleft$ select the node that is overlapped in the most threads
24:             $(x,y) \leftarrow CN[\mathrm{argmax}_c\, s_c]$ and push others into $TS$
25:         **else**
26:             $\triangleleft$ select the node from the top of stack
27:             **break**, **if** $TS$ is empty
28:             $(x,y) \leftarrow pop(TS)$
29:         **end if**
30:     **end if**
31: **end loop**
```

4.2 Parallel Collision Query with Workload Balancing

Both Algorithm 1 and Algorithm 2 use the per-thread per-query strategy, which is easy to implement. However, when the idle threads wait for busy threads or when the execution path of threads diverges, the parallel efficiency on the GPUs is low. Algorithm 2 can reduce this problem in some cases, but it still distributes the tasks among the separate GPU cores and cannot make full use of the GPU's computational power.

In this section, we present the parallel collision query algorithm based on workload balancing which further improves the performance. In this algorithm, the task of each thread is no longer one complete collision query or continuous collision query (for local planning). Instead, each thread only performs BV overlap tests. In other words, the unit task for each thread is distributed in a more fine-grained manner. Basically, we formulate the problem of performing multiple collision queries as a pool of BV overlap tests which can be performed in parallel. It is easier to distribute these fine-grained tasks in a uniform manner onto all the GPU cores, and thereby balancing the load among them, than to distribute the collision query tasks.

All the tasks are stored in Q large work queues in the GPU's main memory, which has a higher latency compared to the shared memory. When computing a single collision query [16], the tasks are in the form of BVTT nodes (x,y). Each thread will fetch some tasks from one work queue into its local work queue on the shared memory and traverse the corresponding BVTT nodes. The children generated for each node are also pushed into the local queue as new tasks. This process is repeated for all the tasks remaining in the queue, until the number of threads with full or empty local work queues exceeds a given threshold (we use 50% in our implementation) and non-empty local queues are copied back to the work queues on main memory. Since each thread performs simple tasks with few branches, our algorithm can make full use of GPU cores if there are sufficient tasks in all the work queues. However, during BVTT traversal, the tasks are generated dynamically and thus different queues may have varying numbers of tasks and this can lead to an uneven workload among the GPU cores. We use a balancing algorithm that redistributes the tasks among work queues (Figure 2). Suppose the number of tasks in each work queue is $n_i, 1 \leq i \leq Q$. Whenever $\exists i$, $n_i < T_l$ or $n_i > T_u$, we execute our balancing algorithm among all the queues and the number of tasks in each queue becomes $n_i^* = \frac{\sum_{k=1}^{Q} n_k}{Q}, 1 \leq i \leq Q$, where T_l and T_u are two thresholds (we use *chunk-size* for T_l and the $W -$ *chunk-size* for T_u, where W is the maximum size of work queue).

In order to handle N collision queries simultaneously, we use several strategies, which are similar to the ones highlighted in Figure 1. First, we can repeat the single query above algorithm [16] for each query. However, this has two main disadvantages. First, the GPU kernel has to be called N times from the CPU, which is expensive for large N (which can be $\gg 10000$ for motion planning applications). Secondly, for each query, work queues are initialized with only one item (i.e. the root node of the BVTT), therefore the GPU's computational power cannot be fully exploited at the beginning of each query, as shown in the slow ascending part in Figure 1(a). Similarly, at the end of each query, most tasks are finished and some of the GPU cores become idle, which corresponds to the slow descending part in Figure 1(a).

As a result, we use the strategy shown in Figure 1(b): we divide the N queries into $\lceil \frac{N}{M} \rceil$ different sets each of size M with $M \leq N$ and initialize the work queues with M different BVTT roots for each iteration. Usually M cannot be N because we need to use $t \cdot M$ GPU global memory to store the transform information for the

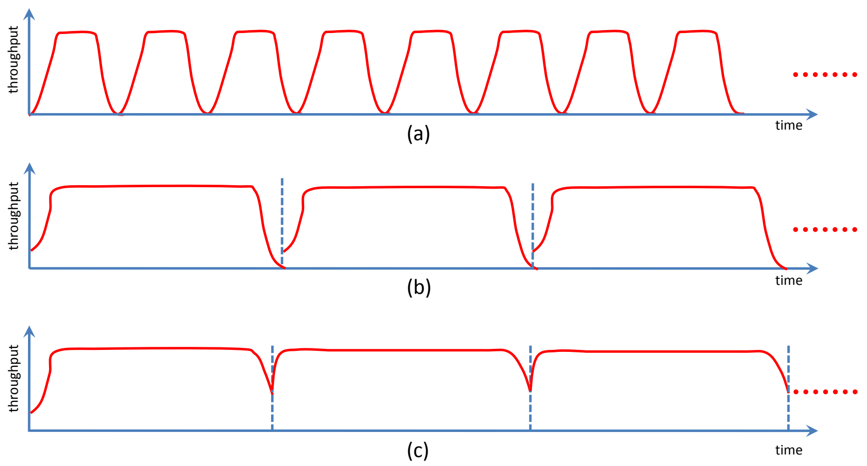


Fig. 1 Different strategies for parallel collision query using work queues. (a) Naive way: repeat the single collision query algorithm in [16] one by one; (b) Work queues are initialized by some BVTT root nodes and we repeat the process until all queries are performed. (c) is similar to (b) except that new BVTT root nodes are added to the work queues by the pump kernel, when there are not a sufficient number of tasks in the queue.

queries, where constant $t \leq \frac{\#global\ memory}{M}$ and we usually use $M = 50$. In this case, we only need to invoke the solution kernel $\lceil \frac{N}{M} \rceil$ times. The number of tasks available in the work queues changes more smoothly over time, with fewer ascending and descending parts, which implies higher throughput of the GPUs. Moreover, the work queues are initialized with many more tasks, which results in high performance at the beginning of each iteration. In practice, as nodes from more than one BVTT of different queries co-exist in the same queue, we need to distinguish them by representing each BVTT node by (x, y, i) instead of (x, y), where i is the index of collision query.

We can further improve the efficiency by using the pump operation (Algorithm 3 and Fig 2). That is, instead of initializing the work queues after it is completely empty, we add M BVTT root nodes of unresolved collision queries into the work queues when the number of tasks in it decreases to a threshold (we use $10 \cdot$ *chunk-size*). As a result, the few ascending and descending parts in Figure 1(b) can be further flattened as shown in Figure 1(c). Pump operation can reduce the timing overload of interrupting traversal kernels or copying data between global memory and shared memory, and therefore improve the overall efficiency of collision computation.

4.3 Analysis

In this section, we analyze the algorithms described above using the *parallel random access machine* (PRAM) model, which is a popular tool to analyze the complexity

Algorithm 3. Traversal with Workload Balancing

```
1: task_kernel()
2: input abort signal $signal$, $N$ random configurations $\{\mathbf{q}_i\}_{i=1}^N$, BVH$_a$ for the robot and
   BVH$_b$ for the obstacles
3: shared $WQ[] \equiv$ local work queue
4: initialize $WQ$ by tasks in global work queues
5: $\lhd$ traverse on work queues instead of BVTTs
6: loop
7:    $(x,y,i) \leftarrow pop(WQ)$
8:    if $overlap(\text{BVH}_a(\mathbf{q}_i)[x], \text{BVH}_b[y])$ then
9:       if $isLeaf(x)$ && $isLeaf(y)$ then
10:         update collision status of $i$-th query if $exactIntersect(\text{BVH}_a(\mathbf{q}_i)[x], \text{BVH}_b[y])$
11:      else
12:         $WQ[] \leftarrow (x,y,i)$'s children
13:      end if
14:   end if
15:   if $WQ$ is full or empty then
16:      $atomicInc(signal)$, break
17:   end if
18: end loop
19: return if $signal > 50\%Q$
```

```
1: balance_process()
2: copy local queue back to global work queue                        $\lhd$ manage_kernel
3: compute size of each work queue $n_i, 1 \leq i \leq Q$
4: if $\exists i, n_i < T_l || n_i > T_u$ then
5:    rearrange the tasks so that each queue has $n_i^* = \frac{\sum_{k=1}^{Q} n_k}{Q}$ tasks    $\lhd$ balance_kernel
6:    add more tasks in global queue if $\sum_{k=1}^{Q} n_k < T_{pump}$                  $\lhd$ pump_kernel
7: end if
```

of parallel algorithms [11]. Of course, current GPU architectures have many properties that can not be described by PRAM model, such as SIMT, shared memory, etc. However, PRAM analysis can still provide some insight into GPU algorithm's performance.

Suppose we have n collision queries, which means that we need to traverse n BVTT of the same tree structure but with different geometry configurations. We also suppose the GPU has p parallel processors. For convenience, assume $n = ap, a \in \mathbb{Z}$. Let the complexity to traverse the i-th BVTT be $W(i)$, $1 \leq i \leq n$. Then the complexity of a sequential CPU algorithm is $T_S(n) = \sum_{i=1}^{n} W(i)$ and the complexity of Algorithm 1 would be $T_N(n) = \sum_{k=0}^{a-1} \max_{j=1}^{p} W(kp+j)$. If we sort $\{W(i)\}_{i=1}^{n}$ in ascending order and denote $W^*(i)$ as the i-th element in the new order, we can prove $\sum_{k=0}^{a-1} \max_{j=1}^{p} W(kp+j) \geq \sum_{k=1}^{a} W^*(kp)$. Therefore $T_N(n) \geq \sum_{k=1}^{a} W^*(kp)$. Moreover, it is obvious that $\sum_{i=1}^{n} W(i) \geq T_N(n) \geq \frac{\sum_{i=1}^{n} W(i)}{p}$, which means $T_N(n) = \Theta(T_S(n))$, i.e. $T_N(n)$ is work-efficient [11].

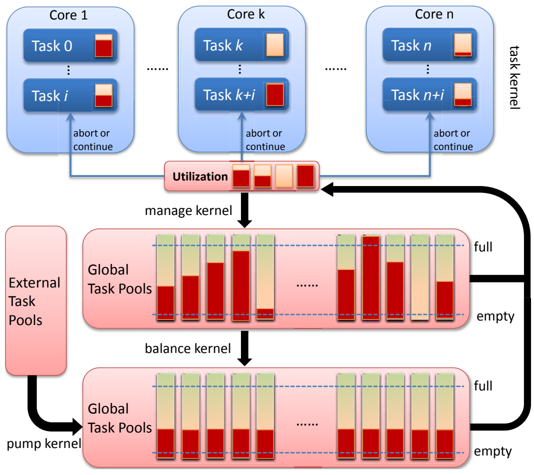


Fig. 2 Load balancing strategy for our parallel collision query algorithm. Each thread keeps its own local work queue in local memory. After processing a task, each thread is either able to run further or has an empty or full work queue and terminates. Once the number of GPU cores terminated exceeds a given threshold, the *manage kernel* is called and copies the local queues back onto global work queues. If no work queue has too many or too few tasks, the *task kernel* restarts. Otherwise, the *balance kernel* is called to balance the tasks among all queues. If there are not sufficient tasks in the queues, more BVTT root nodes will be 'pumped' in by the *pump kernel*.

According to the analysis in Section 4.1, we know that the expected complexity $\hat{W}(i)$ for i-th BVTT traversal in Algorithm 2 should be smaller than $W(i)$ because of the near-optimal traversing order. Moreover, the clustering strategy is similar to ordering different BVTTs, so that the BVTTs with similar traversal paths are arranged closely to each other and thus the probability is higher that they would be distributed on the same GPU core. Of course we can not implement ordering exactly because the BVTT traversal complexity is not known a priori. Therefore the complexity of Algorithm 2 is $T_P(n) \approx \sum_{k=1}^{a} \hat{W}^*(kp)$, with $\hat{W}^* \leq W^*$.

The complexity for Algorithm 3 is simple: $T_B(n) = \frac{\sum_{i=1}^{n} W(i)}{p} + B(n)$, where the first item is the timing complexity for BVTT traversal and the second item $B(n)$ is the timing complexity for balancing step. As $B(n) > 0$, the acceleration ratio of GPU with p-processors is less than p. We need to reduce the overload of balancing step to improve the efficiency of Algorithm 3.

Therefore, all three algorithms are work-efficient. If $B(n) = o(T_S(n))$, then $T_N(n) \geq T_P(n) \geq T_B(n)$ and Algorithm 3 is the most efficient one. If $B(n)$ is $\Theta(T_S(n))$, which means the overhead of balancing kernel is large, then it is possible to have $T_B(n) > T_P(n)$. Moreover, for large models, $W(i)$ would be quite different and the performance difference between three algorithms would be larger.

5 Implementation and Results

In this section, we present some details of the implementation and highlight the performance of our algorithm on different benchmarks. All the timings reported here were recorded on a machine using an Intel Core i7 3.2GHz CPU and 6GB memory. We implemented our collision and planning algorithms using CUDA on a NVIDIA GTX 480 GPU with 1GB of video memory.

We use the motion planning framework called *gPlanner* introduced in [23, 22], which uses PRM as the underlying planning algorithm as it is more suitable to exploit the multiple cores and data parallelism on GPUs. It can either compute a complete roadmap or we use a lazy version to perform a single motion planning query. We replace the collision module in gPlanner with the new algorithms described above. As observed in [23], more than 90% time of the planning algorithm is spent in collision queries, i.e. milestone computation step and local planning step.

In order to compare the performance of different parallel collision detection algorithms, we use the benchmarks highlighted in Figure 3. Their geometric complexities are highlighted in Table 1. For rigid body benchmarks, we generate $50,000$ random configurations and compute a collision-free path by using different variants of our parallel collision detection algorithm. For articulated model benchmark, we generate $100,000$ random configurations. For milestone computation, we directly use the collision detection algorithms. For local planning, we first need to unfold all the interpolated configurations: we denote the BVTT for the j-th interpolated query between the i-th local path as $\mathrm{BVTT}(i,j)$ and its node as (x,y,i,j). In order to avoid unnecessary computations, we first add BVTT root nodes with small j into the work queues, i.e. $(1,1,i,j) \prec (1,1,i',j')$, if$j < j'$. As a result, once a collision is found at $\mathrm{BVTT}(i,j_0)$, we need not to traverse $\mathrm{BVTT}(i,j)$ when $j > j_0$.

Table 1 Geometric complexity of our benchmarks. Large-piano is a piano with more vertices and faces by subdividing the piano model.

	piano	large-piano	helicopter	humanoid
#robot-faces	6540	34880	3612	27749
#obstace-faces	648	13824	2840	3495
DOF	6	6	6	38

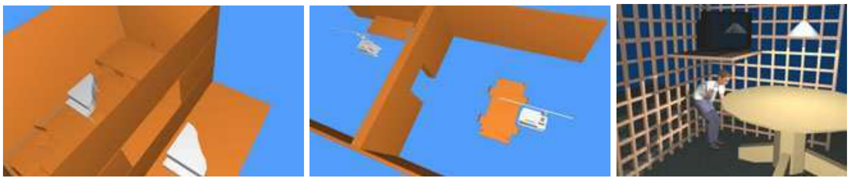

(a) piano (b) helicopter (c) humanoid

Fig. 3 Benchmarks for parallel collision queries.

Table 2 Comparison of different algorithms in milestone computation (timing in milliseconds). 32 and 128 are the different sizes used for the traversal stack; C and no-C means using pre-clustering and not using pre-clustering, respectively; timing of Algorithm 3 includes two parts: traversal part and balancing part.

	Algorithm 1				Algorithm 2				Algorithm 3	
	32, no-C	32, C	128, no-C	128, C	32, no-C	32, C	128, no-C	128, C	traverse	balancing
piano	117	113	239	224	177	131	168	130	68	3.69
large-piano	409	387	738	710	613	535	617	529	155	15.1
helicopter	158	151	286	272	224	166	226	163	56	2.3
humanoid	2392	2322	2379	2316	2068	1877	2073	1823	337	106

Table 3 Comparison of different algorithms in local planning (timing in milliseconds). 32 and 128 are the different sizes used for the traversal stack; C and no-C means using pre-clustering and not using pre-clustering, respectively; timing of Algorithm 3 includes two parts: traversal part and balancing part.

	Algorithm 1				Algorithm 2				Algorithm 3	
	32, no-C	32, C	128, no-C	128, C	32, no-C	32, C	128, no-C	128, C	traverse	balancing
piano	1203	1148	2213	2076	1018	822	1520	1344	1054	34
large-piano	4126	3823	8288	7587	5162	4017	7513	6091	1139	66
helicopter	4528	4388	7646	7413	3941	3339	5219	4645	913	41
humanoid	5726	5319	9273	8650	4839	4788	9012	8837	6082	1964

For Algorithm 1 and Algorithm 2, we further test the performance for different traversal sizes (32 and 128). Both algorithms give correct results when using a larger stack size (128). For smaller stack sizes, the algorithms will stop once the stack is filled. Algorithm 1 may report a collision when the stack overflows while Algorithm 2 returns a collision-free query. Therefore, Algorithm 1 may suffer from false positive errors while Algorithm 2 may suffer from false negative errors. We also compare the performance of Algorithm 1 and Algorithm 2 when the clustering algorithm described in Section 4.1 is used and when it is not.

The timing results are shown in Table 2 and Table 3. We can observe: (1) Algorithm 1 and Algorithm 2 both work better when local traverse stack is smaller and when pre-clustering is used. However for large models, traversal stack of size 32 may overflow and the collision results will be incorrect, which happens for the large-piano benchmarks in Table 2 and Table 3. Algorithm 1's performance will be terribly reduced when traverse stack size increases to 128 while Algorithm 2 does not change much. The reason is that Algorithm 2 uses per-packet stack, which is about 32 times less than using per-thread stack. Clustering and packet can result in a more than 50% speed-up. Moreover, the improvement of Algorithm 2 over Algorithm 1 is increased on larger models (large-piano) than on smaller models (piano). (2) Algorithm 3 is usually the fastest one among all the variations of the three algorithms. It can result in more than 5-10x increase in acceleration.

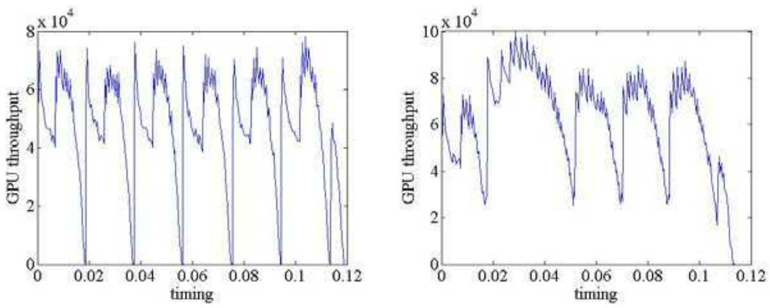


Fig. 4 GPU throughput improvement caused by pump kernel. Left figure shows the throughput without using the pump and right figure shows the throughput using the pump.

As observed in [23, 22], all these benchmarks are dominated by milestone computation and local planning steps as part of the overall parallel motion planning framework. The two parts take more than 50% running time in both the basic PRM and lazy PRM. Therefore, the overall planning algorithm can be improved by at least 40%-45%.

In Figure 4, we also show how the pump kernel increases the GPU throughput (i.e. the number of tasks available in work queues for GPU cores to fetch) in workload balancing based algorithm Algorithm 3. The maximum throughput (i.e. the maximum number of BV overlap tests performed by GPU kernels) increases from 8×10^4 to nearly 10^5 and the minimum throughput increases from 0 to 2.5×10^4. For piano and helicopter, we can compute a collision-free path from the initial to the goal configuration in in 879ms and 778ms separately using PRM or 72.79ms or 72.68ms using lazy PRM.

6 Conclusion and Future Work

In this paper, we introduce two novel parallel collision query algorithms for real-time motion planning on GPUs. The first algorithm is based on configuration-packet tracing, is easy to implement and can improve the parallel performance by performing more coherent traversals and reduce the memory consumed by traversal stacks. It can provide more than 50% speed-up as compared to simple parallel methods. The second algorithm is based on workload balancing, and decomposes parallel collision queries into fine-grained tasks of BVTT node operations. The algorithm uses a light-weight task-balancing strategy to guarantee that all GPU cores are fully loaded and achieves close to the peak performance on GPUs. It can provide 5-10X speed-up compared to simple parallel strategy. The overall performance of the GPU-based randomized planner also increases more than 50% when compared to the previous GPU planner.

There are many avenues for future work. We are interested in using more advanced sampling schemes with the planner to further improve its performance and allow us to work on motion planning problems with narrow passages. Furthermore, we would like to adjust the planner to generate smooth paths and integrate our planner with certain robots (e.g. PR2).

Acknowledgements. This work was supported in part by ARO Contract W911NF-04-1-0088, NSF awards 0636208, 0917040 and 0904990, DARPA/RDECOM Contract WR91CRB-08-C-0137, and Intel.

References

1. Aila, T., Laine, S.: Understanding the efficiency of ray traversal on GPUs. In: High Performance Graphics, pp. 145–149 (2009)
2. Akinc, M., Bekris, K.E., Chen, B.Y., Ladd, A.M., Plaku, E., Kavraki, L.E.: Probabilistic roadmaps of trees for parallel computation of multiple query roadmaps. In: Robotics Research. Springer Tracts in Advanced Robotics, vol. 15, pp. 80–89. Springer, Heidelberg (2005)
3. Amato, N., Dale, L.: Probabilistic roadmap methods are embarrassingly parallel. In: International Conference on Robotics and Automation, pp. 688–694 (1999)
4. Banerjee, A., Ghosh, J.: Scalable clustering algorithms with balancing constraints. Data Mining and Knowledge Discovery 13(3), 365–395 (2006)
5. Barraquand, J., Latombe, J.C.: Robot motion planning: A distributed representation approach. International Journal of Robotics Research 10(6) (1991)
6. Che, S., Boyer, M., Meng, J., Tarjan, D., Sheaffer, J.W., Skadron, K.: A performance study of general-purpose applications on graphics processors using cuda. Journal of Parallel and Distributed Computing 68(10), 1370–1380 (2008)
7. Foskey, M., Garber, M., Lin, M., Manocha, D.: A voronoi-based hybrid planner. In: International Conference on Intelligent Robots and Systems, pp. 55–60 (2001)
8. Gunther, J., Popov, S., Seidel, H.P., Slusallek, P.: Realtime ray tracing on GPU with BVH-based packet traversal. In: IEEE Symposium on Interactive Ray Tracing, pp. 113–118 (2007)
9. Harris, M.: Optimizing parallel reduction in CUDA. NVIDIA Developer Technology (2009)
10. Hoff, K., Culver, T., Keyser, J., Lin, M., Manocha, D.: Interactive motion planning using hardware accelerated computation of generalized voronoi diagrams. In: International Conference on Robotics and Automation, pp. 2931–2937 (2000)
11. JáJá, J.: An introduction to parallel algorithms. Addison Wesley Longman Publishing Co., Inc., Amsterdam (1992)
12. Kavraki, L., Svestka, P., Latombe, J.C., Overmars, M.: Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Transactions on Robotics and Automation 12(4), 566–580 (1996)
13. Kim, D., Heo, J.P., Huh, J., Kim, J., Yoon, S.E.: HPCCD: Hybrid parallel continuous collision detection using cpus and gpus. Computer Graphics Forum 28(7), 1791–1800 (2009)
14. Kuffner, J., LaValle, S.: RRT-connect: An efficient approach to single-query path planning. In: International Conference on Robotics and Automation, pp. 995–1001 (2000)
15. Larsen, E., Gottschalk, S., Lin, M., Manocha, D.: Distance queries with rectangular swept sphere volumes. In: International Conference on Robotics and Automation, pp. 3719–3726 (2000)

16. Lauterbach, C., Mo, Q., Manocha, D.: gproximity: Hierarchical gpu-based operations for collision and distance queries. Computer Graphics Forum 29(2), 419–428 (2010)
17. LaValle, S.M.: Planning Algorithms. Cambridge University Press, Cambridge (2006)
18. Lin, M., Manocha, D.: Collision and proximity queries. In: Handbook of Discrete and Computational Geometry, pp. 787–808. CRC Press, Inc., Boca Raton (2004)
19. Lindholm, E., Nickolls, J., Oberman, S., Montrym, J.: NVIDIA Tesla: A unified graphics and computing architecture. IEEE Micro 28(2), 39–55 (2008)
20. Lozano-Perez, T., O'Donnell, P.: Parallel robot motion planning. In: International Conference on Robotics and Automation, pp. 1000–1007 (1991)
21. Owens, J.D., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A.E., Purcell, T.: A survey of general-purpose computation on graphics hardware. Computer Graphics Forum 26(1), 80–113 (2007)
22. Pan, J., Lauterbach, C., Manocha, D.: Efficient nearest-neighbor computation for GPU-based motion planning. In: International Conference on Intelligent Robots and Systems (to appear, 2010)
23. Pan, J., Lauterbach, C., Manocha, D.: g-planner: Real-time motion planning and global navigation using GPUs. In: AAAI Conference on Artificial Intelligence, pp. 1245–1251 (2010)
24. Pisula, C., Hoff, K., Lin, M.C., Manocha, D.: Randomized path planning for a rigid body based on hardware accelerated voronoi sampling. In: International Workshop on Algorithmic Foundation of Robotics, pp. 279–292 (2000)
25. Plaku, E., Bekris, K.E., Kavraki, L.E.: Oops for motion planning: An online open-source programming system. In: International Conference on Robotics and Automation, pp. 3711–3716 (2007)
26. Sud, A., Andersen, E., Curtis, S., Lin, M., Manocha, D.: Real-time path planning for virtual agents in dynamic environments. IEEE Virtual Reality, 91–98 (2007)
27. Talamadupula, K., Benton, J., Schermerhorn, P.: Integrating a closed world planner with an open world. In: ICAPS Workshop on Bridging the Gap Between Task and Motion Planning (2009)
28. Tang, M., Manocha, D., Tong, R.: Mccd: Multi-core collision detection between deformable models. Graphical Models 72(2), 7–23 (2010)
29. Zhang, L., Manocha, D.: A retraction-based RRT planner. In: International Conference on Robotics and Automation, pp. 3743–3750 (2008)

CCQ: Efficient Local Planning Using Connection Collision Query

Min Tang, Young J. Kim, and Dinesh Manocha

Abstract. We introduce a novel proximity query, called *connection collision query* (CCQ), and use it for efficient and exact local planning in sampling-based motion planners. Given two collision-free configurations, CCQ checks whether these configurations can be connected by a given continuous path that either lies completely in the free space or penetrates any obstacle by at most ε, a given threshold. Our approach is general, robust, and can handle different continuous path formulations. We have integrated the CCQ algorithm with sampling-based motion planners and can perform reliable local planning queries with little performance degradation, as compared to prior methods. Moreover, the CCQ-based exact local planner is about an order of magnitude faster than prior exact local planning algorithms.

1 Introduction

Planning a collision-free path for a robot amongst obstacles is an important problem in robotics, CAD/CAM, computer animation and bioinformatics. This problem is well studied and many approaches have been proposed. Over the last few decades, sampling-based motion planners such as probabilistic roadmaps [10] (PRMs) or rapidly-exploring random trees [15] (RRTs) have been shown to be successful in terms of solving challenging problems with high degrees-of-freedom (DoFs) robots. These planners attempt to capture the topology of the free space by generating random configurations and connecting nearby configurations using local planning algorithms.

The main goal of a local planner is to check whether there exists a collision-free path between two free configurations. It is important that the local planner should

Min Tang · Young J. Kim
Dept of Computer Science and Engineering, Ewha Womans University, Seoul, Korea
e-mail: {tangmin,kimy}@ewha.ac.kr

Dinesh Manocha
Dept of Computer Science, University of North Carolina, Chapel Hill, U.S.A
e-mail: dm@cs.unc.edu

be reliable and does not miss any collisions with the obstacles [8, 1]. Moreover, a significant fraction of the overall running time of a sampling-based planner is spent in the local planning routines.

The simplest local planning algorithms compute a continuous interpolating path between free configurations and check the path for collisions with any obstacles. These algorithms sample the continuous path at a fixed resolution and discretely check each of the resulting configurations for collisions. These *fixed-resolution local planning* algorithms are simple to implement, but suffer from two kinds of problems:

1. **Collision-miss:** It is possible for the planner to miss a collision due to insufficient sampling. This can happen in narrow passages or when the path lies close to the obstacle boundary, or when dealing with high DOF articulated models. This affects overall accuracy of the planner.
2. **Collision-resolution:** Most planners tend to be conservative and generate a very high number of samples, which results in a lot of discrete collision queries and affects the running time of the planner.

Overall, it is hard to compute the optimal resolution parameter that is both fast and can guarantee collision-free motion. In order to overcome these problems, some local planners use exact methods such as continuous collision detection (CCD) [18, 30] or dynamic collision checking [23]. However, these *exact local planning* methods are regarded as expensive and are much slower than fixed-resolution local planners [23]. Many well-known implementations of sampling-based planners such as OOPSMP[1] and MSL[2] only use fixed-resolution local planning, though MPK[3] performs exact collision checking for local planning.

Main Results. We introduce a novel proximity query, namely *connection collision query* (CCQ), for fast and exact local planning in sampling-based motion planners. At a high level, our CCQ algorithm can report two types of proximity results:

- **Boolean CCQ query:** Given two collision-free configurations of a moving robot in the configuration space, CCQ checks whether the configurations can be connected by a given path that lies in the free space, namely *Boolean* CCQ_s *query*. In addition, the CCQ query can also check whether the path lies partially inside the obstacle region (C-obstacle) with at most ε-penetration, namely *Boolean* CCQ_p *query*. In this case, the robot may overlap with some obstacles and the extent of penetration is bounded above by ε.
- **Time of violation (ToV) query:** If the Boolean queries report `FALSE` (*i.e.* the path does not exist), the CCQ query reports the first parameter or the configuration along the continuous path that violates these path constraints.

Moreover, our algorithm can easily check different types of continuous paths including a linear interpolating motion in the configuration space or a screw motion.

[1] `http://www.kavrakilab.org/OOPSMP`
[2] `http://msl.cs.uiuc.edu/`
[3] `http://robotics.stanford.edu/~mitul/mpk/`

We have integrated our CCQ algorithm into well-known sampling-based motion planners and compared their performance with prior methods. In practice, we observe that an exact local planning algorithm based on the CCQ query can be at most two times slower than a fixed-resolution local planning based on PRM and RRT, though the paths computed using CCQ queries are guaranteed to be collision-free. Finally, we also show that our CCQ algorithm outperforms prior exact local planners by one order of magnitude.

Paper Organization. The rest of this paper is organized as follows. In Sec. 2, we briefly survey the related work and formulate the CCQ problem in Sec. 3. Sections 4 and 5 describe the CCQ algorithms for rigid robots with separation and penetration constraints, respectively. We describe how our CCQ algorithm can be extended to articulated robots in Sec.6, and highlight the results for different benchmarks in Sec. 7.

2 Previous Work

Our CCQ algorithm is related to continuous collision detection. In this section, we give a brief survey on these proximity queries and local planning.

2.1 Continuous Collision Detection

The term continuous collision detection was first introduced by Redon *et al.* [18] in the context of rigid body dynamics, even though the earlier work on similar problems dates back to the late 1980s [3]. The main focus of CCD algorithms lies in finding the first time of contact for a fast moving object between two discrete collision-free configurations. Many CCD algorithms for rigid models have been proposed [24]: these include algebraic equation solvers, swept volume formulations, adaptive bisection approach, kinetic data structures approach, Minkowski sum-based formulations and conservative advancement (CA).

For articulated models, Redon *et al.*[19] present a method based on continuous OBB-tree test, and Zhang *et al.* [30] have extended the CA method to articulated models. In the context of motion planning, Schwarzer *et al.* [23] present a dynamic collision checking algorithm to guarantee a collision-free motion between two configurations. These algorithms have been mainly used for rigid body dynamics and their application to sampling-based planning has been limited [23]. In practice, the performance of these exact local planning methods is considered rather slow for motion planners. Moreover, current CCD algorithms are not optimized for reporting Boolean results and cannot handle penetration queries such as CCQ$_p$, that are useful for local planners and narrow passages.

2.2 Local Planning

There are two important issues related to our work in terms of local planning: the type of continuous interpolating path and the validity of the path in terms of collisions. The former is related to motion interpolation between collision-free samples, and the latter is related to collision checking.

2.2.1 Motion Interpolation

In the context of local planning, different types of motion interpolation methods have been used such as linear motion in C-space [23], spherical motion in C-space [11], screw motion [20], etc. These motion trajectories are rather simple to compute and cost-effective for local planning.

More sophisticated motion interpolation techniques have been introduced to find an effective local path by taking into account the robot/obstacle contacts [9, 5], variational schemes [25] and distance constraints [27]. Amato *et al.* [1] evaluate different distance metrics and local planners, and show that the translational distance becomes more important than the rotational distance in cluttered scenes.

2.2.2 Collision Checking

Given a path connecting two collision-free configurations, a conventional way of local planning is to sample the path at discrete intervals and perform static collision detection along the discrete path [13, 14]. Some exact collision checking methods have been proposed for local planning such as [23, 4] using adaptive bisection.

Since collision checking can take more than 90% of the total running time in sampling-based planners, lazy collision, lazy collision evaluation techniques have been proposed [22, 2] to improve the overall performance of a planner. The main idea is to defer collision evaluation along the path until it is absolutely necessary. While these techniques help to greatly improve the performance of PRM-like algorithms, but they do not improve the reliability of resolution-based collision checkers.

When narrow passages are present in the configuration space, it is hard to capture the connectivity of the free space by using simple collision checking, since it may report a lot of invalid local paths. However, some retraction-based planners [7, 6, 4, 28] allow slight penetration into the obstacle region based on penetration depth computation, which makes the local planning more effective.

3 Problem Formulation

We start this section by introducing the notation that is used throughout the paper. Next, we give a precise formulation of CCQ.

3.1 Notations and Assumptions

We use bold-faced letters to denote vector quantities (*e.g.* $\mathbf{o}$). Many other symbols used in the paper are given in Table 1. We assume that both the robot $\mathfrak{A}$ and obstacle $\mathfrak{B}$ are rigid and defined in $\mathbb{R}^3$ workspace. Moreover, the robot has 6 DoFs and the

Table 1 Notations.

Notation	Meaning
$\mathfrak{A}, \mathfrak{B}, \partial\mathfrak{A}, \partial\mathfrak{B}$	robot, obstacle and their boundaries
$\mathscr{C}$	C-space of $\mathfrak{A}$
$\mathbf{q}, \mathbf{q}(t)$	a sample in C-space and a $1D$ curve in C-space
$\mathfrak{A}(\mathbf{q}), \mathfrak{A}(t)$	placements of the robot $\mathfrak{A}$ at $\mathbf{q}$ and $\mathbf{q}(t)$
$\mathscr{F}, \mathscr{O}$	C-free and C-obstacle region in $\mathscr{C}$ (*i.e.* $\mathscr{C} = \mathscr{F} \cup \mathscr{O}$)
$\|\cdot,\cdot\|$	Euclidean distance operator

obstacle is fixed in space; thus, the C-space of $\mathfrak{A}$ is SE(3). We briefly discuss how to handle high DoF robots later in Sec. 6.

3.2 Local Planning in Sampling-Based Motion Planner

Given the starting $\mathbf{q}_0$ and goal $\mathbf{q}_1$ configurations in $\mathscr{F}$, most sampling-based randomized planners compute a search graph $\mathscr{G}$ to explore the C-space, where the vertex corresponds to a sample in $\mathscr{F}$ and each edge corresponds to a $1D$ curve in C-space connecting two collision-free samples. More specifically, sampling-based planners work in the following manner:

1. **Sample Generation:** Sample a collision-free configuration $\mathbf{q}_1$ in $\mathscr{F}$.
2. **Local Planning:** Check whether $\mathbf{q}_1$ can be connected to a vertex $\mathbf{q}_0$ in $\mathscr{G}$ by some collision-free, continuous path $\mathbf{q}(t)$ in C-space. If so, a new edge connecting $\mathbf{q}_0, \mathbf{q}_1$ is created and added to $\mathscr{G}$ along with the vertex $\mathbf{q}_1$.
3. **Graph Search:** Perform graph search on $\mathscr{G}$ to find a path from $\mathbf{q}_0$ to $\mathbf{q}_1$. If such a path is found, the algorithm reports the path; otherwise, go back to step 1 and repeat.

In the local planning step, the choice of a continuous path $\mathbf{q}(t)$ interpolating $\mathbf{q}_0, \mathbf{q}_1$ may vary depending on the topology of $\mathscr{F}$. Once a specific path formulation is chosen, the algorithm needs to check whether that path is collision-free or not.

3.3 Connection Collision Query

Now we define the CCQ proximity query, the main problem to solve in this paper. Let us assume that two collision-free samples $\mathbf{q}_0, \mathbf{q}_1 \in \mathscr{F}$ in $\mathscr{C}$ and a time-parameterized, continuous $1D$ curve $\mathbf{q}(t)$ in $\mathscr{C}$ connecting $\mathbf{q}_0$ and $\mathbf{q}_1$ for $t \in [0,1]$; *i.e.* $\mathbf{q}(0) = \mathbf{q}_0, \mathbf{q}(1) = \mathbf{q}_1$. Then, the CCQ with separation constraint is formally defined as checking whether the following predicate CCQ$_s$ is `TRUE`:

$$\text{CCQ}_s : \forall t \in [0,1] \Rightarrow \mathbf{q}(t) \in \mathscr{F}. \tag{1}$$

Moreover, if CCQ$_s$ is `FALSE`, we want to determine the maximum value of t that satisfies CCQ$_s$. We call such t as the time of violation (ToV) with separation, τ_s. More formally,

$$\tau_s \equiv \max_t \{ \forall s \in [0,t] \mid \mathbf{q}(s) \in \mathscr{F} \}. \tag{2}$$

The Boolean CCQ$_s$ query is useful for local planning in PRM and RRT, and the ToV CCQ$_s$ query can be used for local planning or the expansion step in RRT.

On the other hand, the notion of CCQ with ε-penetration is a less restrictive version of connection query than CCQ with separation constraint, as it allows slight penetration (quantified by ε) into the C-obstacle region for the C-space curve $\mathbf{q}(t)$. Formally, we define CCQ with ε-penetration as checking whether the following predicate CCQ$_p$ is `TRUE`:

$$\begin{aligned}\text{CCQ}_p : \forall t \in [0,1] \Rightarrow \{\mathbf{q}(t) \in \mathscr{F}\} \vee \\ \{\mathbf{q}(t) \in \mathscr{O} \wedge \forall \mathbf{p} \in \mathfrak{A}(t) \cap \mathfrak{B}, \|\mathbf{p} - \partial\mathfrak{B}\| \leq \varepsilon\}.\end{aligned} \tag{3}$$

Furthermore, if CCQ$_p$ is `FALSE`, we also determine the maximum value of t that satisfies CCQ$_p$, called the ToV with ε-penetration, τ_p. More formally, τ_p is defined as:

$$\begin{aligned}\tau_p \equiv \max_t \{ \forall s \in [0,t] \mid \mathbf{q}(s) \in \mathscr{F} \vee \\ \{\mathbf{q}(s) \in \mathscr{O} \wedge \forall \mathbf{p} \in \mathfrak{A}(s) \cap \mathfrak{B}, \|\mathbf{p} - \partial\mathfrak{B}\| \leq \varepsilon\}\}.\end{aligned} \tag{4}$$

The CCQ$_p$ query can be used for PRM and RRT when a small amount of penetration is allowed for a robot along the local path. Moreover, retraction-based planners may use CCQ$_p$ to generate samples with slight penetration [7, 6, 4].

4 CCQ with Separation Constraint

In this section, we present our algorithm to perform the CCQ$_s$ query. We start this section by explaining the conservative advancement (CA) technique upon which our CCQ algorithm is based. Next, we explain the procedure to compute the ToV information τ_s in Eq.2 along with CCQ$_s$. Finally, we provide a fast technique to solve the Boolean version of CCQ$_s$ (*i.e.* Eq.1).

4.1 Conservative Advancement

Our CCQ algorithm is based on the conservative advancement (CA) algorithm [16] for convex objects undergoing continuous motion. In CA, the time of contact (ToC) τ between two convex objects $\mathfrak{A}$ and $\mathfrak{B}$ is obtained by iteratively advancing $\mathfrak{A}$ by Δt_s toward $\mathfrak{B}$ without generating collisions. Here, Δt_s can be calculated by:

$$\Delta t_s \leq \frac{\|\mathfrak{A}(t),\mathfrak{B}\|}{\mu} \tag{5}$$

where μ is the bound of motion of $\mathfrak{A}(t)$ for $t \in [0,1]$ projected onto the closest direction from $\mathfrak{A}(t)$ to $\mathfrak{B}$, known as the directional motion bound [29]. Then, the ToC is obtained as:

$$\tau = \sum_i \Delta t_s^i \tag{6}$$

where Δt_s^i denotes the ith CA iteration. The iteration continues until $\|\mathfrak{A}(\tau), \mathfrak{B}\| \approx 0$. This idea can be extended to non-convex models using bounding volume hierarchies [24].

4.2 Time of Violation Query for CCQ$_s$

In case of CCQ$_s$, the time of violation (ToV) is equivalent to the time of contact (ToC) in CA. Moreover, if the path $\mathbf{q}(t)$ is a linear motion in C-space, one can employ the C^{2}A algorithm [24] based on CA to compute τ_s for the robot $\mathfrak{A}$. We also show that we can devise a variant of C^{2}A algorithm that can handle the screw motion for $\mathbf{q}(t)$.

The screw motion consists of rotation about an axis ω in space by an angle of θ radians, followed by translation along the same axis by an amount of d as shown in Fig.1.The screw motion can be represented by using four parameters $(\omega, \theta, \mathbf{a}, \mathbf{d})$, where $\mathbf{a}$ is any point on the axis ω. Given two configurations $\mathbf{q_0}$ and $\mathbf{q_1}$ in SE(3), the screw parameters can be easily computed [21].

The main challenge in computing τ_s under screw motion is to compute the directional motion bound μ for Eq.5. Let us assume that our robot $\mathfrak{A}$ is convex with the origin $\mathbf{o}^b$ of the body attached frame. Let $\mathbf{p}$ be any point on $\mathfrak{A}$ with $\mathbf{p}^b$ representing the same point but defined with respect to the body frame, $\mathbf{n}$ be the closest direction from $\mathfrak{A}$ to the obstacle $\mathfrak{B}$ at $t = 0$, $\mathbf{p}_\perp$ be a vector projected from $\mathbf{p}$ to the axis ω. Then, an upper bound μ of the motion of any point on $\mathfrak{A}$ under screw motion, projected onto $\mathbf{n}$ is:

$$\begin{aligned}
\mu &= \max_{\mathbf{p}\in\mathfrak{A}} \left(\int_0^1 (\dot{\mathbf{p}}(t) \cdot \mathbf{n})\, dt \right) \\
&= \max_{\mathbf{p}\in\mathfrak{A}} \left(\int_0^1 ((\mathbf{v} + \omega \times \mathbf{p}_\perp(t)) \cdot \mathbf{n})\, dt \right) \\
&\le \max(\mathbf{v}\cdot\mathbf{n}, 0) + \|\omega \times \mathbf{n}\| \left(\max_{\mathbf{p}\in\mathfrak{A}} \left(\int_0^1 \|\mathbf{p}_\perp(t)\|\, dt \right) \right) \\
&\le \max(d\omega\cdot\mathbf{n}, 0) + \|\omega \times \mathbf{n}\| \left(\left\| \left(\mathbf{o}^b - \mathbf{a}\right) \times \omega \right\| + \max_{\mathbf{p}\in\mathfrak{A}} \left\|\mathbf{p}^b\right\| \right).
\end{aligned} \tag{7}$$

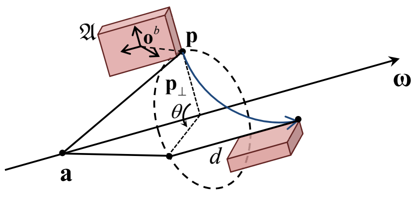


Fig. 1 Screw Motion.

Note that $\max_{\mathbf{p}\in\mathfrak{A}} \left\|\mathbf{p}^b\right\|$ can be calculated as preprocess, since $\mathbf{p}^b$ is defined with respect to the body frame. A similar bound can be obtained for other motion trajectories such as spherical motions [11].

4.3 Boolean Version of CCQ$_s$

From the previous section, the CCQ$_s$ predicate in Eq.1 can be trivially determined by checking whether $\tau_s \geq 1$ (`TRUE`) or not (`FALSE`). However, one can devise a more efficient way to answer the CCQ$_s$ predicate without explicitly computing τ_s.

Given the starting $\mathbf{q}_0$ and goal $\mathbf{q}_1$ configurations, the main idea in evaluating CCQ$_s$ is to perform dual advancements from both end-configurations $\mathbf{q}_0, \mathbf{q}_1$ with opposite velocities, and iterate this process until collision is found or the path turns out to be collision-free. The dual advancement is more effective than the normal advancement using a single end-configuration since the normal advancement is always conservative (i.e. collision will be never identified until the final ToV value is obtained).

More specifically, as shown in Fig. 2, we perform a single CA iteration from $\mathbf{q}_0$ towards $\mathbf{q}_1$ as before and compute the first advancement time, Δt_0^+. Similarly, we perform another CA iteration but from $\mathbf{q}_1$ towards $\mathbf{q}_0$ with a negative velocity (*e.g.* $(-\mathbf{v}, -\omega)$) and compute the advancement time, Δt_1^-.

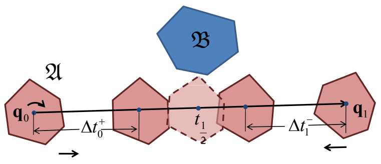


Fig. 2 A Single Step in the Boolean Query. Dual advancements are performed from $\mathbf{q}_0$ towards $\mathbf{q}_1$ by Δt_0^+, and from $\mathbf{q}_1$ towards $\mathbf{q}_0$ by Δt_1^-. The collision is checked at $\mathbf{q}(\frac{1}{2})$.

If $(\Delta t_0^+ + \Delta t_1^-) \geq 1$, then the entire path $\mathbf{q}(t)$ is collision-free, thus the predicate is returned as `TRUE`; otherwise, we bisect the time interval at $t_{\frac{1}{2}} = \frac{t_0+t_1}{2}$ and perform collision detection at the configuration $\mathbf{q}(t_{\frac{1}{2}})$. If collision is detected at $\mathbf{q}(t_{\frac{1}{2}})$, CCQ$_s$ is reported as `FALSE` and the procedure is terminated. Otherwise, the same dual CA procedure is executed recursively on two sub-paths, $\{[\mathbf{q}(\Delta t_0^+), \mathbf{q}(t_{\frac{1}{2}})], [\mathbf{q}(t_{\frac{1}{2}}), \mathbf{q}(1 - \Delta t_1^-)]\}$. Note that the remaining path segments $\{[\mathbf{q}(0), \mathbf{q}(\Delta t_0^+)], [\mathbf{q}(1 - \Delta t_1^-), \mathbf{q}(1)]\}$ are collision-free because of conservative advance mechanism. This procedure is iterated until the separation condition is satisfied or a collision is detected. We provide a pseudo-code for CCQ$_s$ in Alg.1.

Algorithm 1. CCQ$_s$
Input: initial and goal configurations $\mathbf{q}_0$, $\mathbf{q}_1$, interpolating motion $\mathbf{q}(t)$
Output: whether Eq. 1 is `TRUE` or `FALSE`

```
 1: {Initialize the queue with $[\mathbf{q}(0), \mathbf{q}(1)]$.}
 2: while Queue $\neq \emptyset$ do
 3:    Pop an element $[\mathbf{q}(t_a), \mathbf{q}(t_b)]$ from the queue;
 4:    $t_{\frac{1}{2}} = \frac{t_a+t_b}{2}$;
 5:    if $\mathbf{q}(t_{\frac{1}{2}})$ is in-collision then
 6:       return FALSE;
 7:    end if
 8:    Perform CA from $\mathbf{q}(t_a)$ with a positive velocity and find the step size $\Delta t_a^+$;
 9:    Perform CA from $\mathbf{q}(t_b)$ with a negative velocity and find the step size $\Delta t_b^-$;
10:    if $\left(\Delta t_a^+ + \Delta t_b^-\right) < (t_b - t_a)$ then
11:       Push $[\mathbf{q}(t_a + \Delta t_a^+), \mathbf{q}(t_{\frac{1}{2}})]$ and $[\mathbf{q}(t_{\frac{1}{2}}), \mathbf{q}(t_b - \Delta t_b^-)]$ onto the queue;
12:    end if
13: end while
14: return TRUE;
```

5 CCQ with Penetration Constraints

The CCQ$_s$ algorithm presented in Sec.4 strictly imposes that the interpolating path $\mathbf{q}(t)$ should lie entirely inside $\mathscr{F}$. However, this condition is rather restrictive since a slight overlap between the robot and the obstacles may be useful in practice and is used by retraction-based planners [7, 4, 28]. For instance, often the curved surface model of a robot is tessellated with some surface deviation error ε and thus ε-penetration does not necessarily imply actual interference [4]. The notion of CCQ$_p$ is that we allow slight penetration for a robot along the path as long as the penetration amount is less than some threshold, ε.

5.1 Penetration Depth

To quantify the amount of penetration for a robot $\mathfrak{A}$, we need a suitable metric. The penetration depth (PD) is a proper metric to quantify the amount of overlap between $\mathfrak{A}$ and $\mathfrak{B}$. In the literature, different types of penetration depth are known [26] and in our case, we use pointwise penetration depth [24] since it is computationally cheaper to compute as compared to other penetration measures.

When $\mathfrak{A}$ and $\mathfrak{B}$ overlap, the pointwise penetration depth is defined as the point of deepest interpenetration of $\mathfrak{A}$ and $\mathfrak{B}$. Formally, the pointwise penetration depth (or PD for short) can be defined as:

$$PD \equiv \mathscr{H}\left(\mathfrak{A} \cap \partial(\mathfrak{A} \cap \mathfrak{B}), \mathfrak{B} \cap \partial(\mathfrak{A} \cap \mathfrak{B})\right) \tag{8}$$

where $\mathscr{H}(\cdot,\cdot)$ denotes the two-sided Hausdorff distance operator between surfaces.

5.2 Boolean Version of CCQ_p

We first explain how to evaluate the CCQ$_p$ predicate in Eq.3. The main idea of our evaluation algorithm is to decompose the advancement step size Δt into two substeps Δt_s and Δt_p (*i.e.* $\Delta t = \Delta t_s + \Delta t_p$) such that collision-free motion is generated during Δt_s while Δt_p may induce penetration with the PD value being less than ε. Then, we perform dual CAs from the end-configurations $\mathbf{q}_0, \mathbf{q}_1$ like CCQ$_s$ in Sec. 4.3.

Since Δt_s can be calculated just like in Eq. 5, computing Δt boils down to calculating Δt_p. In general, computing Δt_p can be quite challenging since one needs to search the entire C-space (both C-free and C-obstacle) where the placement of $\mathfrak{A}$ at $\mathbf{q}(t + \Delta t)$ may yield either collision-free or in-collision configuration. In order to compute a feasible solution for Δt_p, we use a conservative approach.

The key idea is that, after the advancement of $\Delta t_s + \Delta t_p$ time step, want to guarantee that the robot still remains collision-free at $\mathbf{q}(t + \Delta t_s + \Delta t_p)$. Taking advantage of this constraint, we first move the robot to $\mathfrak{A}(t + \Delta t_s)$, and then calculate Δt_p that can bound the motion of $\mathfrak{A}$ by less than 2ε so that the possible PD between $\mathfrak{A}$ and $\mathfrak{B}$ can be less than ε, as shown in Fig. 3.

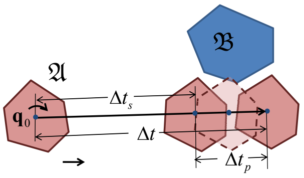


Fig. 3 Decomposition of the Time Step Δt into Δt_s and Δt_p for CCQ$_p$. Δt_s corresponds to the collision-free time step and Δt_p to the time step that may result in ε-penetration.

More precisely, an upper bound of the time step size Δt_p can be computed by observing the fact that the robot should not travel by more than 2ε; otherwise, the penetration depth can be greater than ε. Thus, assuming that the robot and obstacles are both convex, we have:

$$\Delta t_p \leq \frac{2\varepsilon}{\mu_u} \tag{9}$$

where μ_u is the maximum amount of motion that a point on $\mathfrak{A}$ can make between the time interval of $[0, 1]$. Note that μ_u is an undirected motion bound unlike the directed one μ in Eq.5, since no closest direction will be defined for a robot in collision with obstacles. Essentially, μ_u depends on the underlying path. We present simple formulas to compute μ_u for both linear (Eq.10) and screw (Eq.11) motions as shown below. Here, $\mathbf{p}, \mathbf{p}^b, \mathbf{p}_\perp, \mathbf{o}^b$ have the same meanings as defined in Sec.4.2.

Linear Motion

$$\begin{aligned}\mu_u &= \max_{\mathbf{p}\in\mathfrak{A}} \left(\int_0^1 \|\dot{\mathbf{p}}_i(t)\|\, dt \right) \\ &= \max_{\mathbf{p}\in\mathfrak{A}} \left(\int_0^1 \left\|\mathbf{v} + \omega \times \mathbf{p}^b(t)\right\|\, dt \right) \\ &\le \|\mathbf{v}\| + \max_{\mathbf{p}\in\mathfrak{A}} \left(\int_0^1 \left\|\omega \times \mathbf{p}^b(t)\right\|\, dt \right) \\ &\le \|\mathbf{v}\| + \|\omega\| \max_{\mathbf{p}\in\mathfrak{A}} \left\|\mathbf{p}^b\right\|\end{aligned} \tag{10}$$

Screw Motion

$$\begin{aligned}\mu_u &= \max_{\mathbf{p}\in\mathfrak{A}} \left(\int_0^1 \|\dot{\mathbf{p}}_i(t)\|\, dt \right) \\ &= \max_{\mathbf{p}\in\mathfrak{A}} \left(\int_0^1 \|\mathbf{v} + \omega \times \mathbf{p}_\perp(t)\|\, dt \right) \\ &\le \|\mathbf{v}\| + \max_{\mathbf{p}\in\mathfrak{A}} \left(\int_0^1 \|\omega \times \mathbf{p}_\perp(t)\|\, dt \right) \\ &\le \|\mathbf{v}\| + \|\omega\| \left(\left\|\left(\mathbf{o}^b - \mathbf{a}\right) \times \omega\right\| + \max_{\mathbf{p}\in\mathfrak{A}} \left\|\mathbf{p}^b\right\| \right)\end{aligned} \tag{11}$$

The result of our algorithm is conservative in the sense that our algorithm does not report a false-positive result; *i.e.* if the algorithm reports `TRUE`, it guarantees that CCQ$_p$ is indeed `TRUE`.

5.3 *Time of Violation Query for CCQ$_p$*

A simple way to compute the ToV in Eq.4 can be devised similarly to evaluating CCQ$_p$ by decomposing the ToV into the one corresponding to collision-free motion τ_s (Eq.2) and one to ε-penetration $\Delta t'_p$: *i.e.*

$$\tau_{p1} = \left(\sum_i \Delta t_s^i \right) + \Delta t_{p'} = \tau_s + \Delta t_{p'}. \tag{12}$$

Moreover, in order to guarantee ε-penetration, $\Delta t'_p$ is calculated such that the motion of $\mathfrak{A}$ starting at $t = \tau_s$ should be bounded above by ε:

$$\Delta t'_p \le \frac{\varepsilon}{\mu_u}. \tag{13}$$

Here, the undirected motion bound μ_u can be calculated similarly as in the previous section. However, there are two issues related to computing the ToV, as shown in Eq.12:

- τ_{p1} provides a lower bound of the ToV with ε-penetration, but this may be a loose bound since ε is typically much smaller than μ_u.

- The placement of the robot at $\mathfrak{A}(\tau_{p1})$ may correspond to an in-collision sample. This can be problematic for most sampling-based planners where only collision-free samples are permitted to represent the connectivity of the free C-space.

Note that the second issue is more severe than the first one in practice. We introduce an alternative way to compute τ_p to overcome these issues.

The main idea is that instead of accumulating the collision-free time steps first (*i.e.* τ_s), we intertwine collision-free and in-collision motions for every time step, just like the Boolean query in the previous section. Thus, the new ToV τ_{p2} is:

$$\tau_{p2} = \sum_i \left(\Delta t_s^i + \Delta t_p^i\right). \tag{14}$$

Here, $\Delta t_s^i, \Delta t_p^i$ for the ith iteration are calculated using Eq. 5 and Eq. 9, respectively. The above iteration continues until the ith iteration yields a collision. Thus, by construction, $\mathfrak{A}(\tau_{p2})$ is collision-free. Moreover, in general, $\tau_{p1} \leq \tau_{p2}$; however this is not always true but less likely to happen in practice since Eq. 14 continues to iterate until collision is found unlike Eq. 12, as illustrated in Fig.4.

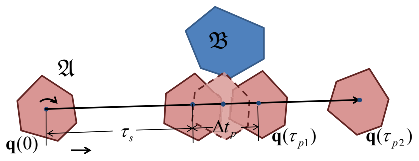


Fig. 4 Comparison between τ_{p1} and τ_{p2}. In general, $\tau_{p2} > \tau_{p1}$ since more iterations will be performed for τ_{p2} until collision is found at $\mathbf{q}(\tau_{p2})$.

6 Extension to Articulated Robots

Our CCQ algorithms for rigid robots can be extended to articulated robots. The basic equations that support CCQ algorithms such as Eqs. 6 or 12 can be reused as long as the directed and undirected motion bounds μ, μ_u can be calculated. However, this turns out to be relatively straightforward. For instance, in the case of linear motion, the directed motion bound μ for an articulated robot can be obtained using the same motion bound presented by Zhang *et al.* [30]. Moreover, the spatial and temporal culling techniques proposed in the paper to accelerate the query performance are also reusable for CCQ queries between articulated models.

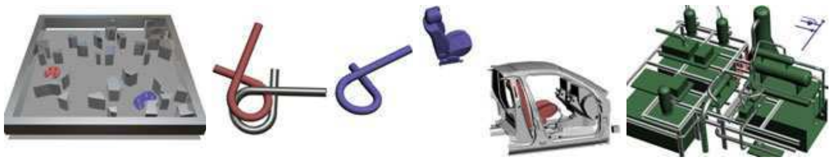
7 Results and Discussion

In this section, we describe the implementation results of our CCQ algorithms, and benchmark the performance of the algorithms by plugging them into well-known, sampling-based planners. Finally, we compare our algorithm against prior exact local planning techniques.

7.1 Implementation Details

We have implemented our CCQ algorithm using C++ on a PC running Windows Vista, equipped with Intel Dual CPU 2.40GHz and 5GB main memory. We have extended public-domain collision libraries such as PQP [12] and C^2A. Note that these collision libraries are designed only for static proximity computation or ToV computation (similar to τ_s) under a linear motion. Throughout the experiments reported in the paper, we set the penetration threshold ε for CCQ_p and τ_p as one tenth of the radius of the smallest enclosing sphere of $\mathfrak{A}$.

To measure the performance of our algorithms, we have used the benchmarking models and planning scenarios as shown in Table 2 and Fig.5 with sampling-based motion planners including PRM and RRT. These benchmarking models consist of 1K $\sim$ 30K triangles, and the test scenarios have narrow passages for the solution path. Typical query time for our CCQ algorithms takes a few milli-seconds; for instance, the most complicated benchmark, the car seat, takes 21.2 msec and 28.3 msec for ToV and Boolean queries, respectively.



(a) Maze (b) Alpha Puzzle (c) Car Seat (d) Pipe

Fig. 5 Benchmarking Scenes. For each benchmark scene, the starting and goal configurations of the robot are colored in red and blue, respectively.

Table 2 Benchmarking Model Complexities.

Benchmarks	$\mathfrak{A}$	$\mathfrak{B}$	# of tri ($\mathfrak{A}$)	# of tri ($\mathfrak{B}$)
Maze	CAD piece	Maze	2572	922
Alpha Puzzle	Alpha	Alpha	1008	1008
Car seat	Seat	Car Body	15197	30790
Pipe	Pipe	Machinery	10352	38146

7.2 Probabilistic Roadmap with CCQ

In Sec.3.2, we have explained the basic steps of sampling-based planners. These planners use a different local planning step (the step 2 in Sec. 3.2).

In conventional PRM-based planners, this Boolean checking is implemented by performing fixed-resolution collision detection along the path, namely fixed-resolution local planning (DCD). In Table 3, we show the performance of PRM with DCD with varying resolution parameters and a linear path. Here, the resolution parameter means the average number of collision checks performed for each local

Table 3 The performance of PRM in seconds based on fixed-resolution local planning (DCD) with different resolutions for the maze benchmark.

Avg. Collision Resolution	23	40	47	80	128
PRM with DCD (Boolean)	12.70s	15.88s	18.76s	39.49s	44.75s

path. We have used the OOPSMP implementation of PRM, and only the maze and pipe benchmarks were solvable by OOPSMP within a reasonable amount of time. The optimal performance is obtained when the resolution is 23, and as the resolution parameter becomes less than 23, the OOPSMP may not be able to compute a collision-free path. In any case, the DCD local planner still does not guarantee the correctness of the path in terms of collision-free motion.

However, exact local planning is made possible by running the Boolean version of our CCQ algorithm on the path. In Table 4, we highlight the performance of CCQ-based local planning algorithms (CCQ_s and CCQ_p) with PRM, and compare it against that of the DCD local planning method with the optimal resolution parameter. In case of the pipe benchmark, the PRM performance using our algorithm is similar to that of the DCD. In case of the maze benchmark, our CCQ-based local planner is about 1.8 times slower than DCD local planner. Even for this benchmark, when the resolution parameter becomes higher than 80, our CCQ algorithm performs faster than DCD, even though the DCD local planner still cannot guarantee the correctness of the solution path. Also notice that CCQ_p takes less time than CCQ_s since the former is a less restrictive query than the latter.

Table 4 The performance of PRM using DCD local planner and CCQ-based local planner. The CCQ-based local planner can guarantee collision-free motion while the other cannot give such guarantees.

Benchmark	DCD	Boolean Query	
		CCQ_s	CCQ_p
Maze	12.70s	36.34s	24.09s
Pipe	8425.09s	9610.13s	8535.60s

7.3 Rapidly-Exploring Random Tree with CCQ

Both ToV and Boolean CCQ can be employed to implement exact local planning for RRT planer. Specifically, when the new node is to be extended along some path, if the path is not collision-free, the path can be entirely abandoned (Boolean query) or the partial collision-free segment of the path before the ToV can be still kept (ToV query).

In Fig. 6, we show the performance of RRT planner with our CCQ algorithms and DCD local planner with the optimal resolution parameter. Also, different types of motion paths such as linear and screw motion have been tested. We also have used the OOPSMP implementation of RRT for this experiment. To find the optimal resolution parameter for DCD local planner, we test different resolution parameters

Table 5 The performance of RRT in seconds based on fixed-resolution local planning (DCD) with different average resolutions for the alpha puzzle benchmark. In this case, RRT uses the ToV query. When the resolution is less than 4, RRT cannot find a path

Avg. Collision Resolution	4.21	5.96	6.01	6.97
RRT with DCD (ToV)	25.60s	0.25s	2.08s	39.65s

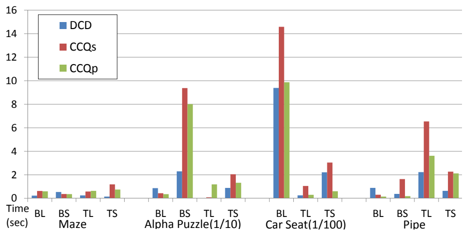


Fig. 6 The Performance of RRT using DCD and CCQ-based Local Planner. The x-axis represents different benchmarking scenes with different queries such as BL (Boolean query with a linear motion), BS (Boolean query with a screw motion), TL (ToV query with a linear motion), and TS (ToV query with a screw motion) for each benchmark. The y-axis denotes the planning time in seconds for the maze and pipe benchmark, in tens of seconds for the alpha puzzle, and in hundreds of seconds for the car seat. The blue, red and green bars denote the planning time using DCD, CCQ_s-based, and CCQ_p-based local planners, respectively.

ranging between $[3,15]$; for instance, see Table 5 for the alpha puzzle benchmark using the ToV query based on DCD local planner. Similar to PRM, the variation in performance depends on the resolution parameter, but it does not show the linear relationship between the resolution and performance unlike PRM since computing an accurate ToV using higher resolution requires many more collision checks. Thus, picking a right value for the resolution parameter is even more difficult in case of RRT.

In our benchmarks, the RRT with CCQ-based local planner is roughly two times slower than the one with DCD local planner with the optimal resolution, which defined as the minimum resolution to find a path. However, in some cases such as the Maze (BS), Alpha puzzle (BL) and pipe (BL) benchmarks in Fig.6, the RRT with CCQ-based local planner is even faster than the one with DCD local planner since the number of collision checks can be kept minimal. For the car seat benchmark, the Boolean query with a screw motion (BS) could not find out a collision-free path in a reasonable amount of time.

7.4 Comparisons with Prior Approaches

We also compare the performance of our CCQ-based local planning algorithm with the prior exact local planning algorithms such as the dynamic collision checking method (DCC) [23] implemented in MPK. To the best of our knowledge, the

Table 6 Performance Comparisons between dynamic collision checking (DCC), CCQ_s and C^2A -based local planner. The timings are the total collision checking time in seconds used for local planning.

Benchmarks	# of triangles	CCQ_s	DCC	C^2A
Pipe	48K	0.29s	1.82s	1.78s
Alpha-shape with two Holes	1K	4.5s	63.8s	17.9s

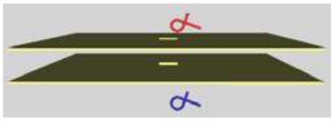


Fig. 7 Alpha-Shape Through Two Holes. The red and blue alpha shapes represent the starting and goal configurations, respectively.

dynamic collision checking algorithm is the only public-domain exact local planner that has been integrated into sampling-based motion planner.

Since DCC supports only a Boolean query with a linear motion and separation constraints, we compare the performance of the Boolean version of our CCQ_s against DCC by plugging CCQ_s into the MPK planner, as shown in Table 6. For benchmarks, we use the same pipe model in Fig.5-(d), but shrink the robot a little to enable MPK to find a solution path. We also use another benchmark model as shown in Fig. 7, the alpha-shape with two holes. In this case, we plan a path for an alpha-shape tunnelling through two holes, and measure the average performance of DCC and CCQ_s-based local planner. We also compare the CCQ_s algorithm with C^2A-based local planning algorithm [24] in two benchmarks, as shown in Table. 6.

In our experiments, CCQ_s-based local planner is about an order of magnitude faster than DCC local planner mainly because CCQ uses a tighter, directional motion bound than DCC relying on undirectional motion bound. A similar explanation was also provided in [29] why the directional bound is superior to the undirectional one. Another reason is because of the dual advancement mechanism in CCQ-based local planner. Moreover, CCQ_s is about 5 times faster than C^2A in our experiment, because of the dual advancement mechanism.

The ToV version of our CCQ_s algorithm has a similar objective as continuous collision detection algorithms. Since our algorithm is based on the known fastest CCD algorithm C^2A [24], it shows a similar performance of that of C^2A. However, C^2A is not optimized for a Boolean query and does not support CCQ with penetration constraints. Ferre and Laumond's work [4] supports a penetration query, but their work is not available freely and is essentially similar to DCC [23].

8 Conclusions

We have presented a novel proximity query, CCQ, with separation and penetration constraints. It can be used for efficient and exact local planning in sampling-based planner. In practice, we have shown that the CCQ-based local planner is only two times slower or sometimes even faster than the fixed-resolution local planner. Moreover, CCQ-based local planners outperform the state-of-the-art exact local planners by almost an order of magnitude. Our CCQ algorithm can be also extended to a more general type of motion as long as its bound can be conservatively obtained.

There are a few limitations in our CCQ algorithm. Both CCQ_s and CCQ_p algorithms are sensitive to threshold values; *e.g.* the termination condition threshold for CA or CCQ_s and penetration threshold ε for CCQ_p. The motion bound calculation such as μ or μ_u depends on the underlying path. When the robot moves with a very high rotational velocity, many CA iterations might be required to converge.

For future work, it may be possible for a planner to try different types of paths and automatically choose the suitable or optimal one. We would like to extend our CCQ framework to deformable robots. We are also interested in applying our CCQ technique to other applications such as dynamics simulation where the ToV computation is required. In particular, the use of CCQ_p may also provide a direction for contact dynamics where slight penetration is allowed (*e.g.* penalty-based method). Finally, we would like to design parallel GPU-based extension of CCQ and use it for real-time planning [17].

Acknowledgements. This research was supported in part by the IT R&D program of MKE/MCST/IITA (2008-F-033-02, Development of Real-time Physics Simulation Engine for e-Entertainment) and the KRF grant (2009-0086684). Dinesh Manocha was supported in part by ARO Contract W911NF-04-1-0088, NSF awards 0636208, 0917040 and 0904990, and DARPA/RDECOM Contract WR91CRB-08-C-0137.

References

1. Amato, N., Bayazit, O., Jones, C., Vallejo, D.: Choosing good distance metrics and local planners for probabilistic roadmap methods. IEEE Transactions on Robotics and Automation 16(4), 442–447 (2000)
2. Bohlin, R., Kavraki, L.: Path planning using Lazy PRM. In: Proceedings IEEE International Conference on Robotics & Automation, pp. 521–528 (2000)
3. Canny, J.F.: Collision detection for moving polyhedra. IEEE Trans. PAMI 8, 200–209 (1986)
4. Ferre, E., Laumond, J.-P.: An iterative diffusion algorithm for part disassembly. In: Proceedings IEEE International Conference on Robotics & Automation, pp. 3149–3154 (2004)
5. Hofer, M., Pottmann, H., Ravani, B.: Geometric design of motions constrained by a contacting surface pair. Comput. Aided Geom. Des. 20(8-9), 523–547 (2003)
6. Hsu, D.: Randomized single-query motion planning in expansive spaces. PhD thesis (2000)
7. Hsu, D., Kavraki, L.E., Latombe, J.-C., Motwani, R., Sorkin, S.: On finding narrow passages with probabilistic roadmap planners. In: International Workshop on the Algorithmic Foundations of Robotics (WAFR), pp. 141–154 (1998)

8. Isto, P.: Constructing probabilistic roadmaps with powerful local planning and path optimization. In: Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 2323–2328 (2002)
9. Ji, X., Xiao, J.: Planning motion compliant to complex contact states. International Journal of Robotics Research 20(6), 446–465 (2001)
10. Kavraki, L.E., Svestka, P., Latombe, J.-C., Overmars, M.H.: Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Transactions on Robotics & Automation 12(4), 566–580 (1996)
11. Kuffner, J.J.: Effective sampling and distance metrics for 3D rigid body path planning. In: Proceedings IEEE International Conference on Robotics & Automation, pp. 3993–3998 (2004)
12. Larsen, E., Gottschalk, S., Lin, M., Manocha, D.: Fast proximity queries with swept sphere volumes. Technical Report TR99-018, Department of Computer Science, University of North Carolina (1999)
13. Latombe, J.-C.: Robot Motion Planning. Kluwer, Boston (1991)
14. LaValle, S.M.: Planning Algorithms. Cambridge University Press, Cambridge (2006), `http://planning.cs.uiuc.edu/`
15. LaValle, S.M., Kuffner, J.J.: Rapidly-exploring random trees: Progress and prospects. In: Proceedings Workshop on the Algorithmic Foundations of Robotics, pp. 293–308 (2000)
16. Mirtich, B.V.: Impulse-based Dynamic Simulation of Rigid Body Systems. PhD thesis, University of California, Berkeley (1996)
17. Pan, J., Lauterbach, C., Manocha, D.: G-planner: Real-time motion planning and global navigation using gpus. In: AAAI Conference on Artificial Intelligence, pp. 1245–1251 (2010)
18. Redon, S., Kheddar, A., Coquillart, S.: Fast continuous collision detection between rigid bodies. In: Proc. of Eurographics (Computer Graphics Forum), pp. 279–288 (2002)
19. Redon, S., Kim, Y.J., Lin, M.C., Manocha, D.: Fast continuous collision detection for articulated models. In: Proceedings of ACM Symposium on Solid Modeling and Applications, pp. 145–156 (2004)
20. Redon, S., Lin, M.: Practical local planning in the contact space. In: Proceedings IEEE International Conference on Robotics & Automation, pp. 4200–4205 (2005)
21. Rossignac, J.R., Kim, J.J.: Computing and visualizing pose-interpolating 3d motions. Computer-Aided Design 33(4), 279–291 (2001)
22. Sánchez, G., Latombe, J.-C.: A single-query bi-directional probabilistic roadmap planner with lazy collision checking, pp. 403–417 (2003)
23. Schwarzer, F., Saha, M., Latombe, J.-C.: Adaptive dynamic collision checking for single and multiple articulated robots in complex environments. IEEE Transactions on Robotics 21(3), 338–353 (2005)
24. Tang, M., Kim, Y.J., Manocha, D.: C^{2}A: Controlled conservative advancement for continuous collision detection of polygonal models. In: Proc. of IEEE Conference on Robotics and Automation (2009)
25. Zefran, M., Kumar, V.: A variational calculus framework for motion planning. In: Proceedings IEEE International Conference on Robotics & Automation, pp. 415–420 (1997)
26. Zhang, L., Kim, Y.J., Manocha, D.: A fast and practical algorithm for generalized penetration depth computation. In: Robotics: Science and Systems (2007)
27. Zhang, L., Manocha, D.: Constrained motion interpolation with distance constraints. In: International Workshop on the Algorithmic Foundations of Robotics (WAFR), pp. 269–284 (2008)

28. Zhang, L., Manocha, D.: An efficient retraction-based RRT planner. In: IEEE International Conference on Robotics and Automation (ICRA), pp. 3743–3750 (2008)
29. Zhang, X., Lee, M., Kim, Y.J.: Interactive continuous collision detection for non-convex polyhedra. The Visual Computer, 749–760 (2006)
30. Zhang, X., Redon, S., Lee, M., Kim, Y.J.: Continuous collision detection for articulated models using taylor models and temporal culling. In: ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007), vol. 26(3), p. 15 (2007)

Modeling Contact Friction and Joint Friction in Dynamic Robotic Simulation Using the Principle of Maximum Dissipation

Evan Drumwright and Dylan A. Shell

Abstract. We present a unified treatment for modeling Coulomb and viscous friction within multi-rigid body simulation using the *principle of maximum dissipation*. This principle is used to build two different methods—an event-driven impulse-based method and a time stepping method—for modeling contact. The same principle is used to effect joint friction in articulated mechanisms. Experiments show that the contact models are able to be solved faster and more robustly than alternative models. Experiments on the joint friction model show that it is as accurate as a standard model while permitting much larger simulation step sizes to be employed.

1 Introduction

Rigid body dynamics is used extensively within robotics in order to simulate robots, learn optimal controls, and develop inverse dynamics controllers. The forward dynamics of rigid bodies in free space has been well understood for some time, and the recent advent of differential algebraic equation (DAE) based methods has made the dynamics of bodies in contact straightforward to compute as well. However, fast and stable robotic simulation remains somewhat elusive. The numerical algorithms used to compute contact forces run (on average) in time $O(n^3)$ in the number of contact points and are numerically brittle. In this paper, we present a class of methods for modeling contact with friction in multi-rigid body simulation that is not only faster empirically than existing methods, but is solvable with numerical robustness. We also extend our approach to modeling joint friction and show how it is at least as accurate as standard joint friction models, while permitting much larger simulation step sizes (and thus much greater simulation speed).

Evan Drumwright
George Washington University
e-mail: `edrumwri@gmail.com`

Dylan A. Shell
Texas A&M University
e-mail: `dshell@cs.tamu.edu`

Our approach centers around the *principle of maximal dissipation*. Paraphrasing [44], the principle of maximal dissipation states that, for bodies in contact, the friction force is the one force (of all possible friction forces) that maximizes the rate of energy dissipation. In this paper, we show that we can use the principle of maximal dissipation to implicitly solve for frictional forces, in contrast to prior approaches that set the frictional force by explicitly using the direction of relative motion. By employing this strategy, we are able to dispense with the complementarity constraints that are nearly universally employed and instead formulate the contact problem using a convex optimization model.

2 Methods for Modeling Contact with Friction in Multi-rigid Body Simulation

2.1 Background

As stated in the previous section, modeling contact with friction in multi-rigid body simulation has been extensively conducted using complementarity constraints. Such constraints, which take the form $\mathbf{a} \geq 0, \mathbf{b} \geq 0, \mathbf{a}^{\mathsf{T}}\mathbf{b} = 0$, have been utilized in multi-rigid body simulation to ensure that forces are not applied at contact points at which bodies are separating [4], that either sticking or sliding friction is applied [4], and that forces are applied only for joints at their limits [28]. A non-exhaustive survey of the literature dedicated to modeling contact with complementarity constraints includes [26, 32, 27, 33, 34, 31, 9, 2, 38, 11, 51, 4, 6, 45, 5, 3, 50, 1, 39, 18, 7, 37, 49, 48]. Initial efforts on modeling contact with friction [26, 27, 9] attempted to solve a linear complementarity problem (LCP) for the unknown forces and accelerations. Later work—under which researchers realized that solving for forces and accelerations could be subject to *inconsistent configurations*[1] [8]—solved instead for unknown impulsive forces and velocities; this approach was able to avoid the problem of inconsistent configurations. Both iterative sequential impulse schemes (*e.g.*, [30, 22]) and simultaneous impulse-based methods (*e.g.*, [4, 45]) were employed. The former methods have been viewed as *splitting methods* [15] for solving linear complementarity problems by Lacoursière, who reported slow convergence rates for coupled problems using such approaches [24]. Correspondingly, much of the multi rigid-body simulation community currently uses explicit linear or nonlinear complementarity problem formulations (Anitescu and Potra [4], most prominently) with impulses and velocities in order to model contact with friction.

Fig. 1 illustrates why complementarity conditions are necessary at the acceleration level; further discussion of their requirement at the acceleration level is present in [9] and [14]. Given that using impulsive forces is necessary to avoid the problem of inconsistent configurations (exemplified by the problem of [36]), it must be discerned whether the complementarity conditions are accurate and appropriate in that new context (*i.e.*, at the velocity level). We stress that the context is indeed new,

[1] An inconsistent configuration is a contact configuration that has no solution using non-impulsive forces.

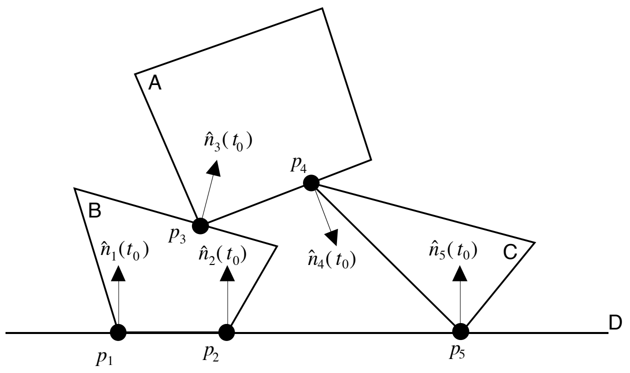


Fig. 1 A figure (taken from [10]) illustrating the necessity of complementarity constraints for resting (non-impacting, non-separating) contact at the acceleration level. Complementarity constraints would ensure that body A does not move upward unnaturally fast if a force were applied to A– say by a strong wind moving between B and A– accelerating A upward. In the absence of such a force, the complementarity condition keeps blocks A and B from interpenetrating.

because only "resting" (zero relative normal velocity) contacts are treated with forces at the acceleration level[2] while both resting and impacting contact are treated with impulses at the velocity level.

With respect to accuracy, [14] argues that the complementarity conditions do not necessarily reflect reality for impacting contacts. He conducts an experiment using carbon paper, a coin, and a hammer, that serves to prove his argument: the physical accuracy of complementarity-based models, at least in some scenarios, is lacking. Further physical experimentation is warranted, especially given the dominance of complementarity-constrained-methods at the velocity level in the literature, but if we accept that nature does not require complementarity constraints, do such conditions lead to models that are more advantageous in some other way (*e.g.*, computationally?)

Linear and nonlinear complementarity problem-based models are, in fact, inferior with respect to computation, at least relative to the models introduced in this paper. Solutions to bisymmetric LCPs—the form that many models take—are NP-hard in the worst case in the number of contact points (though the expected time solution is $O(n^3)$ [15]). The contact models are non-convex, so only a few algorithms are capable of solving LCP-based contact models; codes for solving the NCP-based models are even more rare.[3] The LCP-based models linearize the friction cone, and the

[2] Unless a penalty method is used; such methods are irrelevant to this discussion because we are assuming non-interpenetrating contact.

[3] The reader may question the desire to have multiple algorithms capable of solving a contact model. Optimization algorithms can be numerically brittle, so using multiple algorithms can reduce the likelihood of a simulation failing to progress.

fidelity of the friction cone approximation increases the size and, correspondingly, the computational cost of the model to be solved.

Given that complementarity conditions may not be physically warranted and that such models can be hard to solve computationally, what is the reason behind their strong popularity? We believe this quote from [14] is instructive:

> It is emphasized that many authors in this area are aware of the possible physical inaccuracies behind the complementarity assumption. For example, [32] states that a primary benefit of such a formulation is internal mathematical consistency, and empirical corrections towards better accuracy can be accommodated later. Baraff (personal communication) mentions that there is no reason to think that real systems obey the complementarity conditions. However, in Pfeiffer and Glocker's [38] discussion of the "corner law of contact dynamics" there is unfortunately no explicit mention of the possible lack of physical realism behind the complementarity conditions in the presence of impacts. [4] focus on mathematical aspects of their formulation, and also omit discussion of physical realism. The authors of the latter two authoritative works may therefore unintentionally convey an inaccurate impression to a reader who is new to the field.

2.2 Two Representations for the Contact Models

Given that complementarity conditions are not a necessary feature of contact models, we now proceed to present our complementarity-free contact models. We will use two representations for these contact models. Both representations have been employed previously in the literature. We utilize the two representations here in order to make the community aware of their existence, to unify them, and, hopefully, to expose them to further study (for determination of computational or numerical advantages, for example).

A / b representation

The first representation formulates the contact model in terms of matrices **A** and **b** (alternatively named **K** and **u** by [30]). This representation has been employed by [9], [30], [23], and [17]. In this representation, **b** is a vector of relative velocities in the $3n$-dimensional contact space and **A** is the $3n \times 3n$ sized *contact space inertia matrix* that transforms impulsive forces to relative velocities in contact space. **A** is dense, symmetric, and positive-semi definite; the latter two properties were proven by [30]. The matrices **A** and **b** are related by the equation:

$$\mathbf{b}^{+} = \mathbf{A}\mathbf{x} + \mathbf{b} \tag{1}$$

where **b** is the vector of relative velocities pre-contact (external forces, such as gravity, are integrated into this vector), **x** is the vector of impulses applied in the contact space, and $\mathbf{b}^{+}$ is the vector of relative velocities after impulses are applied.

The matrices **A** and **b** can be determined formulaically if the contacting bodies are not articulated, and via application of test impulses otherwise (*cf.*, [30, 23]). By using test impulses, the contact model can remain ignorant of whether the bodies are articulated and, if articulated, of whether the bodies are formulated using maximal or reduced coordinates. This is an advantage of the **A** / **b** representation; this representation also tends to produce simpler—though not necessarily computationally advantageous—objective and constraint function gradients and Hessians for solving the contact model.

Generalized coordinate representation

The generalized coordinate representation has been used in work by [4] and [45], among others. This representation uses matrices **M** (the generalized inertia matrix), **N** (the contact normal Jacobian), **J** (the joint constraint Jacobian); vectors $\mathbf{c}_n$ (the contact normal impulse magnitudes), $\mathbf{c}_j$ (the joint constraint impulse magnitudes), **q** (the generalized coordinates), **v** (the generalized velocities) and **k** (the generalized external forces), and scalar h (the step size).

For models with complementarity constraints, additional matrices are used, both to enforce the complementarity constraints and to provide a linearized friction cone; we do not list such matrices here. Because our model does not employ complementarity constraints, it is able to provide a true friction cone and still remain computationally tractable. The true friction cone is obtained using Jacobian matrices corresponding to the two tangential directions at each contact normal; we denote these matrices **S** and **T** and the corresponding contact tangent impulses as $\mathbf{c}_s$ and $\mathbf{c}_t$.

All of the matrices described above are related using the following formulae:

$$\mathbf{v}^{t+1} = \mathbf{M}^{-1}(\mathbf{N}^\mathsf{T}\mathbf{c}_n + \mathbf{S}^\mathsf{T}\mathbf{c}_s + \mathbf{T}^\mathsf{T}\mathbf{c}_t + \mathbf{J}^\mathsf{T}\mathbf{c}_j + h\mathbf{k}) + \mathbf{v}^t \quad (2)$$

$$\mathbf{q}^{t+1} = \mathbf{q}^t + h\mathbf{v}^{t+1} \quad (3)$$

The second equation reflects that this representation is typically utilized in a semi-implicit integration scheme.

Note that **M** is symmetric, positive-definite. **M**, **N**, **S**, **T** and **J** are sparse and correspondingly make efficient determination of the objective and constraint gradients and Hessians conceptually more involved than with the **A** / **b** representation. Computing **M**, **N**, **S**, **T** and **J** is quite simple, however, and computationally far more efficient than the test impulse method. We show below, however, that **A** and **b** can also be determined efficiently using the matrices from the generalized coordinate representation.

Unification of the A / b and generalized coordinate representations

We may write **b** and **A** using the generalized coordinate representation as

$$\mathbf{b} = \begin{bmatrix}\mathbf{N}\\ \mathbf{S}\\ \mathbf{T}\end{bmatrix}(\mathbf{v}^t + h\mathbf{M}^{-1}\mathbf{k}), \qquad \mathbf{A} = \begin{bmatrix}\mathbf{N}\\ \mathbf{S}\\ \mathbf{T}\end{bmatrix}\mathbf{C}\begin{bmatrix}\mathbf{N}^\mathsf{T} & \mathbf{S}^\mathsf{T} & \mathbf{T}^\mathsf{T}\end{bmatrix},$$

where $\mathbf{C} \triangleq \mathbf{M}^{-1} - \mathbf{M}^{-1}\mathbf{J}^\mathsf{T}(\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^\mathsf{T})^{-1}\mathbf{J}\mathbf{M}^{-1}$.

These equations show that recovering **N**, **S**, **T**, and **J** from **A** and **b** is not possible.

2.3 The Contact Models for Event-Driven and Time-Stepping Simulation

Computer-based rigid body simulation methods have been categorized into three schemes by Brogliato et al. [12]: penalty, event-driven, and time-stepping. Given that penalty methods necessarily cannot enforce non-interpenetration, we focus instead on event-driven and time-stepping methods. The former are able to utilize arbitrary integration schemes (time-stepping methods are frequently restricted to semi-implicit Euler integration) while the latter aim to be able to avoid (possibly nonexistent) accuracy problems due to continually restarting the integration process; [12] provide greater detail of the rationale behind development of the time-stepping approaches. We note that [16] has shown that one purported advantage of time-stepping approaches over event-driven simulations—avoidance of Zeno points—is nonexistent. Sec. 2.4 and 2.5 present both event-driven and time-stepping methods in order to show that our method is applicable to both. Sec. 2.4 uses the $\mathbf{A}/\mathbf{b}$ representation, while Sec. 2.5 uses the generalized coordinate representation. The choice of representation used for the individual methods is arbitrary: the two representations are interchangeable.

2.4 Event-Driven Impulse-Based Method

[30] defines the work done by collision impulses using the $\mathbf{A}/\mathbf{b}$ representation as $\frac{1}{2}\mathbf{x}^{\mathsf{T}}(\mathbf{A}\mathbf{x}+2\mathbf{b})$. We can use the principle of maximal dissipation to determine the set of impulses that maximally dissipate kinetic energy. In particular, we can formulate and solve the following optimization problem.

Quadratic Program 1

Minimize $\frac{1}{2}\mathbf{x}^{\mathsf{T}}(\mathbf{A}\mathbf{x}+2\mathbf{b})$

Subject to:

$\begin{bmatrix}\mathbf{A}_N & \mathbf{B}\end{bmatrix}\mathbf{x}+\mathbf{b}_N \geq 0$ (Noninterpenetration constraint)

$\mathbf{x}_N \geq 0$ (Compressive force constraint)

$\kappa \geq \mathbf{1}^{\mathsf{T}}\mathbf{x}_N$ (Friction cone constraint)

$\mu_c^2\mathbf{x}_{N_i}^2 + \mu_v^2(\mathbf{b}_{T_{1i}}^2+\mathbf{b}_{T_{2i}}^2) \geq \mathbf{x}_{T_{1i}}^2+\mathbf{x}_{T_{2i}}^2 \quad \forall i \in 1\ldots n.$ (Coulomb/viscous friction constraint)

where $\mathbf{A} = \begin{bmatrix}\mathbf{A}_N & \mathbf{B}\\ \mathbf{B}^{\mathsf{T}} & \mathbf{A}_T\end{bmatrix}$ is partitioned into $n\times n$ block $\mathbf{A}_N$, $n\times 2n$ block $\mathbf{B}$, and $2n\times 2n$ block $\mathbf{A}_T$. Similarly, $\mathbf{x}$ and $\mathbf{b}$ are partitioned into $\mathbf{x} = \begin{bmatrix}\mathbf{x}_N & \mathbf{x}_{T_1} & \mathbf{x}_{T_2}\end{bmatrix}^{\mathsf{T}}$ and $\mathbf{b} = \begin{bmatrix}\mathbf{b}_N & \mathbf{b}_{T_1} & \mathbf{b}_{T_2}\end{bmatrix}^{\mathsf{T}}$, respectively. Scalar κ is the sum of the normal impulses that describe the minimum kinetic energy solution when the tangential impulses are zero; this scalar is described further in Sec. 2.6.

2.5 Time Stepping Method

Equivalent in spirit to the event-driven method is the time-stepping method, which, like that of [4], is a semi-implicit scheme and, like that of [45], adds contact constraint stabilization to the dynamics equations.

Quadratic Program 2

Minimize $\frac{1}{2}\mathbf{v}^{t+1\mathsf{T}}\mathbf{M}\mathbf{v}^{t+1}$

Subject to:

$\mathbf{N}\mathbf{v}^{t+1} \geq 0$ (Noninterpenetration constraint)

$\mathbf{J}\mathbf{v}^{t+1} = 0$ (Bilateral joint constraint)

$\mathbf{c}_n \geq 0$ (Compressive force constraint)

$\mathbf{1}^{\mathsf{T}}\mathbf{c}_n \leq \kappa$ (Friction cone constraint)

$$\mu_c^2\mathbf{c}_{n_i}^2 + \mu_v^2[\mathbf{S}_i(\mathbf{v}^t + h\mathbf{M}^{-1}\mathbf{k})]^2 + \mu_v^2[\mathbf{T}_i(\mathbf{v}^t + h\mathbf{M}^{-1}\mathbf{k})]^2 \geq \mathbf{c}_{s_i}^2 + \mathbf{c}_{t_i}^2 \quad \forall i \in 1\ldots n.$$

(Coulomb/viscous friction constraint)

where $\mathbf{S}_i$ and $\mathbf{T}_i$ refer to the i^{th} row of $\mathbf{S}$ and $\mathbf{T}$, respectively. Unlike the event-driven method, the time-stepping method includes a constraint for bilateral joints in case the method is used with maximal-coordinate formulated articulated bodies; the $\mathbf{A}/\mathbf{b}$ representation implicitly encodes such constraints into the matrix / vector formulations.

2.6 Solving the Models

Both of the contact models introduced in the previous sections are convex and can be solved in polynomial time in the number of contacts using interior-point methods. Determining the value κ requires solving the models in an initial, frictionless phase (phase I). Next, a frictional phase (phase II) is solved. If normal restitution is necessary, a third phase is required as well.[4] Fortunately, phase I is a quadratic programming model with box constraints, and can thus be solved extremely quickly using a gradient projection method [35]. The model for phase I ($\mathbf{A}/\mathbf{b}$ formulation) is:

[4] We omit details of this third phase but refer the reader to [4], which describes a Poisson-type restitution model that is applicable to our models also.

Quadratic Program 3

Minimize	$\frac{1}{2}\mathbf{x}_N^{\mathsf{T}}\mathbf{A}_N\mathbf{x}_N + \mathbf{x}_N^{\mathsf{T}}\mathbf{b}_N$	
Subject to:		
	$\mathbf{x}_N \geq 0$	(Compressive force constraint)

Given that $\mathbf{A}_N$ is symmetric and positive semi-definite (follows from symmetry and positive semi-definiteness of $\mathbf{A}$), this model describes a convex linear complementarity problem (see [15], p. 5). As a result, the constraint $\mathbf{A}_N\mathbf{x}_N + \mathbf{b}_N \geq 0$ (which is equivalent to our non-interpenetration constraint) is automatically satisfied at the optimum. This convex LCP always has a solution and we prove that this solution does not increase the energy in the system (see Appendix). As phase II cannot increase the energy in the system our contact models are energetically consistent.

Phase I has two objectives: determine an energetically consistent, *feasible point*[5] for the nonlinear Quadratic Program 1 and determine κ. The value κ is determined by calculating the sum $\mathbf{1}^{\mathsf{T}}\mathbf{x}_N$, using the result from phase I. Observe that the friction cone inequality constraint in Quadratic Program 1 restricts the sum of the normal impulses to that determined in phase I; thus, phase II can reorder, transfer, or remove some normal force, but the friction cone is prevented from becoming enlarged arbitrarily (*i.e.*, the normal forces cannot be increased without bound to increase the amount of frictional force applicable) in order to decrease the kinetic energy more rapidly.

We note that, although we use a linear complementarity problem formulation to show that our contact models are energetically consistent, our contact models do not use complementarity constraints: neither phase I nor phase II of the contact models explicitly include any such constraint.

3 Evaluating the Contact Models

There does not currently exist a standardized set of benchmark models for evaluating the performance and accuracy of contact models. In order to evaluate contact models, previous approaches have used either physical experimentation on a single benchmark scenario (*e.g.*, [46]) or pathological computer-based scenarios (*e.g.*, [30, 40, 13]) that are known to exhibit certain qualitative behavior in the real world.

Like [4], we do not provide exhaustive experimentation to indicate the predictive abilities of our contact model. Instead, we note that our model possesses the following properties, which are also possessed by leading alternative contact models that treat multiple contact points simultaneously (*e.g.*, [4, 45]):

1. Positive work is not done by the model (energy is not added to the system).
2. Interpenetration constraints are not violated.

[5] A feasible point is one which respects all inequality constraints. In the case of our contact model, a feasible point will respect non-interpenetration, compressive normal force, summed normal force, and Coulomb and viscous friction constraints.

3. Only compressive normal forces are applied.
4. The friction cone is not enlarged artificially.
5. The principle of maximal dissipation is obeyed.

Given that these are the main characterizations of both systems, we can expect the emergent behavior of both the complementarity-based methods and our non-complementarity-based methods to be similar. Indeed, as Fig. 2 shows, the method of Anitescu and Potra (solved using Lemke's algorithm [25, 15]) produces identical results (to numerical precision) to our non-complementarity-based model on at least one scenario.

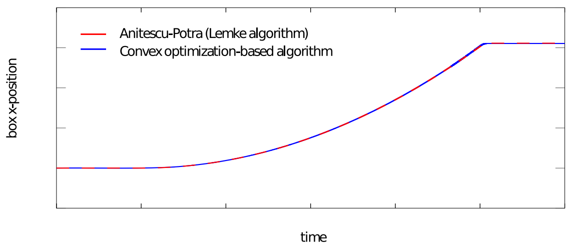


Fig. 2 Plot illustrating accuracy of a box sliding down a ramp under both the method of [4] and the convex optimization-based method introduced in this paper. Although the two models are formulated quite differently, the simulated results are identical.

We do not claim that our method is more accurate than that of Anitescu and Potra or Stewart and Trinkle (notwithstanding the linearized friction cones generally employed by those two methods). The advantage of our method lies in the computational domain. Lacking complementarity constraints, our model is solvable faster than competing methods; additionally, the gradient projection method we use to solve the first phase of our algorithm—recall that the first phase of our algorithm finds a point that respects properties (1)–(4) above—never fails to produce a solution; failure in phase II of our method will only affect the accuracy of the solution and will not lead to interpenetration or positive work (energy gain).

The experiments below showcase these advantages of our method.

4 Experiments

4.1 Event-Driven Example: Internal Combustion Engine

We constructed an inline, four cylinder internal combustion engine in order to illustrate the ability of our method to treat moderate numbers of contacts (between one and two hundred) far faster than complementarity-based models. All joints within the simulated engine were realized by contact constraints only. The engine was simulated by applying a torque to the crankshaft, which caused the cylinders to move upward and downward within the crankcase. The crankcase and crankshaft guides are stationary within the simulation—they possess infinite inertia, so they do not move dynamically—and the remaining parts are all of mass $1.0kg$. Although the

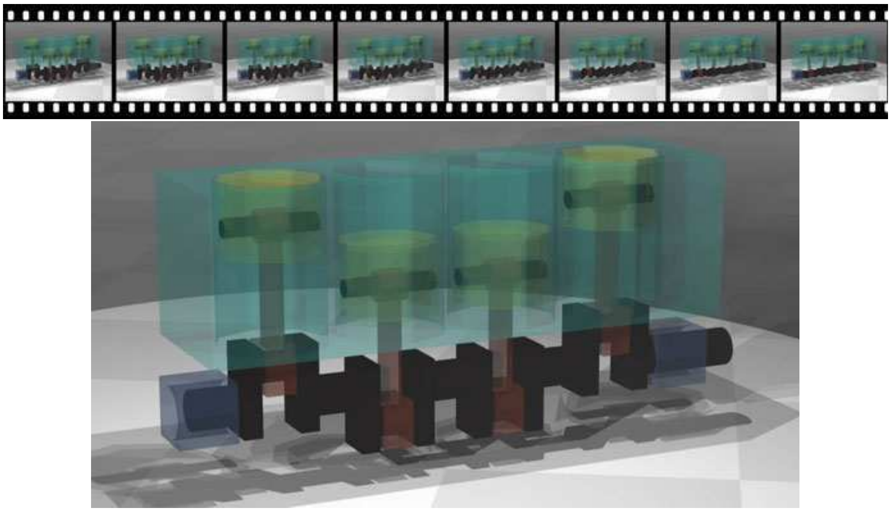


Fig. 3 Frames in sequence from a simulation of the rigid-body dynamics and interactions within an internal combustion engine

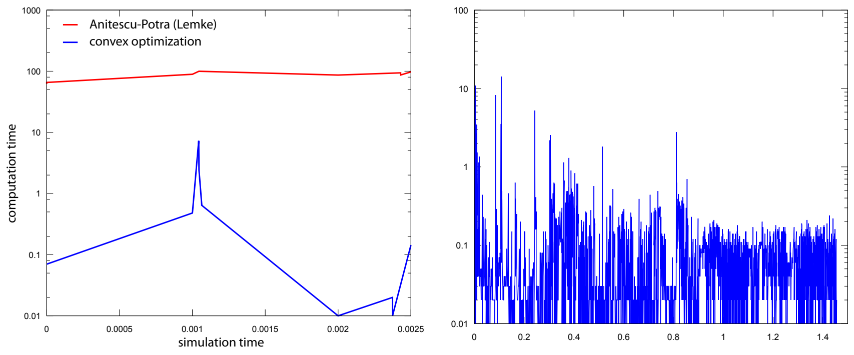
masses do not reflect reality, the moment-of-inertia matrix for each part is calculated using its triangle mesh geometry via the method of [29]. Gravity acts along the vertical direction of the simulation at $9.8m/s^2$ and the engine surfaces are given zero coefficients of Coulomb and viscous friction: we initially wanted to judge how rapidly the polygonal-based geometric representation causes energy loss.

For purposes of comparison, we used the method of [4] with the simplest (*i.e.*, pyramidal) friction model. Using the pyramidal friction model resulted in six LCP variables per contact; thus, LCPs of order between 600 and 1200 were generated. We point out that– although the contact was frictionless– neither model (*i.e.*, neither our complementarity-free model nor that of Anitescu and Potra) took advantage of that fact: the full frictional models (with zero coefficients of friction) were used.

4.2 Time-Stepping Example: Granular Matter

We simulated 1001 spheres of radius 0.04m dropping into and settling within a box to illustrate the feasibility of our method on large scale simulations. We note that similar simulations have been conducted at even larger scales by [47]; however, that work is less general and exhibits several features (*e.g.*, permits interpenetration, non-Coulomb friction model) that limit its applicability outside of granular matter simulation. Figs. 6 and 7 show several snapshots taken from the simulation and depict the rapid evolution of the system.

As Fig. 5 indicates, we tested our time-stepping method against a time stepping implementation of the contact model of [4]; we used two different LCP solvers



(a) Timings of both methods (0–0.0025s) (b) Convex optimization method timing (0-1s)

Fig. 4 Computation timings required to solve the contact model for a single cylinder of the internal combustion engine using both the method of [4] and the method introduced in this paper. Timings for the Anitescu-Potra method are only provided to 0.0025 seconds of simulation time; the Lemke-based solver [19] could not solve the LCP problem to sufficient tolerances to continue the simulation past that point. The PATH solver [21] was able to complete only a single iteration of the simulation, and is thus not included in the comparison.

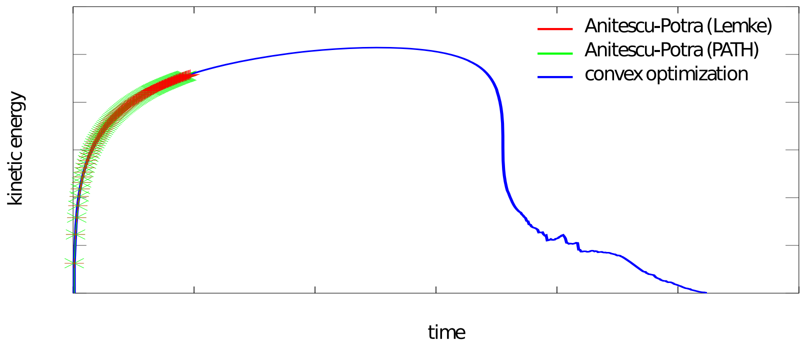


Fig. 5 Kinetic energy of the granular simulation over approximately five seconds of simulation time, plotted for the Anitescu-Potra [4] model (using two solvers) and our maximum-dissipation-based model. The lack of robustness in the linear complementarity problem solvers is evident here, as neither solver for the Anitescu-Potra model was able to simulate to one second of simulation time. The identical system energy for all three approaches (up to the failure of the Anitescu-Potra solvers) provides some evidence that the models are generating identical results.

(LEMKE [19] and PATH [21]) though both exhibited issues with robustness. As in the previous experiment, the pyramidal friction model was used to effect minimum computation time.

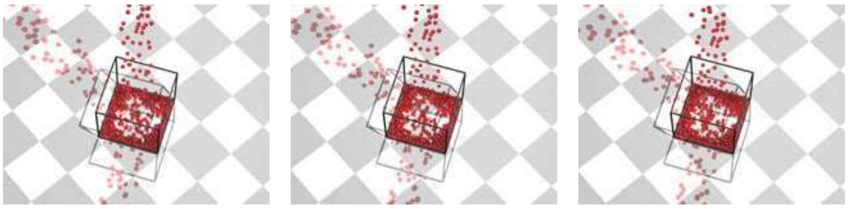


Fig. 6 Frames from time $t = 1.95s$, $t = 1.96s$, and $t = 1.97s$ show the granules dropping into the box.

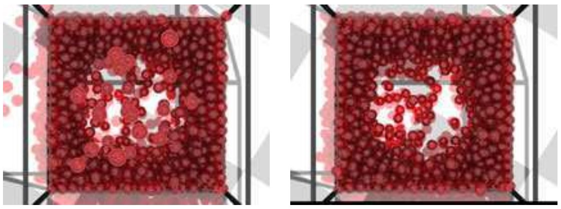


Fig. 7 Frames from time $t = 3.50s$, and $t = 3.55s$ depict the granules settling within the box.

5 Modeling Joint Friction for Articulated Bodies Formulated in Reduced Coordinates

Many robotics applications simulate articulated bodies in reduced coordinates [42] for several reasons. The reduced coordinate representation is more amenable to producing matrices useful to roboticists (*e.g.*, the joint space inertia matrix, the end-effector Jacobian matrix, *etc.*) and does not require tweaking parameters to minimize joint constraint violations. The reduced coordinate formulation admits generally simple formulae for modeling Coulomb and viscous friction at robot joints. From [41], p. 141, the torques at joints due to joint friction can be modeled as:

$$\tau_\mu = \mu_v \dot{\mathbf{q}} + \mu_c \,\mathbf{sgn}(\dot{\mathbf{q}}) \tag{4}$$

where μ_v and μ_c are the coefficients for viscous and Coulomb friction, respectively.

The issue with this model is that it tends to make the differential equations *stiff* (*i.e.*, difficult to solve numerically) using even moderately large values of μ_c and μ_v; this statement is particularly true for μ_c, which uses the discontinuous signum function. The practical effect of this issue is that either extremely small integration steps must be taken or the joint friction must be poorly modeled. We can, however, use the principle of maximum dissipation and velocity-level dynamics equations to model friction properly; Sec. 5.3 will show that our approach models the viscous component of Equation 4 exactly (for sufficiently small coefficients of friction or step sizes of the latter), and that the Coulomb component of Equation 4 asymptotically approaches our model as the integration step tends to zero.

5.1 *Maximum Dissipation-Based Joint Friction Model*

Under the principle of maximum dissipation, we wish to find the impulses that minimize the new kinetic energy. The change in joint velocity is given by the formula $\Delta\dot{\mathbf{q}} = \mathbf{H}^{-1}\mathbf{x}$, where $\dot{\mathbf{q}}$ is the joint-space velocity, $\mathbf{H}$ is the joint-space inertia matrix, and $\mathbf{x}$ is the vector of applied impulses. Thus we wish to minimize the quantity $\frac{1}{2}(\dot{\mathbf{q}}+\Delta\dot{\mathbf{q}})^{\mathsf{T}}\mathbf{H}(\dot{\mathbf{q}}+\Delta\dot{\mathbf{q}})$ subject to Coulomb and viscous constraints on the applied impulses $\mathbf{x}$.

Quadratic Program 4

Minimize $\frac{1}{2}\mathbf{x}^{\mathsf{T}}\mathbf{H}^{-1}\mathbf{x}+\mathbf{x}^{\mathsf{T}}\dot{\mathbf{q}}$

Subject to: $-\mu_c\mathbf{1}-\mu_v\dot{\mathbf{q}} \leq \mathbf{x} \leq \mu_c\mathbf{1}+\mu_v\dot{\mathbf{q}}$ (Joint friction model)

5.2 *Solving the Joint Friction Model*

The joint friction model provided above is a quadratic program with box constraints. Due to the symmetry and positive semi-definiteness of $\mathbf{H}$ (see [20]), the quadratic

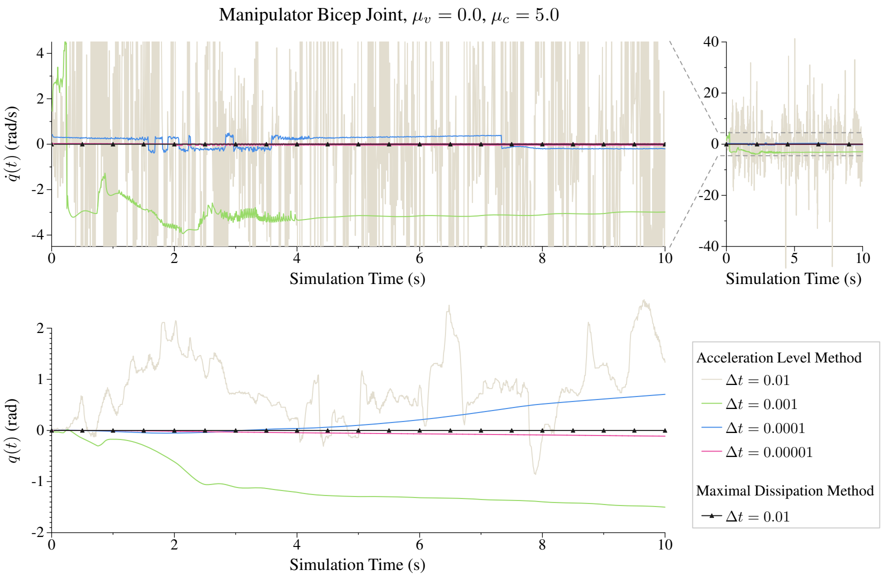


Fig. 8 Plots show that joint friction is comparable between the maximal-dissipation-based and acceleration level methods; the former method produces this behavior at a step size three orders of magnitude larger than the latter. The correct behavior is for the joint position (*bottom*) and velocity (*top*) to remain at zero, given that the coefficient of Coulomb friction is so large (5.0). Note that the joint velocity for the acceleration level method is unstable for $\Delta t = 0.01$, relatively large for $\Delta t = 0.001$, relatively small for $\Delta t = 0.0001$, and near zero for $\Delta t = 0.00001$.

program is convex, and thus a global minimum can be found in polynomial time. In fact, programs with hundreds of variables—well within the range of the joint spaces of all robots produced to date—can be solved within microseconds using a gradient projection method [35].

5.3 Empirical Results

Figs. 8 and 9 depict the efficacy of using the quadratic program defined above for effecting joint friction. We simulated an anthropomorphic, eight degree-of-freedom (DOF) manipulator arm acting only under the influence of gravity from an initial position. In Fig. 8, coefficients of joint friction of $\mu_c = 5.0$ and $\mu_v = 0.0$ were used, while the coefficients of joint friction were $\mu_c = 0.25$ and $\mu_v = 0.1$ in Fig. 9. Two methods were used to model joint friction: the acceleration level method described in [41] (*i.e.*, Equation 4) and the quadratic programming method based on the principle of maximal dissipation described above.

Although Figs. 8 and 9 depict only two DOFs of the robot arm (the bicep and shoulder, respectively), the results in the plots are indicative of all the DOFs for the robot: the acceleration level method converges to the maximal dissipation-based method as the step size for the former becomes sufficiently small. Indeed, the top plots in the two figures (*i.e.*, the joint velocities) indicate the result for the acceleration level method when the step size is not sufficiently small: the simulation

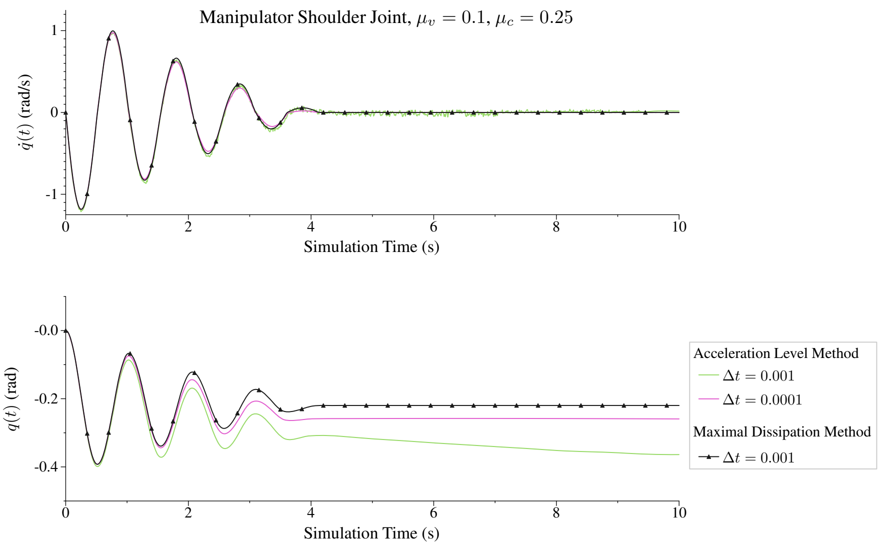


Fig. 9 Plots show that joint friction is comparable between the maximal-dissipation-based and acceleration level methods; the former method produces this behavior at a larger step size than the latter. Note that the joint velocity for the acceleration level method oscillates notably for $\Delta t = 0.001$ and these effects are reduced for $\Delta t = 0.0001$.

becomes unstable. Our results indicate that the quadratic programming model based on the principle of maximal dissipation is as accurate as an accepted model of joint friction but that the former admits simulation step sizes several orders of magnitude higher (and, correspondingly, simulation several orders of magnitude faster).

6 Conclusions

Roboticists are very familiar with complementarity-based contact models like that of [4]; such models have been incorporated into popular simulators like ODE [43]. Consequently, the perception of the accuracy (and inaccuracy) of such models has been informed by considerable practice. The complementarity-free models that were introduced in this paper do not possess such a track record, and direct, empirical comparison between such models is the subject of future work. Nevertheless, the principle of maximal dissipation is accepted by the applied mechanics community, and we have shown evidence that—at minimum—this principle can be used to simulate plausibly mechanisms, granular matter, and joint friction. If the accuracy of the complementarity-free models proves acceptable, the computational advantages intrinsic to our models will yield considerable speedups in robotic simulation. Finally, we argue that, even if the models presented in this paper are found to be poorly predictive (compared to complementarity-based models), Chatterjee's work makes it clear that non-complementarity-based contact models should be investigated further.

References

1. Anitescu, M.: Optimization-based simulation of nonsmooth dynamics. Mathematical Programming, Series A 105, 113–143 (2006)
2. Anitescu, M., Cremer, J., Potra, F.: Properties of complementarity formulations for contact problems with friction. In: Ferris, M., Pang, J.-S. (eds.) Complementarity and Variational Problems: State of the Art, pp. 12–21. SIAM, Philadelphia (1997)
3. Anitescu, M., Hart, G.: A constraint-stabilized time-stepping approach for rigid multibody dynamics with joints, contacts, and friction. Intl. J. for Numerical Methods in Eng. 60(14), 2335–2371 (2004)
4. Anitescu, M., Potra, F.: Formulating dynamic multi-rigid-body contact problems with friction as solvable linear complementarity problems. Nonlinear Dyn. 14, 231–247 (1997)
5. Anitescu, M., Potra, F.: A time-stepping method for stiff multi-rigid-body dynamics with contact and friction. Intl. J. for Numerical Methods in Eng. 55, 753–784 (2002)
6. Anitescu, M., Potra, F., Stewart, D.: Time-stepping for three dimensional rigid body dynamics. Computer Methods in Applied Mechanics and Eng. 177, 183–197 (1999)
7. Anitescu, M., Tasora, A.: An iterative approach for cone complementarity problems for nonsmooth dynamics. Computational Optimization and Applications (2008)
8. Baraff, D.: Coping with friction for non-penetrating rigid body simulation. Computer Graphics 25(4), 31–40 (1991)

9. Baraff, D.: Fast contact force computation for nonpenetrating rigid bodies. In: Proc. of SIGGRAPH, Orlando, FL (1994)
10. Baraff, D.: An introduction to physically based modeling: Rigid body simulation II – constrained rigid body dynamics. Technical report, Robotics Institute, Carnegie Mellon University (1997)
11. Brogliato, B.: Nonsmooth Impact Mechanics: Models, Dynamics, and Control. Springer, London (1996)
12. Brogliato, B., ten Dam, A.A., Paoli, L., Génot, F., Abadie, M.: Numerical simulation of finite dimensional multibody nonsmooth mechanical systems. ASME Appl. Mech. Reviews 55(2), 107–150 (2002)
13. Ceanga, V., Hurmuzlu, Y.: A new look at an old problem: Newton's cradle. ASME J. Appl. Mech. 68, 575–583 (2001)
14. Chatterjee, A.: On the realism of complementarity conditions in rigid-body collisions. Nonlinear Dyn. 20, 159–168 (1999)
15. Cottle, R., Pang, J.-S., Stone, R.E.: The Linear Complementarity Problem. Academic Press, Boston (1992)
16. Drumwright, E.: Avoiding Zeno's paradox in impulse-based rigid body simulation. In: Proc. of IEEE Intl. Conf. on Robotics and Automation (ICRA), Anchorage, AK (2010)
17. Drumwright, E., Shell, D.: A robust and tractable contact model for dynamic robotic simulation. In: Proc. of ACM Symp. on Applied Computing (SAC) (2009)
18. Erleben, K.: Velocity-based shock propagation for multibody dynamics animation. ACM Trans. on Graphics 26(12) (2007)
19. Fackler, P., Miranda, M.: Implementation of a modified lemke's complementary pivoting algorithm (2002), `http://people.sc.fsu.edu/~burkardt/m_src/lemke/lemke.m`
20. Featherstone, R.: Robot Dynamics Algorithms. Kluwer, Dordrecht (1987)
21. Ferris, M., Munson, T.: Complementarity problems in GAMS and the PATH solver. J. of Economic Dynamics and Control 24(2), 165–188 (2000)
22. Guendelman, E., Bridson, R., Fedkiw, R.: Nonconvex rigid bodies with stacking. ACM Trans. on Graphics 22(3), 871–878 (2003)
23. Kokkevis, E.: Practical physics for articulated characters. In: Proc. of Game Developers Conf. (2004)
24. Lacoursière, C.: Splitting methods for dry frictional contact problems in rigid multibody systems: Preliminary performance results. In: Ollila, M. (ed.) Proc. of SIGRAD, pp. 11–16 (2003)
25. Lemke, C.: Bimatrix equilibrium points and mathematical programming. Management Science 11, 681–689 (1965)
26. Löstedt, P.: Mechanical systems of rigid bodies subject to unilateral constraints. SIAM J. on Applied Mathematics 42(2), 281–296 (1982)
27. Löstedt, P.: Numerical simulation of time-dependent contact friction problems in rigid body mechanics. SIAM J. of Scientific Statistical Computing 5(2), 370–393 (1984)
28. Miller, A., Christensen, H.: Implementation of multi-rigid-body dynamics within a robotic grasping simulator. In: Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA), pp. 2262–2268 (2003)
29. Mirtich, B.: Fast and accurate computation of polyhedral mass properties. J. of Graphics Tools 1(2) (1996)
30. Mirtich, B.: Impulse-based Dynamic Simulation of Rigid Body Systems. PhD thesis, University of California, Berkeley (1996)

31. Monteiro-Marques, M.: Differential inclusions in nonsmooth mechanical problems: Shocks and dry friction. In: Progress in Nonlinear Differential Equations and Their Applications, vol. 9, Birkhäuser, Verlag (1993)
32. Moreau, J.: Standard inelastic shocks and the dynamics of unilateral constraints. In: Unilateral problems in structural analysis, pp. 173–221. Springer, New York (1983)
33. Moreau, J.: Standard inelastic shocks and the dynamics of unilateral constraints. In: C.I.S.M. Courses and Lectures, vol. 288, pp. 173–221. Springer, Vienna (1985)
34. Moreau, J.: Unilateral contact and dry friction in finite freedom dynamics. In: Nonsmooth Mechanics and Applications, pp. 1–82. Springer, Heidelberg (1988)
35. Nocedal, J., Wright, S.: Numerical Optimization, 2nd edn. Springer, Heidelberg (2006)
36. Painlevé, P.: Sur le lois du frottement de glissemment. C. R. Académie des Sciences Paris 121, 112–115 (1895)
37. Petra, C., Gavrea, B., Anitescu, M., Potra, F.: A computational study of the use of an optimization-based method for simulating large multibody systems. In: Optimization Methods and Software (2009)
38. Pfeiffer, F., Glocker, C.: Dynamics of rigid body systems with unilateral constraints. John Wiley and Sons, Chichester (1996)
39. Potra, F., Anitescu, M., Gavrea, B., Trinkle, J.: A linearly implicit trapezoidal method for stiff multibody dynamics with contact, joints, and friction. Intl. J. for Numerical Methods in Eng. 66(7), 1079–1124 (2006)
40. Sauer, J., Schoemer, E., Lennerz, C.: Real-time rigid body simulations of some "classical mechanics toys". In: Proc. of European Simulation Symposium and Exhibition, Nottingham, UK (1998)
41. Sciavicco, L., Siciliano, B.: Modeling and Control of Robot Manipulators, 2nd edn. Springer, London (2000)
42. Shabana, A.: Computational Dynamics, 2nd edn. John Wiley & Sons, Chichester (2001)
43. Smith, R.: ODE: Open Dynamics Engine (2001),`http://www.ode.org` (Last access: July 2010)
44. Stewart, D.: Rigid-body dynamics with friction and impact. SIAM Review 42(1), 3–39 (2000)
45. Stewart, D., Trinkle, J.: An implicit time-stepping scheme for rigid body dynamics with coulomb friction. In: Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA), San Francisco, CA (2000)
46. Stoianovici, D., Hurmuzlu, Y.: A critical study of the applicability of rigid-body collision theory. ASME J. Appl. Mech. 63, 307–316 (1996)
47. Tasora, A., Anitescu, M.: A convex complementarity approach for simulating large granular flow. J. of Computational Nonlinear Dynamics 5(3) (2010)
48. Tassa, Y., Todorov, E.: Stochastic complementarity for local control of continuous dynamics. In: Proc. of Robotics: Science and Systems (2010)
49. Todorov, E.: Implicit nonlinear complementarity: a new approach to contact dynamics. In: Proc. of Intl. Conf. on Robotics and Automation ICRA (2010)
50. Trinkle, J., Berard, S., Pang, J.S.: A time-stepping scheme for quasistatic multibody systems. In: Proc. of Intl. Symp. on Assembly and Task Planning, pp. 174–181 (2005)
51. Trinkle, J., Pang, J.-S., Sudarsky, S., Lo, G.: On dynamic multi-rigid-body contact problems with coulomb friction. Zeithscrift fur Angewandte Mathematik und Mechanik 77(4), 267–279 (1997)

A The LCP Determined Feasible Point Is Energetically Consistent

We now prove that the solution determined in phase I in Sec. 2.6 is energetically consistent. Working from [30], the equation for work done by collision impulses is $\frac{1}{2}\mathbf{z}^{\mathsf{T}}(\mathbf{A}\mathbf{z}+2\mathbf{b})$.

Given that $\mathbf{z} \triangleq \begin{bmatrix} \mathbf{y} \\ \mathbf{0} \end{bmatrix}$, we prove $\frac{1}{2}\mathbf{y}^{\mathsf{T}}(\mathbf{A}_N\mathbf{y}+2\mathbf{b}_N) \leq 0$.
The proof relies upon the linear complementarity solver finding the solution $\mathbf{y}$, which yields $\mathbf{y}^{\mathsf{T}}(\mathbf{A}_N\mathbf{y}+\mathbf{b}_N) = 0$. Given the linear complementarity condition $\mathbf{y}^{\mathsf{T}}(\mathbf{A}_N\mathbf{y}+\mathbf{b}_N) = 0$, we are left with

$$\mathbf{y}^{\mathsf{T}}\mathbf{b}_N \leq 0.$$

The above equation must hold, because $\mathbf{y}^{\mathsf{T}}(\mathbf{A}_N\mathbf{y}+\mathbf{b}_N) = 0$ and because $\mathbf{A}_N$ is symmetric, positive semi-definite, which implies that $\mathbf{y}^{\mathsf{T}}\mathbf{A}_N\mathbf{y} \geq 0$ (and therefore, that $\mathbf{y}^{\mathsf{T}}\mathbf{b}_N \leq 0$).

Energy-Based Modeling of Tangential Compliance in 3-Dimensional Impact

Yan-Bin Jia

Abstract. This paper studies modeling of tangential compliance as two rigid bodies collide in the space. Stronge's spring-based contact structure [13, pp. 95-96] is extended to three dimensions. Slip or stick is indicated by the tangential motion of a massless particle connected to the contact point (viewed as an infinitesimal region) on one body via three orthogonal springs. We show that the effect of tangential compliance can be analyzed using normal impulse rather than time, contrary to a previous claim by Stronge. This is primarily due to the ability of updating the elastic energies of the three springs without knowledge of their stiffnesses or length changes. The change rates, nevertheless, are computable. So are sliding velocity and tangential impulse. The latter is then integrated into impact equations and contact kinematics, making the whole system driven by normal impulse alone. Examples include a ball and a pencil bouncing on a table, and a massé billiard shot. The theory has potential impact on impulsive robotic manipulation in which the ability to deal with friction and compliance is vital for skillful maneuvers.

1 Introduction

Impulse-based manipulation is an area in robotics where very little work [6, 14] is known. An impulsive force has very short execution time, and thus good potential for improving task efficiency. Its use could considerably simplify the robotic mechanism needed to perform a manipulation task, while avoiding uncertainties accumulated over repeated complex operations. The primary reason for the lack of research attention is possibly because the foundation of modeling rigid body impact is not fully developed and the existing theories often seem either too simple to be realistic or too complex to be applicable, especially in the presence of friction and compliance, not to mention nonlinear

Yan-Bin Jia
Department of Computer Science, Iowa State University
e-mail: jia@cs.iastate.edu

viscoelastic effects. Discrepancies often exist between an introduced theory and the findings from an experiment intended for its validation.

Before presenting some related work on impact mechanics, we give a brief review of rigid body impact. Suppose during an impact the ith contact force $\boldsymbol{f}_i$ is applied at the location $\boldsymbol{r}_i$ on a body. Integration of the acceleration equation $\dot{\boldsymbol{V}} = d\boldsymbol{V}/dt = \sum_i \boldsymbol{f}_i/m$ over the impact duration Δt yields the total impulse $\sum_i \boldsymbol{I}_i = m\Delta\boldsymbol{V}$. Similarly, integrating the angular acceleration equation $\sum_i \boldsymbol{r}_i \times \boldsymbol{f}_i = Q\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times Q\boldsymbol{\omega}$, where Q is the body's angular inertia matrix, we obtain $\sum_i \boldsymbol{r}_i \times \boldsymbol{I}_i = Q\Delta\boldsymbol{\omega}$ since $\boldsymbol{\omega}$ is bounded and $\Delta t \to 0$. The following linear impact equations relate the body's velocities to individual impulses:

$$\Delta V = \frac{1}{m}\sum_i \boldsymbol{I}_i \quad \text{and} \quad \Delta\boldsymbol{\omega} = Q^{-1}\sum_i (\boldsymbol{r}_i \times \boldsymbol{I}_i). \tag{1}$$

An impulse $\boldsymbol{I}$ can be decomposed into a component of magnitude I_n along the contact normal and a tangential component $\boldsymbol{I}_\perp$. The normal impulse I_n increases during both compression and restitution phases of impact. The ratio of its amount of accumulation during restitution to that during compression is a constant under Poisson's hypothesis. In solving an impact problem, I_n is often treated as the variable [9] with whose growth the velocities, the contact mode, and the impact phase are updated.

The tangential impulse $\boldsymbol{I}_\perp$, meanwhile, depends on the sequence of contact modes that occur during the impact. If the contact is sliding, the differential accumulations $d\boldsymbol{I}_\perp$ and dI_n are related under Coulomb's law of friction, with the former opposing the instantaneous slip direction. If the contact is sticking, $d\boldsymbol{I}_\perp$ is in a direction to counter the tendency of slip. As the direction varies, the tangential impulse accumulates along a plane curve, and the total impulse along a space curve. A closed-form solution rarely exists.

Efforts on impact analysis have struggled over the consistencies between laws of Coulomb's friction and energy conservation, and Poisson's impulse-based hypothesis of restitution. Routh's graphical method [10] to construct the impulse trajectory has proven successful for analyzing 2-dimensional impacts, and has been later extended by various researchers [4, 15, 1]. For 3-dimensional impact, Darboux [3] was the first to describe impact dynamics in terms of normal impulse in the form of a differential equation. His result was later rediscovered by Keller [9] who also used the equation's solution to determine the varying slip direction.

The above efforts have neglected the effect of tangential compliance and assumed that all work done by the tangential reaction force is lost to friction. When tangential compliance is not negligible, however, part of the work is converted into recoverable internal energy, despite the loss of the remaining part to friction. Approaches [2, 11], designed to produce a ratio of tangential to normal impulse equal to the coefficient of friction, did not exactly follow Coulomb's law of friction. Stronge [13] developed a lumped parameter representation of compliance, and applied a time-dependent analysis to track the change in the tangential velocity during a collision. His model could predict slip or stick at the contact under Coulomb's law. However, without knowing the duration of impact, the analysis can only be used to perceive contact modes qualitatively rather than to carry out specific computation.

Computation of tangential impulse is the key to solving an impact problem and the focus of this paper. We extend the structure of Stronge's linear model of planar impact with compliance [13, pp. 95-96] to develop a theory for 3-dimensional impact that is based on normal impulse only and consistent with both laws of Coulomb friction and energy conservation.

2 Tangential Impulse

During a collision of two bodies, the gravitational forces are negligible compared to the contact force. The configuration can be oriented to keep the contact tangent plane horizontal. To model tangential compliance, we first extend the planar contact structure used by Stronge [13, pp. 95–96] to three dimensions. The "contact point" on the upper body does not directly touch the lower body but is rather connected to a massless particle p via three springs respectively aligned with the upward normal $\hat{\boldsymbol{n}}$ and two orthogonal tangential directions $\hat{\boldsymbol{u}}$ and $\hat{\boldsymbol{w}}$, all unit vectors, as shown in Fig. 1. The direction $\hat{\boldsymbol{u}}$ is chosen to oppose the tangential component of the initial contact velocity $\boldsymbol{v}_0$, and $\hat{\boldsymbol{w}} = \hat{\boldsymbol{n}} \times \hat{\boldsymbol{u}}$. All velocities will be measured along these directions but not relative to p which may move during impact.

The contact force $\boldsymbol{F}$ on the upper body is decomposed as $\boldsymbol{F} = F_u\hat{\boldsymbol{u}} + F_w\hat{\boldsymbol{w}} + F_n\hat{\boldsymbol{n}}$, with each component exerted by one of the three springs. The impulse $\boldsymbol{I} = \int \boldsymbol{F}\,dt$ also has components I_n, I_u, I_w, respectively in the three orthogonal directions; namely,

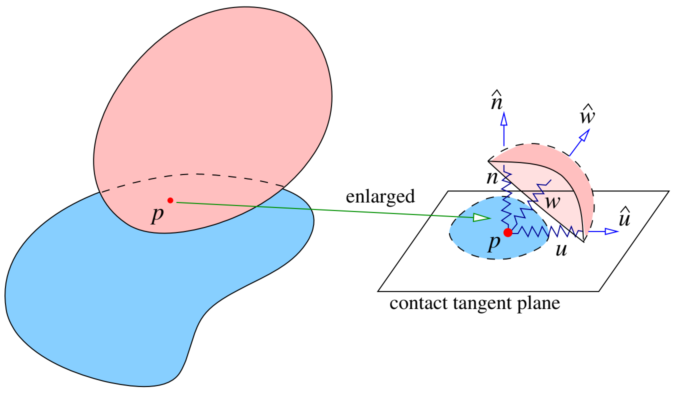


Fig. 1 Compliance model for 3-dimensional impact. The contact point, initially coinciding with the particle p, is blown up into a small region (shown on the right) connected to p via three springs.

$$\boldsymbol{I} = I_n \hat{\boldsymbol{n}} + I_u \hat{\boldsymbol{u}} + I_w \hat{\boldsymbol{w}}. \tag{2}$$

2.1 Impact Model

The impact starts with compression and ends with restitution. In the compression phase, the normal spring compresses and its elastic energy E_n builds up while the normal component v_n of the contact velocity $\boldsymbol{v}$ decreases. Compression ends when $v_n = 0$; at this moment E_n has the maximum value, say, $E_{\max}$. During restitution, the normal spring extends, releasing an amount $e^2 E_{\max}$ of the energy, where $e \in [0,1]$ is referred to as the *energetic coefficient of restitution.* The conventional kinetic coefficient of restitution, introduced by Poisson as the ratio between the normal impulse released during restitution to that accumulated during compression, is not consistent with energy conservation when the direction of contact slip varies during collision [13, p. 47].

We adopt Stronge's explanation [13, p. 96] for the energy loss $(1-e^2)E_{\max}$: at the moment restitution starts, the normal stiffness suddenly scales up $1/e^2$ times. Namely, the normal stiffness k during the impact is given by

$$k = \begin{cases} k_0, & \text{compression,} \\ k_0/e^2, & \text{restitution.} \end{cases} \tag{3}$$

where k_0 is the original stiffness. Meanwhile, the change n in the normal spring length suddenly varies by a factor of e^2. The normal contact force F_n nevertheless stays the same at this phase transition.

The two tangential springs have the same stiffness $k_\perp$ which is invariant during the impact. The ratio $\eta_0^2 = k_0/k_\perp$ is often considered a constant that depends on the Young's moduli and the Poisson's ratios of the materials in contact.[1] In the analysis below, we will use the ratio

$$\eta^2 = k/k_\perp, \tag{4}$$

where $\eta = \eta_0$ during compression and $\eta = \eta_0/e$ during restitution.

Denote by n, u, w the *changes of length* of the three springs, and E_n, E_u, E_w the elastic energies they store, respectively. The contact force components and the elastic energies are given below:

$$F_n = -kn \geq 0, \quad F_u = -k_\perp u, \quad F_w = -k_\perp w; \tag{5}$$

$$E_n = \frac{1}{2}kn^2, \qquad E_u = \frac{1}{2}k_\perp u^2, \quad E_w = \frac{1}{2}k_\perp w^2. \tag{6}$$

[1] For normal indentation by a rigid circular punch on an elastic half space, Johnson [8, pp. 361–366] showed that $\eta_0^2 = \frac{2-\nu}{2(1-\nu)}$, where ν is the Poisson's ratio of the half space. For most materials, this ratio ranges between 0 and 0.5 (Wikipedia).

Impulse Derivatives

Our objective is to describe the entire system in terms of the normal impulse I_n. To avoid any ambiguity, from now on the notation '$\dot{}$' will refer to differentiation with respect to time, while the notation '$'$' will refer to differentiation with respect to I_n (which monotonically increases from zero during impact).

Combining the first equations in (5) and (6), the change rate of the normal impulse I_n over time can be described in terms of the elastic energy E_n:

$$\dot{I}_n = dI_n/dt = F_n = \sqrt{2kE_n}. \tag{7}$$

The derivative is well-defined at the impact phase transition where F_n stays continuous. Meanwhile, from $\dot{n} = v_n$, the normal contact velocity, and $\dot{I}_n = F_n = -kn$ we obtain that $\dot{E}_n = kn\dot{n} = -v_n\dot{I}_n$, thereby the derivative

$$E'_n = dE_n/dI_n = \dot{E}_n/\dot{I}_n = -v_n. \tag{8}$$

Similarly, from the other two pairs of equations in (5) and (6) we obtain the change rates of the two tangential impulses:

$$\dot{I}_u = F_u = -\alpha\sqrt{2k_\perp E_u} \quad \text{and} \quad \dot{I}_w = F_w = -\beta\sqrt{2k_\perp E_w}, \tag{9}$$

where α and β are the signs of the length changes of the u- and w-springs, i.e.,

$$\alpha = \begin{cases} 1 \text{ if } u \geq 0, \\ -1 \text{ if } u < 0; \end{cases} \qquad \beta = \begin{cases} 1 \text{ if } w \geq 0, \\ -1 \text{ if } w < 0. \end{cases} \tag{10}$$

Equation (7) is important because it allows us to convert a derivative with respect to time into one with respect to the normal impulse I_n simply by a division over $\sqrt{2kE_n}$. As an illustration, we have

$$I'_u = \frac{\dot{I}_u}{\dot{I}_n} = -\alpha\sqrt{\frac{k_\perp E_u}{kE_n}} = -\frac{\alpha}{\eta}\sqrt{\frac{E_u}{E_n}} \quad \text{and} \quad I'_w = -\frac{\beta}{\eta}\sqrt{\frac{E_w}{E_n}}. \tag{11}$$

In fact, the stiffnesses k and $k_\perp$ will always occur together in the ratio form.

2.2 Contact Modes

The contact velocity $\boldsymbol{v}$ of the two bodies is obtained from their velocities and angular velocities as well as the locations of contact on each body, based on the contact kinematics. This will be illustrated in the examples in Section 3. For now we just assume that $\boldsymbol{v}$ is provided, and denote its tangential component as $\boldsymbol{v}_\perp$. Then the velocity of the particle p is

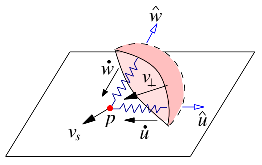


Fig. 2 Sliding velocity of the contact particle p.

$$\boldsymbol{v}_s = \boldsymbol{v}_\perp - \dot{u}\hat{\boldsymbol{u}} - \dot{w}\hat{\boldsymbol{w}}. \tag{12}$$

When $\boldsymbol{v}_s = 0$, i.e., $\boldsymbol{v}_\perp = \dot{u}\hat{\boldsymbol{u}} + \dot{w}\hat{\boldsymbol{w}}$, the contact sticks. In other words, the relative motion of the upper body to the lower body in the contact plane is completely absorbed by the two tangential springs so that p has no motion. When $\boldsymbol{v}_s \neq 0$, it is the *sliding velocity* of the contact.

When slip happens, under Coulomb's law, the tangential contact force $\boldsymbol{F}_\perp = -\mu F_n \hat{\boldsymbol{v}}_s$, where μ is the friction coefficient[2] and

$$\hat{\boldsymbol{v}}_s = \frac{\boldsymbol{v}_s}{\|\boldsymbol{v}_s\|} = \frac{\boldsymbol{v}_\perp - \dot{u}\hat{\boldsymbol{u}} - \dot{w}\hat{\boldsymbol{w}}}{\|\boldsymbol{v}_s\|}, \qquad \text{from (12).} \tag{13}$$

Since the force also exerts on the u- and w-springs, we obtain

$$k_\perp(u\hat{\boldsymbol{u}} + w\hat{\boldsymbol{w}}) = \mu F_n \hat{\boldsymbol{v}}_s, \tag{14}$$

Substitute (13) into (14), and rearrange the terms slightly:

$$\boldsymbol{v}_\perp - \dot{u}\hat{\boldsymbol{u}} - \dot{w}\hat{\boldsymbol{w}} = \frac{k_\perp}{\mu F_n}\|\boldsymbol{v}_s\|(u\hat{\boldsymbol{u}} + w\hat{\boldsymbol{w}}). \tag{15}$$

Equation (15) also holds under the sticking contact since it reduces to $\boldsymbol{v}_\perp = \dot{u}\hat{\boldsymbol{u}} + \dot{w}\hat{\boldsymbol{w}}$ when $\boldsymbol{v}_s = 0$. Take dot products of both sides of (15) with $\hat{\boldsymbol{u}}$, and then multiply by w. Similarly, take dot products with $\hat{\boldsymbol{w}}$ and multiply by u. Subtracting the two resulting equations, we have, under both contact modes,

$$w\dot{u} - u\dot{w} = (\boldsymbol{v}_\perp \cdot \hat{\boldsymbol{u}})w - (\boldsymbol{v}_\perp \cdot \hat{\boldsymbol{w}})u. \tag{16}$$

The contact between the two bodies sticks when $\sqrt{F_u^2 + F_w^2} < \mu F_n$, namely, by (9), when

$$\sqrt{\dot{I}_u^2 + \dot{I}_w^2} < \mu \dot{I}_n. \tag{17}$$

To replace the time derivatives, we substitute equations (7) and (9) into the above, and rearrange the resulting terms after squaring both sides:

$$E_u + E_w < \mu^2\eta^2 E_n. \tag{18}$$

When the contact slips, we have the equality

$$E_u + E_w = \mu^2\eta^2 E_n. \tag{19}$$

2.3 Stick

Since $\boldsymbol{v}_s = 0$, the lengths of the u- and w-springs change at rates

[2] The difference between static and dynamic coefficients is ignored so the value of μ stays constant whether the contact sticks or slips.

$$\dot{u} = \boldsymbol{v}_\perp \cdot \hat{\boldsymbol{u}} = \boldsymbol{v} \cdot \hat{\boldsymbol{u}} \quad \text{and} \quad \dot{w} = \boldsymbol{v} \cdot \hat{\boldsymbol{w}}. \tag{20}$$

Their length changes are

$$u = \int \dot{u}\, dt = \int \frac{\dot{u}}{\dot{I}_n}\, dI_n = \int \frac{\dot{u}}{\sqrt{2kE_n}}\, dI_n = \frac{1}{\sqrt{2k}} \int \frac{\boldsymbol{v}_\perp}{\sqrt{E_n}}\, dI_n \cdot \hat{\boldsymbol{u}},$$
$$w = \frac{1}{\sqrt{2k}} \int \frac{\boldsymbol{v}_\perp}{\sqrt{E_n}}\, dI_n \cdot \hat{\boldsymbol{w}}.$$

Suppose compression ends at $I_n = I_c$ and restitution ends at $I_n = I_r$. Observing (3), we introduce a vector integral

$$\boldsymbol{D} = \begin{cases} \int_0^{I_n} \frac{\boldsymbol{v}_\perp}{\sqrt{E_n}}\, dI_n, & \text{if } I_n \in [0, I_c); \\ \int_0^{I_c} \frac{\boldsymbol{v}_\perp}{\sqrt{E_n}}\, dI_n + \int_{I_c}^{I_n} e \frac{\boldsymbol{v}_\perp}{\sqrt{E_n}}\, dI_n, & \text{if } I_n \in [I_c, I_r], \end{cases} \tag{21}$$

so that during impact the following always hold:

$$u = \frac{\boldsymbol{D} \cdot \hat{\boldsymbol{u}}}{\sqrt{2k_0}} \quad \text{and} \quad w = \frac{\boldsymbol{D} \cdot \hat{\boldsymbol{w}}}{\sqrt{2k_0}}. \tag{22}$$

Instead of computing u and w, we keep track of $\sqrt{2k_0}u$ and $\sqrt{2k_0}w$ by updating $\boldsymbol{D}$, without any knowledge about k_0.

The update of $\boldsymbol{D}$ is possible because $\boldsymbol{v}_\perp$ is from the contact kinematics, and E_n is by (6). The values of α and β in (10) are immediately known from the signs of $\boldsymbol{D} \cdot \hat{\boldsymbol{u}}$ and $\boldsymbol{D} \cdot \hat{\boldsymbol{w}}$. The integral is also used to conveniently evaluate the tangential elastic energies, for

$$E_u = \frac{1}{2} k_\perp u^2 = \frac{1}{4} \frac{k_\perp}{k_0} (\boldsymbol{D} \cdot \hat{\boldsymbol{u}})^2 = \frac{1}{4\eta_0^2} (\boldsymbol{D} \cdot \hat{\boldsymbol{u}})^2 \text{ and } E_w = \frac{1}{4\eta_0^2} (\boldsymbol{D} \cdot \hat{\boldsymbol{w}})^2. \tag{23}$$

2.4 Slip

When the contact slips, equation (19) holds. We substitute the spring energies E_u and E_w from (6) in and obtain, by the use of (4),

$$u^2 + w^2 = 2\mu^2 \frac{k}{k_\perp^2} E_n. \tag{24}$$

Then differentiate (24) with respect to time:

$$u\dot{u} + w\dot{w} = \mu^2 \frac{k}{k_\perp^2} \dot{E}_n. \tag{25}$$

Now, we can solve the spring velocities $\dot{u}$ and $\dot{w}$ from (16) and (25):

$$\dot{u} = \frac{\alpha\mu^2\eta^3 E_n'\sqrt{E_n E_u} + (\boldsymbol{v}_\perp \cdot \hat{\boldsymbol{u}})E_w - \alpha\beta(\boldsymbol{v}_\perp \cdot \hat{\boldsymbol{w}})\sqrt{E_u E_w}}{\mu^2\eta^2 E_n}, \tag{26}$$

$$\dot{w} = \frac{\beta\mu^2\eta^3 E_n'\sqrt{E_n E_w} + (\boldsymbol{v}_\perp \cdot \hat{\boldsymbol{w}})E_u - \alpha\beta(\boldsymbol{v}_\perp \cdot \hat{\boldsymbol{u}})\sqrt{E_u E_w}}{\mu^2\eta^2 E_n}. \tag{27}$$

With $\dot{u}$ and $\dot{w}$ known, the sliding velocity $\boldsymbol{v}_s$ follows from (12).

The change rates (26) and (27) do not tell whether the springs are being compressed (e.g., $u < 0$) or elongated (e.g., $u > 0$). Since

$$u = \int \dot{u}\, dt = \int \frac{\dot{u}}{\sqrt{2kE_n}}\, dI_n \quad \text{and} \quad w = \int \frac{\dot{w}}{\sqrt{2kE_n}}\, dI_n, \tag{28}$$

we introduce two integrals G_u and G_w, where for $\rho = u, v$,

$$G_\rho = \begin{cases} \int_0^{I_n} \frac{\dot{\rho}}{\sqrt{E_n}}\, dI_n, & \text{if } I_n \in [0, I_c), \\ \int_0^{I_c} \frac{\dot{\rho}}{\sqrt{E_n}}\, dI_n + \int_{I_c}^{I_n} e\frac{\dot{\rho}}{\sqrt{E_n}}\, dI_n, & \text{if } I_n \in [I_c, I_r]; \end{cases} \tag{29}$$

Comparing these two equations with (28), $G_u = \sqrt{2k_0}u$ and $G_w = \sqrt{2k_0}w$.

The two integrals G_u and G_w are used to not only track the signs of u and w but also update tangential elastic energies as follows:

$$E_u = \frac{1}{2}k_\perp u^2 = \frac{G_u^2}{4\eta_0^2} \quad \text{and} \quad E_w = \frac{G_w^2}{4\eta_0^2}. \tag{30}$$

2.5 Contact Mode Transitions

At a contact mode switch, we need to initialize the integrals $\boldsymbol{D}$ or G_u and G_w in order to track whether the tangential springs are being compressed or extended and update E_u and E_w during the next contact mode.

Stick to Slip

The contact point switches its mode when $F_u^2 + F_w^2 = \mu^2 F_n^2$, i.e., when (19) holds. We initialize the integrals for slip using (30):

$$G_u = 2\alpha\eta_0\sqrt{E_u} \quad \text{and} \quad G_w = 2\beta\eta_0\sqrt{E_w}, \tag{31}$$

where E_u, E_w, α and β inherit their values from just before the change of contact.

Slip to Stick

The contact switches from slip to stick when the sliding velocity $\boldsymbol{v}_s$ vanishes, that is, when

$$\boldsymbol{v}_\perp = \dot{u}\hat{\boldsymbol{u}} + \dot{w}\hat{\boldsymbol{w}} \tag{32}$$

from (12). Equations (22) imply that

$$\begin{aligned}\boldsymbol{D} = \sqrt{2k_0}(u\hat{\boldsymbol{u}} + w\hat{\boldsymbol{u}}) &= \sqrt{2k_0}\left(\alpha\sqrt{\frac{2E_u}{k_\perp}}\hat{\boldsymbol{u}} + \beta\sqrt{\frac{2E_w}{k_\perp}}\hat{\boldsymbol{w}}\right) \quad \text{by (6) and (10)} \\ &= 2\eta_0\left(\alpha\sqrt{E_u}\hat{\boldsymbol{u}} + \beta\sqrt{E_w}\hat{\boldsymbol{w}}\right). \qquad (33)\end{aligned}$$

2.6 Impact Algorithm

The system of impact equations does not have a closed-form solution in general. Simulation is carried out via numerical integration over I_n with some step size, say, h. The pseudo-code is given below.

```
initialization
$dI_n \leftarrow h$
**while** (compression **or** $E_n > 0$) **do**
   **if** $\boldsymbol{v} \cdot \hat{\boldsymbol{n}} = 0$
      **then** compression ends
   **if** contact sticks
      **then** update $\boldsymbol{D}$ according to (21) and $E_u, E_w$ according to (23)
              **if** slip starts by (19)
                 initialize $G_u$ and $G_w$ as (31)
      **else**  evaluate $\dot{u}$, $\dot{v}$ according to (26), (27)
              update $G_u$, $G_w$ according to (29) and $E_u, E_w$ according to (30)
              **if** stick starts by (32)
                 initialize $\boldsymbol{D}$ as (33)
   evaluate $I'_u$ and $I'_w$ as (11)
   $\boldsymbol{I} \leftarrow \boldsymbol{I} + dI_n \cdot (I'_u\hat{\boldsymbol{u}} + I'_w\hat{\boldsymbol{w}} + \hat{\boldsymbol{n}})$
   update $\boldsymbol{V}$ and $\boldsymbol{\omega}$ using impact equations (1)
   update $\boldsymbol{v}$ using $\boldsymbol{V}$ and $\boldsymbol{\omega}$ according to contact kinematics
   $E_n \leftarrow E_n - v_n dI_n$ by (8)
```

2.7 Impact Initialization

To start the algorithm, we need to initialize the contact mode, the normal and tangential elastic energies, the integral $\boldsymbol{D}$ or G_u, G_w, accordingly, and the tangential impact. Let the initial contact velocity be $\boldsymbol{v}_0 = v_{0n}\hat{\boldsymbol{n}} + \boldsymbol{v}_{0\perp}$. By (8) we have that $E'_n(0) = -v_{0n}$ and $E_n(h) \approx -v_{0n}h$.

Stick or Slip?

We first assume that the impact starts with stick, and apply our analysis from Section 2.3 to derive a condition on $\boldsymbol{v}_0$. Here we look at a small period of time Δt after the impact begins. The force on the u-spring is

$$\dot{I}_u = -k_\perp u = -k_\perp \int_0^{\Delta t} \dot{u}\, dt = -k_\perp \int_0^{\Delta t} (\boldsymbol{v} \cdot \hat{\boldsymbol{u}})\, dt, \quad \text{by (20)}. \qquad (34)$$

Similarly, we obtain the forces exerted by the other two springs:

$$\dot{I}_w = -k_\perp \int_0^{\Delta t} (\boldsymbol{v} \cdot \hat{\boldsymbol{w}})\, dt \quad \text{and} \quad \dot{I}_n = -k_0 \int_0^{\Delta t} (\boldsymbol{v} \cdot \hat{\boldsymbol{n}})\, dt. \tag{35}$$

Substitute the above three time derivatives into the sticking contact condition (17) and move the integral over $(\boldsymbol{v} \cdot \hat{\boldsymbol{n}})$ to the left side of the resulting inequality:

$$\begin{aligned} \lim_{\Delta t \to 0} \frac{\| \int_0^{\Delta t} (\boldsymbol{v} \cdot \hat{\boldsymbol{u}})\hat{\boldsymbol{u}} + (\boldsymbol{v} \cdot \hat{\boldsymbol{w}})\hat{\boldsymbol{w}}\, dt \|}{-\int_0^{\Delta t} (\boldsymbol{v} \cdot \hat{\boldsymbol{n}})\, dt} &= \lim_{\Delta t \to 0} \frac{\int_0^{\Delta t} \sqrt{(\boldsymbol{v} \cdot \hat{\boldsymbol{u}})^2 + (\boldsymbol{v} \cdot \hat{\boldsymbol{w}})^2}\, dt}{-\int_0^{\Delta t} (\boldsymbol{v} \cdot \hat{\boldsymbol{n}})\, dt} \\ &= \frac{\sqrt{(\boldsymbol{v}_0 \cdot \hat{\boldsymbol{u}})^2 + (\boldsymbol{v}_0 \cdot \hat{\boldsymbol{w}})^2}}{-(\boldsymbol{v}_0 \cdot \hat{\boldsymbol{n}})} < \mu \frac{k_0}{k_\perp} = \mu \eta_0^2. \end{aligned}$$

The first equation above follows from that $\boldsymbol{v} \cdot \hat{\boldsymbol{u}}$ and $\boldsymbol{v} \cdot \hat{\boldsymbol{w}}$ do not changes signs within Δt. Hence we infer that the impact starts with a sticking contact if

$$(\boldsymbol{v}_0 \cdot \hat{\boldsymbol{u}})^2 + (\boldsymbol{v}_0 \cdot \hat{\boldsymbol{w}})^2 < \mu^2 \eta_0^4 (\boldsymbol{v}_0 \cdot \hat{\boldsymbol{n}})^2, \tag{36}$$

or a sliding contact if

$$(\boldsymbol{v}_0 \cdot \hat{\boldsymbol{u}})^2 + (\boldsymbol{v}_0 \cdot \hat{\boldsymbol{w}})^2 \geq \mu^2 \eta_0^4 (\boldsymbol{v}_0 \cdot \hat{\boldsymbol{n}})^2. \tag{37}$$

Initial Stick

Using a similar approach, we can obtain the initial values of the derivatives of the tangential impulses when the contact sticks at the beginning:

$$I_u'(0) = \frac{1}{\eta_0^2} \cdot \frac{\boldsymbol{v}_0 \cdot \hat{\boldsymbol{u}}}{\boldsymbol{v}_0 \cdot \hat{\boldsymbol{n}}} \quad \text{and} \quad I_w'(0) = \frac{1}{\eta_0^2} \cdot \frac{\boldsymbol{v}_0 \cdot \hat{\boldsymbol{w}}}{\boldsymbol{v}_0 \cdot \hat{\boldsymbol{n}}}. \tag{38}$$

Next, we substitute $E_n'(I_n) \approx -v_{0n}$ into the integral (21) over $[0, h]$:

$$\begin{aligned} \boldsymbol{D}(h) &= \int_0^h \frac{1}{\sqrt{-v_{0n} I_n + O(I_n^2)}}\, dI_n \cdot \boldsymbol{v}_{0\perp} \approx -\frac{1}{v_{0n}} \cdot 2\sqrt{-v_{0n} I_n}\, \Big|_0^h \cdot \boldsymbol{v}_{0\perp} \\ &= -2\sqrt{-\frac{h}{v_{0n}}} \cdot \boldsymbol{v}_{0\perp}, \quad \text{since } v_{0n} < 0. \end{aligned}$$

The initial elastic energies $E_u(h)$ and $E_w(h)$ of the tangential springs are then evaluated according to (23).

Initial Slip

When the contact initially slips, the impulse derivatives follow Coulomb's law; namely,

$$I'_u(0) = -\mu \frac{\boldsymbol{v}_\perp \cdot \hat{\boldsymbol{u}}}{\|\boldsymbol{v}_\perp\|} = -\mu \hat{\boldsymbol{v}}_0 \cdot \hat{\boldsymbol{u}} \quad \text{and} \quad I'_w(0) = -\mu \hat{\boldsymbol{v}}_0 \cdot \hat{\boldsymbol{w}}. \tag{39}$$

The sliding velocity $\boldsymbol{v}_s$ must have the same direction as the direction $\hat{\boldsymbol{t}}$ of the relative tangential velocity $\boldsymbol{v}_{0\perp}$. Substituting (7) into (14) for F_n, we write down the changes of length of the u- and w-springs, and obtain E_u and E_w from (6), and G_u and G_w from (30).

3 Examples of Bouncing

This section demonstrates incorporation of tangential impulse into impact equations (1) using two examples. We look at bounces of a ball and a pencil, which result in planar and space impulse curves, respectively.

3.1 Ball

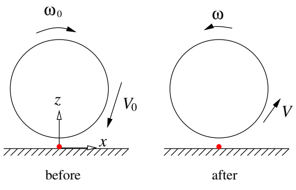


Fig. 3 Bouncing ball.

As shown in Fig. 3, a ball at initial velocity $\boldsymbol{V}_0$ and angular velocity $\boldsymbol{\omega}_0$ collides with a still table. Let r be the ball's radius and m its mass. Hence its angular inertia $\frac{2}{5}mr^2$. Denote by $\hat{\boldsymbol{z}}$ the upward contact normal, and $\boldsymbol{I}$ the impulse exerted by the table on the ball during the collision. The velocity equations (1) specialize to

$$\boldsymbol{V} = \boldsymbol{V}_0 + \frac{\boldsymbol{I}}{m} \quad \text{and} \quad \boldsymbol{\omega} = \boldsymbol{\omega}_0 - \frac{5}{2mr}\hat{\boldsymbol{z}} \times \boldsymbol{I}.$$

We obtain the contact velocity and its tangential component:

$$\begin{aligned} \boldsymbol{v} &= \boldsymbol{V} + \omega \times (-r\hat{\boldsymbol{z}}) = \boldsymbol{v}_0 + \frac{I_z}{m}\hat{\boldsymbol{z}} + \frac{7}{2m}\boldsymbol{I}_\perp, \\ \boldsymbol{v}_\perp &= \boldsymbol{v}_{0\perp} + \frac{7}{2m}\boldsymbol{I}_\perp, \end{aligned}$$

where $\boldsymbol{v}_0$ and $\boldsymbol{v}_{0\perp}$ are their initial values, and $\boldsymbol{I} = I_z\hat{\boldsymbol{z}} + \boldsymbol{I}_\perp$. We also obtain $E'_z = \frac{dE_z}{dI_z} = -(\hat{\boldsymbol{z}} \cdot \boldsymbol{v}_0)I_z - \frac{I_z^2}{2m}$.

Theorem 1. *During the collision of a ball with a still table, the tangential impulse $\boldsymbol{I}_\perp$ is collinear with the initial tangential contact velocity $\boldsymbol{v}_{0\perp}$.*

The proof is by induction over the number of contact mode transitions. Details are omitted due to lack of space.

Impulse Curve

Theorem 1 states that the impulse curve $\boldsymbol{I}$ lies in the vertical plane spanned by the z-axis and $\boldsymbol{v}_{0\perp}$.[3] So we can conveniently place the origin at the contact point, and the x-axis in the opposite direction of $\boldsymbol{v}_{0\perp}$. The x-y-z frame is identified with the n-u-w contact frame for tangential impulse in Section 2.

For simplicity, consider $m = 1$ and $r = 1$. Let the coefficient of friction be $\mu = 0.4$, the coefficient of restitution $e = 0.5$, and Poisson's ratio of the ball $\nu = 0.3$. Here, we use $\eta_0^2 = (2-\nu)/(2-2\nu)$ for a circular punch on a half space [8, pp. 361–366]. Consider $\boldsymbol{V}_0 = (-1, 0, -5)$ and $\boldsymbol{\omega}_0 = (0, 2, 0)$, which yields $\boldsymbol{V}_{0\perp} = (-3, 0, 0)$. After the collision, the ball will bounce backward with a reversal of its rotation: $\boldsymbol{V} = (V_x, 0, V_z) = (0.570982, 0, 2.5)$ and $\boldsymbol{\omega} = (0, \omega_y, 0) = (0, -1.92746, 0)$. Its total energy will decrease from 13.4 to 3.65997.

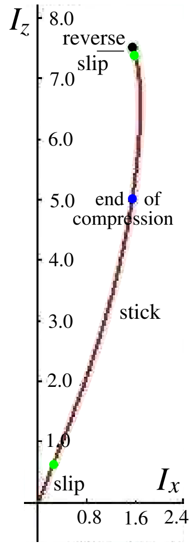


Fig. 4 Impulse curve.

Fig. 4 plots the impulse curve, on which the blue and black dots mark the ends of compression and restitution, respectively, and the two green dots mark the contact mode transitions. The impact starts with a slip, changes from slip to stick at $I_z = 0.62485$, ends compression at $I_z = -mv_{0z} = 5$, starts a reverse slip at $I_z = 7.36575$, and ends restitution at $I_z = -(1+e)\, mv_{0z} = 7.5$.

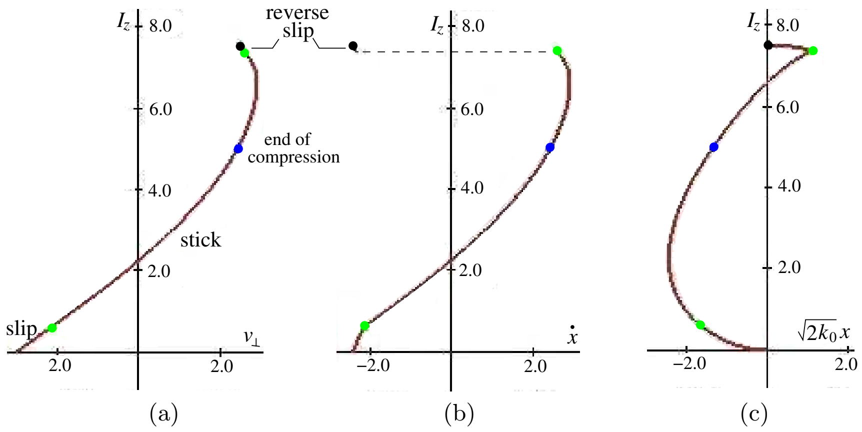


Fig. 5 (a) Tangential contact velocity; (b) x-spring velocity; and (c) varying spring length x scaled by $\sqrt{2k_0}$. The dashed line in (b) marks a discontinuity as reverse slip happens.

[3] It degenerates into a vertical line segment when $\boldsymbol{v}_{0\perp} = 0$.

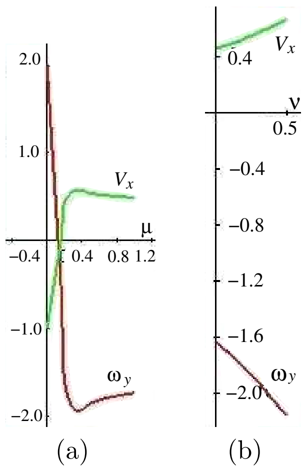


Fig. 6 Effects of (a) friction and (b) Poisson's ratio.

During the impact, the tangential contact velocity and spring velocity, aligned with the x-axis, are treated as scalars here and denoted $v_\perp$ and $\dot{x}$, respectively. As shown in Fig. 5(a), $v_\perp$ starts at -3 and ends at 2.49847. The x-spring velocity increases from -2.42852 with I_z until it equals $v_\perp$ at -2.12528, when the contact switches from slip to stick. Fig. 5(b) shows a sudden change of $\dot{x}$ from 2.59255 to -2.29806 when a slip reversal occurs at $I_z = 7.36575$. To see why, note that under slip $\dot{x}$ must satisfy (25), which becomes $x\dot{x} = (\mu^2 k/k_\perp^2)\dot{E}_z$. Because the transition happens during restitution, $\dot{E}_z < 0$, so x and $\dot{x}$ must have opposite signs at the moment. However, as shown in Fig. 5(b) and (c), both $\dot{x}$ and x were positive before the slip reversal. Hence the sudden change in $\dot{x}$. We can show that during stick the massless particle is in a simple harmonic motion.

Effect of Friction

Fig. 6(a) plots the post-impact velocities V_x and ω_y as the coefficient of friction μ varies from 0 to 1.0, where $\boldsymbol{V}_0 = (-1, 0, -5)$ and $\boldsymbol{\omega}_0 = (0, 2, 0)$. When friction is low ($\mu \leq 0.13$), the ball will bounce to the left but keep the original clockwise rotation about the y-axis. As μ increases from 0.13 but does not exceed 0.16, the ball will reverse its rotation but still bounce to the left. As friction becomes higher ($\mu > 0.16$), the ball will bounce backward with a rotation reversal. At $\mu = 0.36$, both v_x and ω_y reach their extrema 0.57591 and -1.93976, respectively.

Effect of Compliance

The dependence of V_x and ω_y on Poisson's ratio ν over its normal range $[0, 0.5]$ is shown in Fig. 6(b). Again, $\boldsymbol{V}_0 = (-1, 0, -5)$ and $\boldsymbol{\omega}_0 = (0, 2, 0)$. As ν increases from 0 to 0.5, V_x after the impact increases monotonically from 0.45729 to 0.66507, while ω_y decreases monotonically from -1.64322 to -2.16266. The more compliance, the less energy loss.

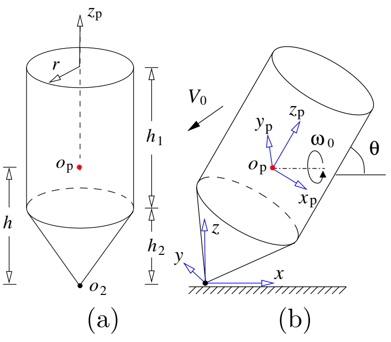


Fig. 7 Pencil with velocities $\boldsymbol{V}_0$ and $\boldsymbol{\omega}_0$ hitting a table: (a) dimensions and (b) frames.

3.2 Pencil

We move on to consider another task which many of us may have tried on a desk — throwing a pencil and watching how it bounces. Most of the time the pencil is thrown with its rubber eraser downward, but here let us consider a pencil strike with the pointed end contacting the desk.

As shown in Fig. 7(a), the pencil is modeled as a cylinder with mass m_1 and height h_1 on top of a cone with mass m_2 and height h_2, where both components have the same mass density. The cross sections of the cylinder and the top face of the cone have the same radius r. The pencil's center of mass o_{p} is located on its axis of symmetry at distance h above the cone's vertex o_2, where $h = (6h_1^2 + 12h_1h_2 + 3h_2^2)/(12h_1 + 4h_2)$. A body frame x_{p}-y_{p}-z_{p} is placed at o_{p} with the z_{p}-axis aligned with the pencil's axis of symmetry.

The moment of inertia Q of the pencil about its center of mass o_{p} is a diagonal matrix with first two principal moments:

$$Q_{11} = Q_{22} = \frac{m}{h_1 + h_2/3}\left(h_1\Big(\frac{3r^2 + h_1^2}{12} + l^2\Big) + \frac{h_2}{3}\left(\frac{3}{5}\Big(\frac{r^2}{4} + h_2^2\Big) + h^2\right)\right),$$

where $m = m_1 + m_2$ and $l = h_1/2 + h_2 - h$.

We treat the simple case where the pencil lies in a vertical plane at the moment of the strike. Let the plane be both the x-z plane of the desk frame at the contact point and the x_{p}-z_{p} plane of the pencil's body frame. See Fig. 7(b). The pencil, tilted at an angle θ just before the hit, has velocity $\boldsymbol{V}_0$ relative to the desk frame and angular velocity $\boldsymbol{\omega}_0 = (\omega_1, \omega_2, \omega_3)$ relative to a (fixed) frame instantaneously coinciding with the pencil frame. The orientation of the pencil frame in the desk frame is described by a rotation matrix R about the y-axis through θ. The velocities are determined from the impulse $\boldsymbol{I} = (I_x, I_y, I_z)$:

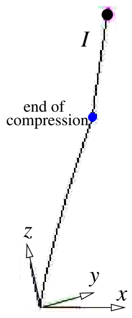


Fig. 8

$$\boldsymbol{V} = \boldsymbol{V}_0 + \frac{\boldsymbol{I}}{m} \quad \text{and} \quad -h\hat{\boldsymbol{z}}_{\mathrm{p}} \times (R^{-1}\boldsymbol{I}) = Q(\boldsymbol{\omega} - \boldsymbol{\omega}_0).$$

The velocity of the contact point during the strike is linear in $\boldsymbol{I}$:

$$\boldsymbol{v} = \boldsymbol{V}_0 + \frac{\boldsymbol{I}}{m} + h\begin{pmatrix} -\omega_2 \sin\theta \\ \omega_1 \\ \omega_2 \cos\theta \end{pmatrix} + \frac{h^2}{Q_{22}}\begin{pmatrix} I_x \sin^2\theta - I_z \sin\theta\cos\theta \\ I_y \\ -I_x \sin\theta\cos\theta + I_z\cos^2\theta \end{pmatrix}. \quad (40)$$

Specifically, we simulate a pencil with $m = 1$, $r = 1$, $h_1 = 3$, and $h_2 = 0.5$. Let $\mu = 0.8$ and $e = 0.5$. The pencil tilts at $\theta = \pi/3$, and strikes the desk with velocities $\boldsymbol{V}_0 = 5(-\cos\frac{\pi}{6}, 0, -\sin\frac{\pi}{6})$ and $\boldsymbol{\omega}_0 = (1, 0.5, 0.5)$. The post-impact velocities are $\boldsymbol{V} = (-1.80954, -0.546988, 1.2076)$ and $\boldsymbol{\omega}_0 = (0.09957, -0.04174, 0.5)$. The pencil bounces upward with reduced motion along the negative x-direction. It has gained a new motion along the negative y-axis. The angular velocity has changed along both x_p- and y_p-axes in the pencil's body frame. Its component along the z_p axis, i.e., the axis of symmetry, remains as 0.5, due to zero torque about the axis during impact.

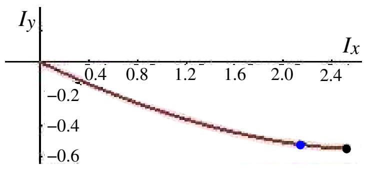


Fig. 9 Tangential impulse.

Fig. 8 plots the impulse curve, which grows from the origin to $(2.52059, -0.54699, 3.7076)$. The contact point slides during impact. The impulse projection onto I_x-I_y plane (see Fig. 9) is also a curve. This shows that the sliding direction was constantly changing in the impact duration.

4 Simultaneous Collisions with Compliance

In [7], we introduced a method to model simultaneous collisions of multiple objects based on transitions between states that characterize different combinations of the objects instantaneously in contact. Tangential impulses due to friction were then treated naively without considering compliance. As a result, the effect of a skillful shot like a massé one could not be modeled based on a measured input.

Tangential impulse due to compliance easily applies to simultaneous impacts. We here illustrate over a massé shot (see Fig. 10). The cue stick hits the ball at a point with outward normal $\hat{\boldsymbol{n}}$, and the ball in turn hits the table with upward normal $\hat{\boldsymbol{z}}$. We set up a local frame at the cue-ball contact with axes $\hat{\boldsymbol{n}}$, $\hat{\boldsymbol{u}}$, and $\hat{\boldsymbol{w}}$, and another frame at the ball-table contact with axes $\hat{\boldsymbol{z}}$, $\hat{\boldsymbol{x}}$, and $\hat{\boldsymbol{y}}$.

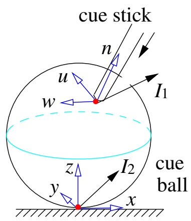


Fig. 10 Billiard shot.

The impulses exerted by the ball on the cue stick and by the table on the ball are respectively $\boldsymbol{I}_1 = I_u\hat{\boldsymbol{u}} + I_w\hat{\boldsymbol{w}} + I_n\hat{\boldsymbol{n}}$ and $\boldsymbol{I}_2 = I_x\hat{\boldsymbol{x}} + I_y\hat{\boldsymbol{y}} + I_z\hat{\boldsymbol{z}}$. According to [7], we can treat exactly one of I_u and I_z as the variable within a state, while the other as a dependent. Now, use the method in Section 2 to obtain I_u and I_w from I_n, and I_x and I_y from I_z. Next, use $\boldsymbol{I}_1$ and $\boldsymbol{I}_2$ to update all velocities, thus closing the loop.

We have designed a mechanical cue stick [7] which allows us to calculate the velocity of the cue tip before a shot. After the shot, the x- and y- components of the ball's velocity and angular velocity can be recovered from its trajectory via some involved steps.

Fig. 11(a) shows the trajectory fit over positions (red dots) sequenced from the video of a shot executed at a point near the top pole of the cue ball. The values of eleven relevant physical constants are omitted due to lack of space. The estimated velocity and angular velocity of the cue ball immediately after the shot are $(-1.65389, -0.36149, \_\_)$ and $(24.2768, 80.537, \_\_)$, respectively.

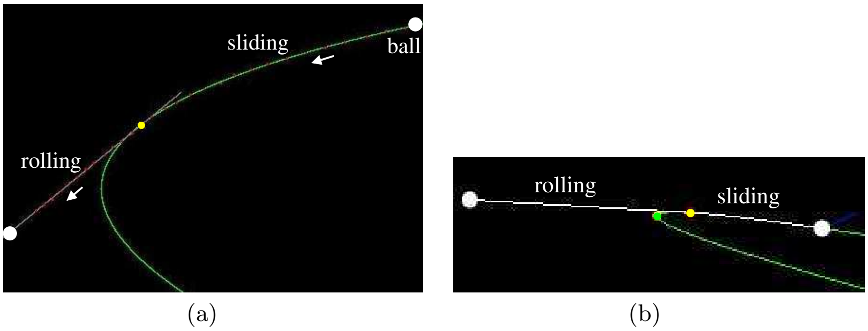


Fig. 11 Billiard trajectories (a) recovered from the video of a real shot and (b) predicted by the impact model with cue velocity estimated from the same video.

Fig. 11(b) shows the predicted ball trajectory by the impact model under the same shooting condition. The ball gets velocity $(-1.65774, 0.24380, 0.73265)$ and angular velocity $(15.9378, 52.9882, -3, 67556)$. Despite the small differences in the two pairs of velocities, the resulting trajectories differ widely. This is in part due to inaccurate measurements of related physical constants (including a guess over the relative stiffness of the cue-ball and ball-table contacts), the point-based impact model, and uncertainties of the shot.

5 Discussion

The key of the introduced compliance model for impact lies in that the elastic energies stored in the three orthogonal springs can be updated as functions of normal impulse without knowledge about their stiffnesses or length changes. The change rates of the spring lengths are nevertheless computable, so is the sliding velocity. Contact modes are decided from elastic energies rather than forces. All these make computation of tangential impulse possible, with normal impulse the sole variable of the impact problem.

In [13], Stronge claimed that the frictional energy loss depends on the sliding speed (i.e., the particle velocity $\boldsymbol{v}_s$), correcting his earlier statement that it depends on the tangential relative velocity $\boldsymbol{v}$. But it was not until the recent work by Hien [5] was the formulation of frictional dissipation completed. In our work, such dissipation is accounted for as the energies E_u and E_w are stored and released by the two tangential springs.

In their study [16] of a dimer bouncing on a vibrated plate, a similar differential impulse relationship is set up based on the ratio of the potential energies stored at the contact points. Coulomb's friction law is applied over the corresponding impulse increments. In our work, the friction law is applied in the forms (18) and (19) over the elastic energies stored at the contact.

The highly nonlinear nature of impulse accumulation (as shown in the pencil and the billiard examples) due to contact compliance would present an obstacle for formulation of a linear complementarity problem (LCP) as in [12] with a time-stepping solution. Our method is more accurate since it does not approximate the contact friction cone as a polyhedral cone, and also more efficient without having to repetitively solve linear systems.

The next step is to further integrate the compliance model with our previously developed model for simultaneous impacts [7]. Modeling of billiard shots provides a challenging test bed for meshing the two theories. A longer term objective is to apply the theory to impulsive robotic manipulation.

Acknowledgements. This work was sponsored in part by Iowa State University. Part of the work was performed in China. The author would like to thank Matt Mason for a pointer to Stronge's work on modeling compliance in impact mechanics, and Mike Erdmann for his encouragement. He is grateful to the anonymous reviewers for their valuable and constructive feedback.

References

1. Ahmed, S., Lankarani, H.M., Pereira, M.: Frictional impact analysis in open-loop multibody mechanical systems. J. Applied Mech. 121, 119–126 (1999)
2. Brach, R.M.: Tangential restitution in collisions. In: Schwer, L.E., et al. (eds.) Computational Techniques for Contact Impact, Penetration and Perforation of Solids, ASME AMD, vol. 103, pp. 1–7 (1989)
3. Darboux, G.: Etude géométrique sur les percussions et le choc des corps. Bulletin des Sciences Mathématiques et Astronomiques, deuxième série, tome 4, 126–160 (1880)
4. Han, I., Gilmore, B.J.: Impact analysis for multiple-body systems with friction and sliding contact. In: Sathyadev, D.P. (ed.) Flexible Assembly Systems, pp. 99–108. American Society Mech. Engineers Design Engr. Div. (1989)
5. Hien, T.: A correction on the calculation of frictional dissipation in planar impact of rough compliant bodies by W. J. Stronge. Int. J. Impact Engr. 37, 995–998 (2010)
6. Huang, W.H., Mason, M.T.: Mechanics, planning, and control for tapping. Int. J. Robotics 19(10), 883–894 (2000)
7. Jia, Y.-B., Mason, M., Erdmann, M.: A state transition diagram for simultaneous collisions with application in billiard shooting. In: Chirikjian, G., et al. (eds.) Algorithmic Foundations of Robotics VIII, pp. 135–150. Springer, Heidelberg (2010)
8. Johnson, K.L.: Contact Mechanics. Cambridge University Press, Cambridge (1985)
9. Keller, J.B.: Impact with friction. J. Applied Mech. 53(1), 1–4 (1986)
10. Routh, E.J.: Dynamics of a System of Rigid Bodies. MacMillan and Co., Basingstoke (1913)
11. Smith, C.E.: Predicting rebounds using rigid-body dynamics. ASME J. Appl. Mech. 58, 754–758 (1991)

12. Stewart, D.E., Trinkle, J.C.: An implicit time-stepping scheme for rigid body dynamics with inelastic collisions and Coulomb friction. Int. J. Numer. Methods Engr. 39, 2673–2691 (1996)
13. Stronge, W.J.: Impact Mechanics. Cambridge University Press, Cambridge (2000)
14. Tagawa, K., Hirota, K., Hirose, M.: Manipulation of dynamically deformable object using impulse-based approach. In: Zadeh, M.H. (ed.) Advances in Haptics, pp. 16–33 (2010)
15. Wang, Y., Mason, M.T.: Two-dimensional rigid-body collisions with friction. J. Applied Mech. 59, 635–642 (1991)
16. Zhao, Z., Liu, C., Brogliato, B.: Planar dynamics of a rigid body system with frictional impacts. Proc. Royal Soc. A: Math. Phys. Engr. Sci. 465, 2267–2292 (2009)

Sampling-Diagram Automata: A Tool for Analyzing Path Quality in Tree Planners

Oren Nechushtan[1,⋆], Barak Raveh[1,2,⋆], and Dan Halperin[1]

Abstract. Sampling-based motion planners are a central tool for solving motion-planning problems in a variety of domains, but the theoretical understanding of their behavior remains limited, in particular with respect to the quality of the paths they generate (in terms of path length, clearance, etc.). In this paper we prove, for a simple family of obstacle settings, that the popular dual-tree planner Bi-RRT may produce low-quality paths that are *arbitrarily worse than optimal* with modest but significant probability, and overlook higher-quality paths even when such paths are easy to produce. At the core of our analysis are probabilistic automata designed to reach an accepting state when a path of significantly low quality has been generated. Complementary experiments suggest that our theoretical bounds are conservative and could be further improved. To the best of our knowledge, this is the first work to study the attainability of high-quality paths that occupy a significant (non-negligible) portion of the space of all paths. The formalism presented in this work can be generalized to other algorithms and other motion-planning problems by defining appropriate predicates, and pave the way to deeper understanding of hallmark planning algorithms.

1 Introduction

The problem of finding collision-free paths for moving objects among obstacles, known as the Mover's problem, is P-Space hard when the number of degrees of freedom is considered a part of the input [24]. The problem of finding optimal paths with respect to some quality measure (such as path length, path clearance, energy,

Oren Nechushtan · Barak Raveh · Dan Halperin
School of Computer Science, Tel-Aviv University
e-mail: theoren,barak,danha@post.tau.ac.il

Barak Raveh
Dept. of Microbiology and Molecular Genetics, IMRIC, The Hebrew University

⋆ ON and BR contributed equally to this work.

etc.) is even harder, and was shown to be NP-hard even for simple special cases of the Mover's problem [1, 3, 18, 23]. Although these hardness results are discouraging in terms of worst-case analysis, in practice, motion planning is effectively used for a wide range of applications in diverse domains [4, 16]. In particular, the wide use of sampling-based algorithms, such as PRM [12], RRT [17], EST [9] and their many variants, has extended the applicability of motion planners beyond the restricted subset of problems that can be solved efficiently by exact algorithms.

Unfortunately, the theoretical understanding of sampling-based algorithms falls far behind the practical one, and in many senses, their proper usage may still be considered somewhat of an art. The most important results that exist to date shed light on the *asymptotic* performance of the PRM [11] and RRT [14] algorithms, which were both shown to be probabilistically complete [9, 11, 14]. However, the required running time for finding a valid solution, if one exists, cannot be computed for new queries at run-time in practice.

In many applications, we require not only a valid solution, but also a *high-quality* path. Several heuristics were devised for improving the *quality* of output paths generated by existing motion planners (e.g., [7, 8, 13, 20, 22]). While these approaches are theoretically motivated to some extent, the actual performance of sampling-based algorithm with regard to path quality is still poorly understood, even in very simple settings. Recently, Karaman and Frazolli[10] analyzed the convergence of sampling-based planners to optimal paths of zero-measure in the configuration, and have shown that at a reasonable cost of additional running time, modified variants of the RRT algorithm reach an optimal solution with probability one, as the number of samples goes to infinity. They also showed that the original RRT algorithm converges to sub-optimal solutions with probability one (under certain assumptions on the quality function used to measure path quality). However, this result does not indicate how bad the sub-optimal solution is – hypothetically, its quality may be $(1-\varepsilon)$ that of an optimal path, for some very small ε.

In this study, we rigorously analyze a simple family of toy-examples, which we call *The Promenade* problem (Figure 1). In this setting, the shortest possible path is highly accessible, without narrow paths leading towards it. We prove that the widely used dual-tree variant of the Rapidly-exploring Random Trees (RRT) algorithm, Bi-RRT [14], may take inferior paths that are *arbitrarily worse than the optimal path* and miss any higher-quality path with small but significant probability. Importantly, we provide a uniform probabilistic bound that holds for any instance of the Promenade problem (Figure 1). Our work, which is to the best of our knowledge the first of this kind, studies the attainability (or rather the non-attainability) of high-quality paths that occupy a non-negligible portion of the space of all paths, in contrast to the recent work by Karaman and Frazzoli [10].

As a more general contribution, we present a novel automaton-based approach for analyzing the performance of the Bi-RRT algorithm in a probabilistic manner, by assigning probability bounds to transitions between automaton states, which capture the progress of the algorithm. We believe that our new approach could greatly improve the currently limited theoretical understanding of sampling-based motion planners, by introducing a formalism for describing their behavior.

Outline: In Section 2 we briefly reiterate the Bi-RRT algorithm and define the Promenade family of obstacle settings. In Section 3 we introduce the Automaton of Sampling-Diagrams (ASD) formalism for describing the progress of the Bi-RRT algorithm, and in Section 4 we instantiate an appropriate ASD that proves a probabilistic lower-bound on the performance of Bi-RRT, stating and proving the main theorems of this manuscript. In Section 5 we empirically demonstrate that our bound holds for a standard implementation of Bi-RRT, and seems a fairly conservative one, and in Section 6 we discuss possible extensions of our method.

2 Problem Setting

2.1 The Bi-RRT Algorithm

Let Ω be a motion-planning problem (defined by a mobile object and a workspace cluttered with obstacles), and let (Ω,q_1,q_2) be the motion-planning query of moving the object between configurations q_1 and q_2 among the obstacles. In this work, we analyze the performance of the Bidirectional-RRT algorithm variant as described in the book by LaValle[16, p. 236], with slightly modified notation (Algorithm 1). The start and goal configurations, q_1 and q_2 respectively, serve as the roots of the trees T_1 and T_2. In each iteration i, the algorithm extends one of the trees T_{cur} within the obstacle-free portion of the configuration space, $\mathcal{C}_{\mathrm{free}}$, towards a random sample σ_i, drawn from the configuration space $\mathcal{C}$ (lines 3-7). The other tree T_{other} is then extended towards T_{cur} (lines 8-12). The algorithms breaks if the two trees were successfully connected and the query solved (line 13). At the end of each iteration, the tree to be extended may be swapped with T_{other} (line 14), most commonly in order to maintain a balance between the size of the two trees. However, the analysis we make here would hold for any possible tree-swapping strategy (see below). The pseudo-code for Bi-RRT is given in Algorithm 1. S_i denotes the swath of T_i (see below).

2.2 Subtle Implementation Issues of Bi-RRT

1. *Tree swaths:* The swath S_i of the tree T_i is an image of the vertices and edges of the tree in the configuration space [16, p. 236]. In a full setting, the swath is the union of line segments that correspond to T_i edges. In an approximate setting, it is reduced to a point subset (e.g., configurations of tree vertices).
2. *Distance metric*: The *nearest* function is defined as the nearest point according to some distance metric, with respect to the full swath, or to an appropriate point subset [16, pp. 230-234].
3. *Impact of tree-swapping strategy:* The choice of tree to be extended in each iteration (*swapping strategy*) may alter the algorithm output solution and complicate its analysis. The strategy of balancing the sizes of the two swath suggested in Algorithm 1 (line 14) is not always trivial to implement – in particular for the case of a full swath, adding even a single sample σ to a tree may considerably increase

Algorithm 1. Bi-RRT(Ω, q_1 , q_2, M)

```
1.  $T_1$.init($q_1$); $T_2$.init($q_2$); $T_{\text{cur}} = T_1$; $T_{\text{other}} = T_2$;
2.  for i = 1 to M do
3.      $q_n \leftarrow$ nearest($S_{\text{cur}}$, $\sigma_i$);
4.      $q_s \leftarrow$ stopping-configuration($q_n$,$\sigma_i$);
5.      if $q_s \neq q_n$ then
6.          $T_{\text{cur}}$.add_vertex($q_s$);
7.          $T_{\text{cur}}$.add_edge($q_n$, $q_s$);
8.          $q'_n \leftarrow$ nearest($S_{\text{other}}$, $q_s$);
9.          $q'_s \leftarrow$ stopping-configuration($q'_n$,$q_s$);
10.         if $q'_s \neq q'_n$ then
11.             $T_{\text{other}}$.add_vertex($q'_s$);
12.             $T_{\text{other}}$.add_edge($q'_n$ , $q'_s$ );
13.         if $q'_s = q_s$ then return SOLUTION;
14.     if $|T_{\text{other}}| < |T_{\text{cur}}|$ then swap($T_{\text{cur}}$,$T_{\text{other}}$);
15. return FAILURE;
```

its swath, by introducing a long line segment towards σ. The analysis we provide in this manuscript considers any, possibly adversarial worst-case scenario, of swapping strategies.

4. *Stopping-configurations:* Given a new sample σ, the tree T_{cur} is extended from $q_n \in \mathcal{C}$, the nearest neighbor of σ in the tree swath S_{cur}. However, the new sample may not be *visible* from q_n due to obstacles (formally, configurations are visible to one another iff the line segment connecting them is fully contained in $\mathcal{C}_{\text{free}}$). The *stopping configuration* function (line 4) is used to find a node $q_s \in \mathcal{C}_{\text{free}}$ that is visible from q_n. The Bi-RRT algorithm adds the stopping configuration, rather than the new sample, to the tree (lines 4-6 in Algorithm 1), but its exact definition is left to interpretation. For simplicity of our analysis, we make the plausible assumption that the stopping configuration is the farthest visible configuration from q_n along the line segment from q_n to σ.

Observation 1. *Given a new sample σ, the Bi-RRT algorithm adds σ to the tree T_{cur} iff σ is visible from its nearest neighbor in S_{cur}.*

2.3 The Promenade Motion-Planning Problem

For some $\alpha > 0$, let $\mathcal{P}_\alpha$ denote a motion-planning problem in $\mathbb{R}^2$ where we translate a point robot in the square configuration space $\mathcal{C} := [0, \alpha + 2]^2$ (for a point robot, the configuration space and the workspace are equivalent). The free configuration space is $\mathcal{C}_{\text{free}} := \mathcal{C} \setminus [1, \alpha + 1]^2$, that is, there is a single $\alpha \times a$ square obstacle in the middle of $\mathcal{C}$, leaving a rectangular ring "promenade" for the robot (see Figure 1). We are interested in bounding the probability that running the Bi-RRT algorithm for solving the Promenade motion-planning problem will result in a low-quality solution (type-B solution in Figure 1).

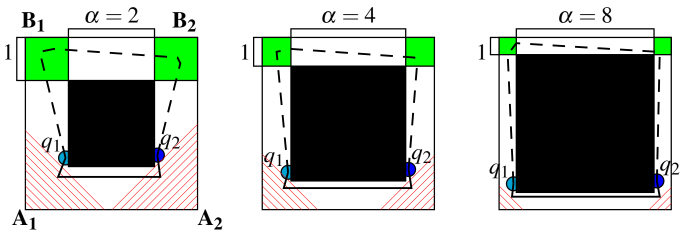


Fig. 1 The Promenade family of motion planning problems, illustrated for inner-square obstacles (black) with three different edge-size values (scenes rescaled for illustration purposes). In these examples, q_1 and q_2 lie on opposite sides of the promenade. The ratio μ between the path length of type-A (solid line, crossing the regions marked A_1 and A_2) and type-B (dashed line, crossing B_1 and B_2) solutions, is approximately $\frac{1}{3}$ in this case. Note that the probabilistic bound obtained by the Automaton of Sampling-Diagrams method for these scenes is not sensitive to the size of the inner-square obstacle relative to the bounding box (for all $\alpha \geq 2$), or to the precise location of q_1 and q_2. In addition, the same bound holds for any, possibly adversarial worst-case scenario of tree-swapping strategy (line 14 in Algorithm 1).

3 Introducing the Automaton of Sampling-Diagrams

We first introduce the formal concepts underlying the construction of an *Automaton of Sampling-Diagrams* (ASD), a *Finite-State Machine* that reads a sequence of *samples* in $\mathcal{C}$, combined with a sequence of *tree-swap decisions*. The rationale for this choice of input is that together, the sequence of tree-swap decisions and samples determines the output path returned by Bi-RRT uniquely. This also makes our analysis robust to any (possibly adversarial worst-case) tree-swapping strategy. After laying out the necessary formalism in this section, we will construct an instance of this type of automaton in Section 4, in which an accepting state is reached only if the Bi-RRT algorithm returns a solution of particularly low quality.

Definition 1. [A Sampling Diagram] *A sampling-diagram D_σ over the configuration space $\mathcal{C}$ is a partial, possibly overlapping subdivision of $\mathcal{C}$ into three subsets, $F_1(D_\sigma)$, $F_2(D_\sigma)$, $R(D_\sigma) \subseteq \mathcal{C}$. We require $R(D_\sigma)$ to be disjoint from $F_1(D_\sigma)$ and $F_2(D_\sigma)$.*

Definition 2. [Automaton of Sampling-Diagrams (ASD)] *An ASD $\mathbb{A}$ is a Finite-State Machine that reads the composite infinite alphabet ($\theta \times \sigma$) for $\theta \in \{\hat{T}_1, \hat{T}_2\}$ and $\sigma \in \mathcal{C}$. Each state $s \in$ States($\mathbb{A}$), corresponds to a sampling-diagram $D_\sigma[s]$, which determines the transition function of $\mathbb{A}$ as follows:*

- *If $\mathbb{A}$ moves to a rejecting or an accepting state, it is trapped in it.*
- *For any other state s and input character (θ, σ):*

  1. *If $\sigma \in R(D_\sigma[s])$, $\mathbb{A}$ moves to a rejecting state.*
  2. *If $\theta = \hat{T}_i$ and $\sigma \in F_i(D_\sigma[s])$, $\mathbb{A}$ moves "forward" to a non-rejecting neighboring state, denoted by $N_i(s)$. $N_1(s)$ and $N_2(s)$ can be the same state.*
  3. *Otherwise, $\mathbb{A}$ remains in s.*

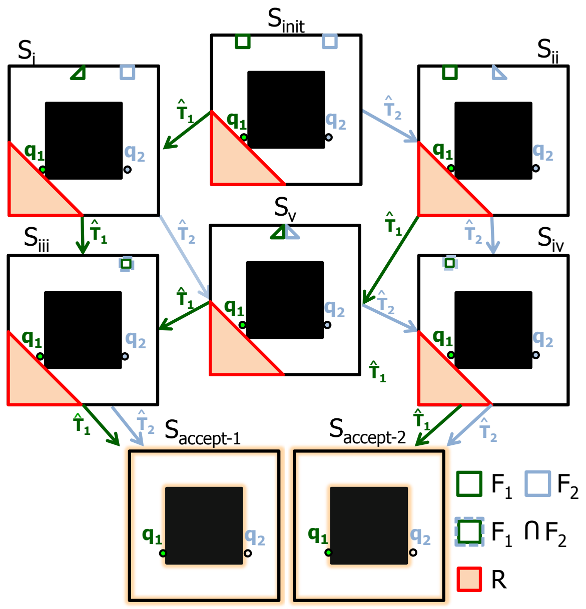


Fig. 2 An illustration of $\mathbb{A}_\alpha$, an Automaton of Sampling-Diagrams (ASD) for the $\mathscr{P}_\alpha$ Promenade problem with $\alpha = 2$, using the ℓ_1 norm. The transition function is set by samples hitting the sampling-diagrams of each state. When a sample hits the F_i region while the tree T_i is being extended, $\mathbb{A}_\alpha$ moves forward along the transition edge labeled $\hat{T}_i$, whereas hitting R moves $\mathbb{A}_\alpha$ to rejecting state (omitted for clarity). The figure is best viewed in color.

Given a sequence of tree-swap decisions $\Theta_m = (\theta_1, \theta_2, \ldots, \theta_m) \in \{\hat{T}_1, \hat{T}_2\}^m$ together with a sequence of samples $\Sigma_m = (\sigma_1, \sigma_2, \ldots, \sigma_m)$, we say that $\mathbb{A}$ reads Θ_m and Σ_m. The sampling-diagrams encode the relation between the tree-swap decisions Θ_m, the samples Σ_m and the automaton transition function, where F and R stand for moving "forward" to a non-rejecting state, or moving to a rejecting state, respectively. This formalism can be generalized (see Section 6).

3.1 Relating ASDs to Bi-RRT – Swath-Realizing ASDs

The following definitions relate the Automaton of Sampling-Diagrams, the tree-swaths created by Bi-RRT, and the combination of amples and tree-swapping decisions that make up the input to both[1].

Definition 3. [Probability Space] *Let $\Sigma_m := (\sigma_1, \sigma_2, \ldots, \sigma_m)$ be a sequence of m samples chosen independently at random from Γ, a probability space over the configuration space $\mathbb{C}$. The probability space of all such sequences Σ_m is Γ^m.*

Definition 4. [Induced Swaths] *Let S_1 and S_2 be the swaths generated by running Bi-RRT, with the tree-swapping decisions $\Theta_m \in \{\hat{T}_1, \hat{T}_2\}^m$, used to decide which tree to extend in each iteration, and the samples $\Sigma_m \in \Gamma^m$. S_1 and S_2 are the Induced Swaths of Θ_m and Σ_M, denoted $S_1[\Theta_m, \Sigma_m]$ and $S_2[\Theta_m, \Sigma_m]$.*

[1] In order to technically simplify the definitions, we let Bi-RRT run after returning the output solutions, but without ever changing the tree-swaths, thereby not affecting the analysis.

Definition 5. [Swath Diagram] *A Swath-Diagram Δ over the configuration-space $\mathcal{C}$ is a partial subdivision of $\mathcal{C}$ into two disjoint subsets, $\Delta^+, \Delta^- \subseteq \mathcal{C}$,*

Definition 6. [Realizing Swath-Diagrams] *The pair of swaths (S_1, S_2) realizes the pair of Swath-Diagrams (Δ_1, Δ_2) if:*

1. $\Delta_i^+ \neq \phi \Rightarrow S_i \cap \Delta_i^+ \neq \phi$ (intersect positive regions),
2. $S_i \cap \Delta_i^- = \phi$ (avoid negative regions),
3. $\Delta_1^+ \cap \Delta_2^+ \neq \phi \Rightarrow S_1 \cap S_2 \cap \Delta_1^+ \cap \Delta_2^+ \neq \phi$ (connect trees).

Swath-Diagrams encode critical configuration-space regions that the tree swath must intersect (Δ^+) or avoid (Δ^-), and regions where the two swaths connect ($\Delta_1^+ \cap \Delta_2^+$). The latter implies that the motion-planning problem is solved.

Definition 7. [Swath-Realizing Automaton of Sampling-Diagrams] *We associate each state s of an ASD $\mathbb{A}$ with a pair of swath-diagrams $\Delta_1(s)$ and $\Delta_2(s)$. $\mathbb{A}$ is a Swath-Realizing ASD if:*

- *The pair of induced-swaths $(S_1[\Theta_m, \Sigma_m], S_2[\Theta_m, \Sigma_m])$ realizes the pair of diagrams $(\Delta_1(s), \Delta_2(s))$, for any input $\Theta_m \in \{\hat{T}_1, \hat{T}_2\}^m$ and $\Sigma_m \in \Gamma^m$ that moves $\mathbb{A}$ to a regular (non-rejecting, non-accepting) state s, for any m.*
- *The Swath-Diagrams of accepting states must be realized when first visited by $\mathbb{A}$.*

The definition of a Swath-Realizing ASD requires that if we run Bi-RRT and advance the ASD in parallel, using the exact same input, then the swaths created by Bi-RRT will invariantly respect the rules associated with the current automaton state. This reduces the problem of analyzing the progress of the Bi-RRT algorithm on a given input to the inspection of the swath-diagrams associated with the current automaton state, which encode the desired expansion of the tree swaths.

4 Proving a Probabilistic Bound for the Promenade Problem

In this section we prove the existence of a Swath-Realizing ASD $\mathbb{A}_\alpha$ for the Promenade family of problems $\mathscr{P}_\alpha$, in which the swath-diagrams of accepting states encode low-quality solutions and in which only the swath-diagram of the rejecting state allows high-quality solutions. An instance of $\mathbb{A}_\alpha$ for $\alpha = 2$ is illustrated in Figures 2 and 3. By carefully analyzing the transition probability between states in the automaton, we shall lower-bound the probability to return a low-quality solution.

Definition 8. [$\leq \mu$-Quality Solution] *For $0 < \mu \leq 1$, a solution path ω to a motion-planning query (Ω,q_1,q_2) is said to be $\leq \mu$-*Quality *if $Q(\omega) \leq \mu Q_{opt}$, where Q_{opt} is the optimal quality for the query.*

Consider the quality function Q that is inversely correlated with the length of a solution path ω under the ℓ_p norm. Formally, $Q(\omega) = \frac{1}{\text{length}_{\ell_p}(\omega)}$. We define the following key configuration-space regions (see Figure 1). A_1 and A_2 are the isosceles right triangles at the bottom-left and bottom-right corners of $\mathcal{C}_{\text{free}}$, respectively. B_1 and B_2 are 1×1 squares adjacent to the top-left and top-right corners, respectively.

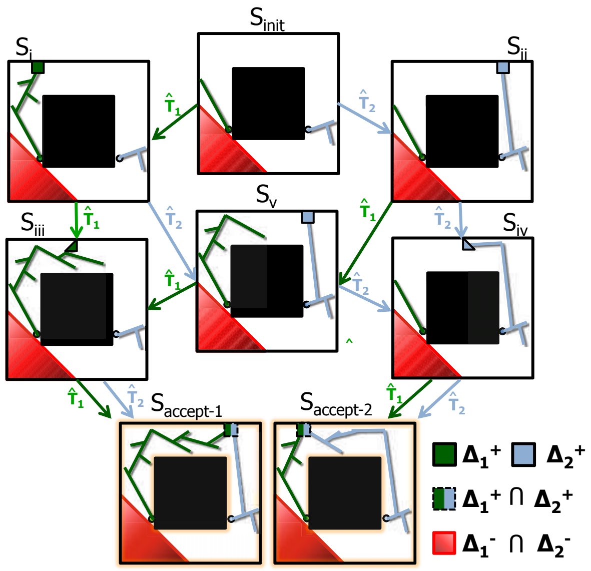


Fig. 3 The swath-diagrams associated with $\mathbb{A}_\alpha$, the automaton shown in Figure 2. The growth of the tree swaths S_1 (dark-green line) and S_2 (light-blue line), induced by running Bi-RRT on the same input given to $\mathbb{A}_\alpha$, is illustrated. $\mathbb{A}_\alpha$ is Swath-Realizing, as it maintains the invariant that each S_i intersects Δ_i^+, avoids Δ_i^- and connects to the other tree whenever $\Delta_1^+ \cap \Delta_2^- \neq \phi$. The figure is best viewed in color.

Definition 9. [Type-A and Type-B paths] *Type-A solution paths to $\mathscr{P}_\alpha$ intersect A_1 but not B_1. Type-B solution paths intersect both B_1 and B_2. Q_A^* and Q_B^* are the optimal qualities for any Type-A or Type-B paths, respectively.*

Type-B solutions topologically correspond to paths going above the obstacle (see Figure 1). For given configuration q_1 and q_2, let μ be defined as: $\mu = \frac{Q_B^*}{Q_A^*}$. It is easy to position q_1 and q_2 such that Type-B solutions are $\leq \mu$-Quality for a small μ.

Theorem 1. [Path Quality in the Promenade Problem] *There exists a constant $c_0 > 0$, such that for any $0 < \mu \leq 1$, any $\alpha \geq 2$, any tree-swapping strategy and using samples drawn from Γ^m. there exist initial and goal configurations q_1 and q_2 for which the path ω returned by Bi-RRTsatisfies:*

$$Pr[\omega \textit{ is} \leq \mu\text{-Quality}] \geq c_0 \ .$$

The proof of Theorem 1 is involved and proceeds in several stages. We first show in Theorem 2 (Section 4.1) that accepting states of the automaton $\mathbb{A}_\alpha$ correspond to type-B solutions and that any type-A solution moves $\mathbb{A}_\alpha$ to a rejecting state. The heart of this proof is in Proposition 1, where we show that $\mathbb{A}_\alpha$ is swath-realizing. The probability to reach an accepting state (low-quality solution) is analyzed in Theorem 3 (Section 4.2). All this applies to $\frac{1}{3} < \mu < 1$ and to the ℓ_1 norm (as it is easiest to illustrate). The extension to any $\mu > 0$ (paths of arbitrarily low quality) and to other ℓ_p is discussed in Section 4.3.

4.1 $\mathbb{A}_\alpha$ *Accepts Inputs That Induce* $\leq \mu$-**Quality** *Solutions*

Theorem 2. *For all $\alpha \geq 2$, for all $q_1 \in \mathcal{C}_{\text{free}}$ enclosed between A_1 and B_1 (on the left-hand side of $\mathcal{C}$), and for all $q_2 \in \mathcal{C}_{\text{free}}$ enclosed between A_2 and B_2 (on the right-hand side of $\mathcal{C}$), the automaton $\mathbb{A}_\alpha$ (as in Figures 2 and 3) moves to:*

1. *A rejecting state on all inputs for which Bi-RRT outputs a type-A solution.*
2. *An accepting state only on inputs for which Bi-RRT outputs a type-B solution.*

Proof. First, it is easy to verify that $\mathbb{A}_\alpha$ is a proper ASD by Definition 2. In Proposition 1 below we show that $\mathbb{A}_\alpha$ is Swath-Realizing. The proof of the Theorem follows immediately:

Part (1): A_1, the isosceles right triangle at the bottom-left corner of the promenade (see Figure 1), is defined as the $\Delta_1^-(s)$ and $\Delta_2^-(s)$ regions for any non-rejecting state s in $\mathbb{A}_\alpha$. Observe that high-quality (type-A) solution must cross A_1. Assuming $\mathbb{A}_\alpha$ is a Swath-Realizing ASD, it must move to a rejecting state on all inputs for which the induced swath intersects A_1.

Part (2): By construction, the diagram regions Δ_1^+ and Δ_2^+ intersect each other in any accepting state of $\mathbb{A}_\alpha$. Assuming $\mathbb{A}_\alpha$ is a Swath-Realizing ASD, the induced swaths S_1 and S_2 necessarily cross each other upon moving to an accepting state (see Definition 6), and contain a solution to the motion-planning problem. From Part (1), this can only be a type-B solution. □

Proposition 1. $\mathbb{A}_\alpha$ *is Swath-Realizing.*

Due to space considerations, we include part of the proof details as supplementary on-line material [19], and only give a sketch of the proof here.

Sketch of Proof. Consider the swaths S_1 and S_2 induced by input to $\mathbb{A}_\alpha$. First, the swath-diagram regions $\Delta_1^-(s)$ and $\Delta_2^-(s)$ are defined as the A_1 region, the isosceles right triangle at the bottom-left corner of the promenade (see Figure 1). Since A_1 is clearly the intersection of a half-plane with $\mathcal{C}$, A_1 forms a "visibility block" in the configuration space. Formally, any $s,t \in \mathcal{C}$, $s,t \notin A_1$ lie on the other half-plane, and therefore the segment between s and t does not intersect A_1. By construction of $\mathbb{A}_\alpha$ (as in Figure 2, no sample hits A_1 until $\mathbb{A}_\alpha$ reaches a rejecting state, because the sampling-diagram rejecting region $R(D_\sigma[s])$ is also defined as A_1 for all regular (non-rejecting, non-accepting) states. Since also $q_1, q_2 \notin A_1$, we conclude that the

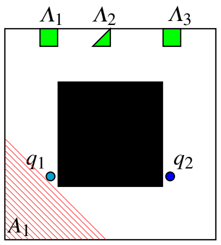


Fig. 4 Key regions in the Promenade configuration space. The directed left-path in the state graph of the automaton $\mathbb{A}_\alpha$ (see Figures 2 and 3) forces the tree rooted at q_1 to intersect Λ_1, then Λ_2, then Λ_3, reaching q_2 through a type-B solution (above the obstacle). These key regions are geometrically designed to be sufficiently close to each other, such that Bi-RRT will connect them in this order.

swaths S_1 and S_2 produced by Bi-RRT may not intersect A_1, which is equivalent to $\Delta_1^-(s)$ and $\Delta_2^-(s)$, for any non-rejecting state s of $\mathbb{A}_\alpha$, as required by Definition 6.

Second, we need to show that whenever we move to a state s, the swaths S_1 and S_2 induced by running Bi-RRT intersect the Δ_1^+ and Δ_2^+ regions, respectively, if those are not empty sets (Definition 6). We proceed by induction on the length of the input. At the initial state, $\Delta_1^+ = \phi$ and $\Delta_2^+ = \phi$, so the induction hypothesis holds trivially. If we move from any state s to itself, the induction hypothesis holds since S_i can only expand further, and by the induction hypothesis, S_i already intersects $\Delta_i^+[s]$ before reading the next input. We now prove the induction for each transition in the left-most directed path in the state graph of $\mathbb{A}_\alpha$, as illustrated in Figures 2 and 3.

$$s_{init} \longrightarrow s_i \longrightarrow s_{iii} \longrightarrow s_{accept1} \ .$$

In each of these transitions, we extend the tree T_1 and its swath S_1. Other directed paths, which correspond to alternative tree-swapping decisions, are analyzed by similar argumentation, and are omitted due to space considerations. The first transition $s_{init} \longrightarrow s_i$ occurs when we extend S_1 and a new sample σ hits Λ_1, a small $\frac{1}{3} \times \frac{1}{3}$ square in the top-left corner of $\mathcal{C}_{\text{free}}$ (see Figure 4). This transition is encoded in the fact that Λ_1 is the "forward" region F_1 of the initial state *sampling-diagram* $D_\sigma[s_{init}]$ (Figure 2). Λ_1 is also the positive *swath-diagram* region of the next state, $\Delta_1^+[s_i]$ (Figure 3), so for the induction hypothesis to hold, S_1 should intersect Λ_1 after Bi-RRT runs on the sample $\sigma \in \Lambda_1$. Indeed, the position of the region Λ_1 is designed to guarantee that any $\sigma \in \Lambda_1$ is visible from its nearest-neighbor in S_1, and therefore will be added to S_1 by the Bi-RRT algorithm by Observation 1 in Section 2). A detailed proof of the last claim is given as on-line material [19, Lemma 7(i)].

The second transition $s_i \longrightarrow s_{iii}$ is designed to bring the swath S_1 closer to the top-right corner region of the promenade ($B2$). The transition occurs when a new sample σ hits the "forward" region $F_1(D_\sigma[s_i])$, a small triangle Λ_2 located at the top edge of the promenade (Figure 4). Again, the triangle Λ_2 is designed such that $\sigma \in \Lambda_2$ will be visible from its nearest neighbor in the swath, and therefore would be added to the swath S_1, making it intersect the positive swath-diagram region $\Delta_1^+[s_{iii}]$. A complete proof is given in the on-line material [19, Lemma 7(ii)], based on the relative position of the triangle Λ_2 to the square Λ_1, which S_1 intersected already at the previous state. Indeed, the rationale behind the position of Λ_1 is also to provide visibility for S_1 towards Λ_2.

The last transition $s_{iii} \longrightarrow s_{accept1}$, brings S_1 to a $\frac{1}{3} \times \frac{1}{3}$ square Λ_3 at the top-right corner of the promenade ($B2$). Note that any point in Λ_3 is visible from q_2 (Figure 4), and T_2 and T_1 will now be connected, completing a type-B solution path to the Promenade problem (see lines 8-12 in Algorithm 1). The detailed proof for this transition is found in [19, Lemma 7(iii)]. In a similar way, each of the remaining transitions in $\mathbb{A}_\alpha$ can be shown to preserve the invariant that the swath intersects the favorable (Δ_i^+) regions of the swath-diagrams, and that T_1 and T_2 are connected whenever $\Delta_1^+ \cap \Delta_2^+ \neq \phi$, as required by Definition 6, hence $\mathbb{A}_\alpha$ is swath-realizing. □

4.2 Probabilistic Analysis of $\mathbb{A}_\alpha$

Theorem 3. *If $\sigma \in \Gamma$ has positive probability to hit the F_1 or F_2 regions of the sampling-diagram $D_\sigma[s]$ for any regular (non-rejecting, non-accepting) state $s \in \mathbb{A}_\alpha$, then there exists $c_0 > 0$ such that for any $m \geq M_0$, for any tree-swapping strategy:*

$$Pr[\text{Accept}] \geq c_0 \, .$$

Where $Pr[\text{Accept}]$ is the probability of $\mathbb{A}_\alpha$ to reach an accepting state after reading the tree-swapping decisions $\Theta_m \in \{\hat{T}_1, \hat{T}_2\}^m$ and samples $\Sigma_m \in \Gamma^m$.

Proof. **[Theorem 3]** Assume that the current state of $\mathbb{A}_\alpha$ is a regular state s, and that the next tree-swapping decision the automaton reads is $\hat{T}_i$. After reading the next sample, the automaton $\mathbb{A}_\alpha$ either: (i) Hits the F_i region of $D_\sigma[s]$ and moves forward to another non-rejecting neighboring state $N_i(s)$ (event $\mathfrak{F}_i(s)$), (ii) hits the R region and moves to a rejecting state (event $\mathfrak{R}(s)$) or, (iii) remains in s. Define (i) and (ii) as critical events.

Denote by $\text{vol}_\Gamma(F_i)$ and $\text{vol}_\Gamma(R)$ the volume of the F_i and R regions in $D_\sigma[s]$, weighted by the probability distribution of Γ, The conditional probability $\pi_i(s)$ to move forward from s to $N_i(s)$, given that a critical event happened, is:

$$\pi_i(s) = Pr[\mathfrak{F}_i(s) | \mathfrak{F}_i(s) \vee \mathfrak{R}(s)] = \frac{\text{vol}_\Gamma(F_i)}{\text{vol}_\Gamma(F_i) + \text{vol}_\Gamma(R)} > 0 \, . \tag{1}$$

The probability is positive since we assumed that Γ assigns positive probabilities to hit $F_i(D_\sigma[s])$ for all non-rejecting states. $\mathbb{A}_\alpha$ is finite and has a directed-acyclic state graph (if we do not consider self-loops) in which the end nodes are the accepting and rejecting states, and serve as traps. Therefore, given sufficient input size, $\mathbb{A}_\alpha$ would eventually either: (1) End up in an accepting state by moving at most k steps, k being the length of the longest directed path in the state graph, or (2) end up in a rejecting state.

Let $\{\psi_i\}$ be the finite set of all directed paths without loops from s_{init} to any accepting state. If $k = 1$, then the conditional probability to move forward from the initial state s_{init} to an accepting state, regardless of the tree-swapping strategy taken along the way (whether $i = 1$ or $i = 2$ in each iteration), is bounded from below by $min[\pi_1(s_{\text{init}}), \pi_2(s_{\text{init}})] > 0$. Let $\pi(s|\psi_i)$ denote the conditional probability to move forward from any state s to the next state in ψ_i. (as in Equation 1). By induction on the length k of the longest directed path, and since the events are independent, the probability of $\mathbb{A}_\alpha$ to end up in an accepting (rather than a rejecting) state is bounded from below by c_1, the minimal product of conditional probabilities to move forward over the states of ψ_i. Formally:

$$c_1 = \min_{\psi_i \in P}[\prod_{s \in \psi_i} \pi(s|\psi_i)] > 0 \, . \tag{2}$$

Let M_0 be the minimal input length that guarantees $Pr[\text{Accept} \vee \text{Reject}] \geq 1 - \varepsilon$ for some small $\varepsilon > 0$. Then for $c_0 = (1 - \varepsilon) \cdot c_1$, for any $m \geq M_0$:

$$Pr[\text{Accept after } m \text{ iterations}] \geq c_0 > 0\ . \tag{3}$$

□

Explicit calculation of c_0, the probabilistic bound to get a $\leq \mu$-Quality solution, leads to the extremely small number of approximately 4×10^{-6} for the uniform distribution, which is the product of transition probabilities ($\frac{1}{19 \cdot 19 \cdot 19 \cdot 37}$), based on the ratio between the area of F_i and $F_i \cup R$ in each transition. Importantly, this bound is true for any $\alpha \geq 2$, since this ratio remains fixed, and does not depend on the tree-swapping strategy. In the next section we show that our bound is conservative.

As a side-benefit of our analysis, we can bound M_0, the minimal number of iterations it takes $\mathbb{A}_\alpha$ to end up in an accepting or rejecting state, by considering also the probability to remain in the same state. This probability is calculated from the probability-weighted volumes of the F_1, F_2 and R regions, which depend also on the edge-size α of the inner-square and on the probability distribution Γ. The analysis can be further extended to bound the time it takes Bi-RRT to return a valid solution. We leave the explicit calculation outside the scope of the current manuscript.

4.3 Extensions to General ℓ_p and Arbitrarily Small μ

If configurations q_1 and q_2 are proximal to the bottom-left and bottom-right corners of the inner square, respectively, then μ reaches $\frac{1}{3}$. In order to generalize the proof to arbitrarily small μ, we need to position q_2 at the bottom side of the promenade, next to A_1 in the bottom-left corner (as in Figure 5), turning Type-B solutions arbitrarily longer than the optimal path through A_1. The appropriate automaton requires the addition of a few more states for crossing the bottom-right corner of the promenade (A_2 region in Figure 1, which now becomes a type-B region). This can be done in a straightforward manner analogously to the crossing of the top-right corner in $\mathbb{A}_\alpha$; the details are omitted due to space limitations. In addition, we showed Theorem 3 only for the ℓ_1 norm, as it was easiest to draw the diagrams for its automaton. However, the proof does not rely on specific properties of ℓ_1 and can be generalized to any ℓ_p for $1 \leq p \leq \infty$, and to the Euclidean norm ℓ_2 in particular.

5 Experiments on the Promenade Problem

We compared our results to a widely used implementation of the Bi-RRT algorithm, using the OOPSMP software for motion-planning [21] (version 1.2β). We empirically tested the performance of the algorithm on an instance of the Promenade problem with $\alpha = 4$. In each experiment, we conducted 5000 independent runs of Bi-RRT followed by standard path smoothing.

In the first experiment, the initial and goal configurations were set at opposite sides of the promenade, next to the bottom-left and bottom-right corners of the inner square (as in Figure 1). In this case, in as much as *49.4% of cases* ($\pm 1.1\%$ std-err) the output path generated was at least three times worse than optimal. This striking result means that our theoretical bound is fairly conservative, and the actual probability of Bi-RRT to produce low-quality paths is higher. This may indicate that when the initial and goal configurations are positioned this way, the actual bound for finding a low-quality paths is $0.5 \pm \delta$ for some small $0 \leq \delta << 1$, where δ goes to zero as the initial and goal configurations reach the corners.

In a second experiment, where the initial and goal configurations were set right next to each other, but such that the bottom-left corner occludes their mutual visibility (Figure 5), a near-optimal path was generated in 94.1% (± 0.3%) of cases. However, in the remaining cases a path that is *over 140 times worse than optimal after smoothing* was selected (Figure 5**B**), even though the start and goal configuration were in absolute proximity to each other and were not separated by a narrow passage. See Section 6 below for experiments in a 3D scene.

6 More Complex Settings

To give a completely worked out and proved example, we used a very simple family of toy scenarios. In this section we point out where we believe the method can be extended to more complex settings vs. where difficulties lie in such extensions.

Several disconnected blocking zones. In the promenade $\mathscr{P}_\alpha$, we can block high-quality solution paths with a *single* triangle ($A1$ in Figure 1). This simplifies the calculus a lot but does not apply in more complicated examples. We can then think of this setting as a sub-problem within a bigger problem. For instance, the analysis follows through also when we replace the single blocking-zone predicate with more complicated predicates describing several (disconnected) blocking zones (Figure 6**A**). In this case, the automaton reaches an accepting state after we

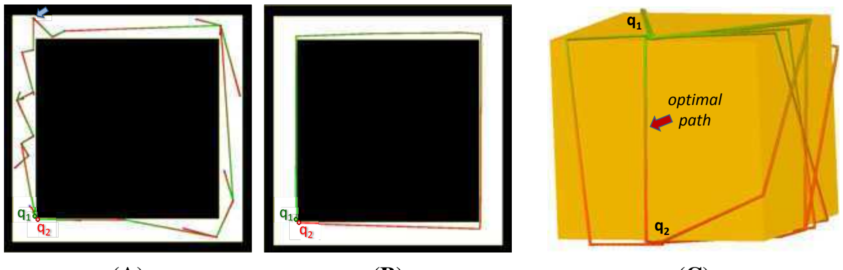
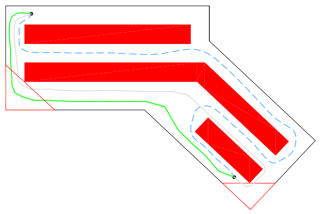
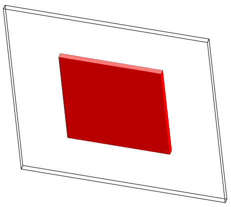


Fig. 5 Experimental results. (**A**) Instance of the Promenade problem for $\alpha = 4$. The initial and goal configuration q_1 and q_2 are proximal, such that their mutual visibility is blocked by the left-bottom corner. The merged trees (merge point marked by arrow) are shown. (**B**) The solution path extracted from the trees in (**A**) is over 140 times worse than optimal, even after standard path smoothing, as typical in 5.9% of independent runs. (**C**) Overlay of representative output paths for the 3D cube-within-a-cube problem. The edge-size of the inner-cube (orange) is 90% that of the bounding-cube (not shown). A low-quality path was returned in 97.3% of cases, even after path smoothing (see Section 6).





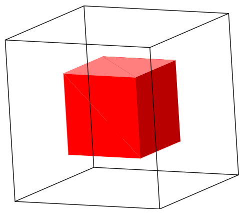


Fig. 6 More complex settings: (**A**) Maze with two disconnected blocking-zones (red triangles), which prevent good quality solutions (green and gray lines), forcing a path with much lower quality (dashed blue line). (**B**) The Promenade sweeped by ε_3 in the z-direction. (**C**) cube-within-a-cube.

sample within some (low-quality-path) zones in the appropriate order (F regions of a sampling-diagram), while avoiding the blocking zones, which move the automaton to the rejecting state (R regions). This type of problems illustrates the sensitivity of tree-based planners to a sequence of critical events, and may justify a heuristic approach we presented recently for improving path quality, by hybridizing sub-solutions from multiple independent runs [5, 22].

Higher dimensions. We consider two extensions of the Promenade problem to higher dimensions. The first extension is by a cross-product with a hyper-box $[0,\varepsilon_3]\times[0,\varepsilon_4]\times\cdots$. In 3-space this amounts to sweeping the two-dimensional scene in the direction of the third axis along a segment of length ε_3 (Figure 6**B**). If each dimension of the hyper-box is sufficiently small, the analysis seems to hold almost verbatim. Another possible extension is the “(hyper)cube-within-a-(hyper)cube” (Figure 6**C**). According to our initial analysis, it may no longer be true that the probabilistic bound is uniformly independent on the edge-size α of the inner (hyper)cube. However, if we are just interested in crafting an ASD to provide a probabilistic bound for a specific instance of the problem, the technique seems to allow it if we define the appropriate critical regions, although the exact details must be worked out and proved. Indeed, in experiments we conducted on this problem where the edge length of the inner-cube is 90% that of the bounding-cube edge (Figure 5**C**), the quality of solution paths was low (over 1.2 worse than optimal and typically much worse) in $97.3\%\pm0.4\%$ of runs (from a total of $2,000$ independent runs).

Other measures of path quality. Analogous ASD’s can be constructed for other common quality measures such as clearance from obstacles or combined weighted clearance-length measures. In particular, it seems that the case of average clearance or bottleneck clearance, where we wish to maximize the minimum clearance along a path, can be addressed by a simple adaptation of the current analysis, if we reduce the clearance of any Type-B solution (Figure 1) by narrowing down the width of the top passage in the Promenade scene, or alternatively, if we block sampling near the medial axis of the free space. The approach that uses the medial axis may lend itself more easily to extensions to higher dimensions.

Other motion-planning algorithms. The essence of the automaton formulation is the dual representation of motion-planning algorithms. This representation reduces the analysis of path quality to the identification of a set of critical events in the progress of such algorithms. We therefore anticipate that our method can ease the quality analysis of other motion-planning algorithms. In particular, it would be interesting to analyze other variants of *RRT*, as those suggested by Karaman and Frazzoli[10], along the lines presented here.

Richer automaton states and alphabets. Another way in which the Automaton of Sampling-Diagrams framework may be generalized to higher-dimensional configuration spaces is by using different types of automaton states or alphabets than the ones presented here, which will assist further to effectively identify critical events during the construction of roadmaps, by taking advantage of a richer set of predicates that can be used to analyze complex problems. In this respect, it may be helpful to borrow from the work on motion-planning with Linear-Temporal Logic (LTL) specifications, where various types of Finite-State Machines are used in conjunction with planning algorithms for the purpose of model checking (see, e.g., [2, 6, 15]).

7 Conclusions

In this study, we show for a common variant of the Bi-RRT algorithm that the probability for low quality paths is bounded away from zero. To the best of our knowledge, we present the first theoretical results on path quality of the Bi-RRT algorithm that are topological in nature in the sense that alternative paths lie in different homotopy classes and are therefore invariant to smoothing. We prove a wide gap between the optimal path, or even any path that is homotopy equivalent to the optimal path, and the quality of actual output paths. Our empirical results suggest that our bound is conservative, and it would be a worthy challenge to reach a tighter bound. An advantage of the automaton formalism is probably that it ignores many of the details involved in analyzing the running progress (perhaps at the cost of looser bounds).

In conclusion, we presented here one of the first theoretical results on the quality of output paths generated by sampling-based algorithm and the first to assess high-quality paths that occupy a non-negligible portion of the space of all paths. We anticipate that analysis of the type we propose here will facilitate the design of robust algorithms that also perform better in practice.

Acknowledgements. We thank Debora Nechushtan for creating the illustrations in Figure 6. This work has been supported in part by the Israel Science Foundation (grant no. 236/06), by the German-Israeli Foundation (grant no. 969/07), by the Hermann Minkowski–Minerva Center for Geometry at Tel Aviv University, and by the EU Project under Contract No 255827 (CGL—Computational Geometry Learning).

References

1. Asano, T., Kirkpatrick, D., Yap, C.: d_1-Optimal motion of a rod. In: 12th ACM Symposium on Computational Geometry, pp. 252–263 (1996)
2. Bhatia, A., Kavraki, L.E., Vardi, M.Y.: Sampling-based motion planning with temporal goals. In: ICRA 2010, pp. 2689–2696 (2010)
3. Canny, J., Reif, J.: New lower bound techniques for robot motion planning problems. In: FOCS 1987, pp. 49–60. IEEE, Los Alamitos (1987)
4. Choset, H., Lynch, K.M., Hutchinson, S., Kantor, G.A., Burgard, W., Kavraki, L.E., Thrun, S.: Principles of Robot Motion: Theory, Algorithms, and Implementations. MIT Press, Cambridge (2005)
5. Enosh, A., Raveh, B., Furman-Schueler, O., Halperin, D., Ben-Tal, N.: Generation, comparison and merging of pathways between protein conformations: Gating in k-channels. Biophysical Journal 95(8), 3850–3860 (2008)
6. Fainekos, G.E., Girard, A., Kress-Gazit, H., Pappas, G.J.: Temporal logic motion planning for dynamic robots. Automatica 45(2), 343–352 (2009)
7. Ferguson, D., Stentz, A.: Anytime RRTs. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 5369–5375 (2006)
8. Geraerts, R., Overmars, M.: Creating high-quality paths for motion planning. IJRR 26(8), 845–863 (2007)
9. Hsu, D., Latombe, J., Motwani, R.: Path planning in expansive configuration spaces. Int. J. Comp. Geo. & App. 4, 495–512 (1999)
10. Karaman, S., Frazzoli, E.: Incremental sampling-based optimal motion planning. Robotics: Science and Systems (2010)
11. Kavraki, L.E., Kolountzakis, M.N., Latombe, J.-C.: Analysis of probabilistic roadmaps for path planning. IEEE Trans. Robot. Automat. 14(1), 166–171 (1998)
12. Kavraki, L.E., Svestka, P., Latombe, J.C., Overmars, M.H.: Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Trans. Robot. Automat. 12(4), 566–580 (1996)
13. Kim, J., Pearce, R.A., Amato, N.M.: Extracting optimal paths from roadmaps for motion planning. In: ICRA 2003, pp. 2424–2429. IEEE, Los Alamitos (2003)
14. Kuffner, J.J., Lavalle, S.M.: RRT-Connect: An efficient approach to single-query path planning. In: ICRA 2000, pp. 995–1001 (2000)
15. Lahijanian, M., Wasniewski, J., Andersson, S., Belta, C.: Motion planning and control from temporal logic specifications with probabilistic satisfaction guarantees. In: ICRA 2010, pp. 3227–3232 (2010)
16. LaValle, S.M.: Planning Algorithms. Cambridge University Press, Cambridge (2006)
17. LaValle, S.M., Kuffner, J.J.: Rapidly-exploring random trees: Progress and prospects. In: Donald, B.R., Lynch, K.M., Rus, D. (eds.) Algorithmic and Computational Robotics: New Directions, pp. 293–308. A K Peters, Wellesley (2001)
18. Mitchell, J.S.B.: Handbook of discrete and computational geometry. Shortest paths and networks, ch. 27, pp. 607–641. CRC Press, Inc., Boca Raton (2004)
19. Nechushtan, O., Raveh, B., Halperin, D.: Supplementary online proofs, `http://acg.cs.tau.ac.il/projects/internal-projects/sda/SuppOnline.pdf`
20. Nieuwenhuisen, D., Overmars, M.H.: Useful cycles in probabilistic roadmap graphs. In: ICRA 2004, pp. 446–452. IEEE, Los Alamitos (2004)

21. Plaku, E., Bekris, K.E., Kavraki, L.E.: OOPS for motion planning: An online open-source programming system. In: ICRA 2007, pp. 3711–3716 (2007)
22. Raveh, B., Enosh, A., Halperin, D.: A little more, a lot better: Improving path quality by a simple path merging algorithm. Computing Research Repository, abs/1001.2391 (2010)
23. Reif, J., Wang, H.: The complexity of the two dimensional curvature-constrained shortest-path problem. In: WAFR 1998, pp. 49–57 (1998)
24. Reif, J.H.: Complexity of the mover's problem and generalizations. In: Proceedings IEEE Symposium on Foundations of Computer Science, pp. 421–427 (1979)

Sufficient Conditions for the Existence of Resolution Complete Planning Algorithms

Dmitry S. Yershov and Steven M. LaValle

Abstract. This paper addresses theoretical foundations of motion planning with differential constraints in the presence of obstacles. We establish general conditions for the existence of resolution complete planning algorithms by introducing a functional analysis framework and reducing algorithm existence to a simple topological property. First, we establish metric spaces over the control function space and the trajectory space. Second, using these metrics and assuming that the control system is Lipschitz continuous, we show that the mapping between open-loop controls and corresponding trajectories is continuous. Next, we prove that the set of all paths connecting the initial state to the goal set is open. Therefore, the set of open-loop controls, corresponding to solution trajectories, must be open. This leads to a simple algorithm that searches for a solution by sampling a control space directly, without building a reachability graph. A dense sample set is given by a discrete-time model. Convergence of the algorithm is proven in the metric of a trajectory space. The results provide some insights into the design of more effective planning algorithms and motion primitives.

1 Introduction

We consider the general problem of motion planning under both differential constraints and obstacles. A control system, geometric robot model, and a model of obstacles in the workspace are given. The task is to compute a control signal that brings a robot along a trajectory from an initial state into a goal region in a state space that may represent configurations and possibly their time derivatives. This problem is a

Dmitry S. Yershov
University of Illinois, Urbana, IL 61801
e-mail: yershov2@uiuc.edu

Steven M. LaValle
University of Illinois, Urbana, IL 61801
e-mail: lavalle@uiuc.edu

unification of several fundamental, classical problems in robotics: 1) *Nonholonomic planning:* In this case, the differential constraints may arise from wheeled mobile robots and planning occurs in the configuration space [1]; however, dynamics and constraints due to angular momentum may also be included. Such problems usually arise from *underactuated* (less controls than the number of degrees of freedom) systems. 2) *Kinodynamic planning:* Here, there are both velocity and acceleration bounds, and the system is fully actuated [2]. 3) *Trajectory planning:* This problem has been pursued for several decades [3, 4, 5] and typically involves computing an open loop control for a manipulator while satisfying the kinematics and dynamics expressed as a control system. See Chapter 14 of [6] for a detailed presentation of this unified class of problems.

In spite of all of this effort, there is still no general characterization of the particular conditions under which an algorithmic solution exists. Since basic motion planning (without differential constraints) is already PSPACE hard [7], and particular instances of motion planning with differential constraints are even harder, there is not much hope for efficient, complete solutions. In this context, *complete* means that the planning algorithm must return a solution whenever one exists; otherwise, report failure. Therefore, virtually all approaches to the problem are *sampling-based*, which employ discretization and heuristics to incrementally explore the state space by concatenating pieces of control signals to obtain a search tree of collision-free trajectories. In this case, the most we can hope for is *resolution completeness* [8], which means that the algorithm correctly finds a solution whenever one exists; however, it may run forever if one does not. This is analogous to classical Turing decidability vs. Turing recognizability, which are comparable to completeness and resolution completeness, respectively. We believe that having general conditions for the existence of resolution complete algorithms may be useful in the formulation of solvable robotics problems, in the design of better sampling-based planning algorithms, in the design of motion primitives [9, 10, 11], and possibly for the verification problem [12, 13], which is a negated form of planning that establishes path *non*existence.

$$\begin{tikzcd}
\text{Motion primitives} \arrow[d] & \text{Input space} \arrow[d] & \text{State space} \arrow[d] \\
\text{Finite length sequences of motion primitives} \arrow[r] & \text{Control space} \arrow[r] & \text{Trajectory space}
\end{tikzcd}$$

Fig. 1 Sets, spaces, and relations

In this paper, we determine simple, general requirements for the existence of a resolution complete planning algorithm based mainly on Lipschitz conditions on the control system mapping. This allows substantial generality and is inspired by analysis of the convergence of numerical dynamic programming algorithms [14, 15]. Our basic approach in this paper is to analyze the relationships between the six spaces shown informally in Fig. 1. The *input space* is the set of possible control system inputs and the *state space* is the configuration space or more generally the phase space of the system. Considering inputs and states parametrized over time, we design suitable metric spaces for both the space of control signals (called the *control space*) and the space of trajectories (the *trajectory space*). By establishing the continuity of the mapping between these spaces induced by the control system, we show that resolution completeness boils down to a simple topological condition. Furthermore, we constructively prove existence by providing a resolution complete algorithm that systematically enumerates candidate solution trajectories by concatenating sequences of *motion primitives*, which form a discrete set, suitable for computation. This is closely resembles the execution trace of most existing planning algorithms, which incrementally sample and search the state space (see [16, 6] for surveys).

Resolution completeness in this general setting is provided by ensuring that the set of all *computed* control signals is dense in the space of *all* control signals. One surprising observation, however, is that this may be achieved even where it is impossible to incrementally reduce the radius of the largest empty ball (*dispersion* [17]) in the space of control signals. This peculiar behavior is explained in Section 2, where basic sampling concepts are defined. Section 3 formally defines the general problem. Section 4 develops a continuous mapping from control space to trajectory space by carefully designing appropriate metric spaces. The main algorithmic constructions and theorems are presented in Section 5, which give sufficient conditions for the existence of a resolution complete planning algorithm. Conclusions appear in Section 6.

2 Precompactness and Sampling Convergence

Before providing the main technical results of the paper, a counterintuitive property regarding sampling and convergence needs to be addressed. Consider sampling-based planning algorithms for the basic motion planning problem (no differential constraints). Achieving resolution completeness (or alternatively, probabilistic completeness) amounts to assuring that the computed samples are dense in the limit as the number of iterations tends to infinity. Intuitively, the sampling resolution gradually increases over time. This notion can be nicely captured by defining *dispersion* of a sample set P in a subset of any metric space X [17]:

$$\delta(P) = \sup_{x\in X}\{\inf_{p\in P}\{\rho(x,p)\}\}\ , \tag{1}$$

in which ρ is the metric.

Typically, the sample set increases gradually during the execution of a planning algorithm, and the dispersion converges to zero in the limit. This is formalized by considering P as an infinite *sample sequence* P, which is a set together with a specified linear ordering. We say that P *converges* to X if and only if

$$\lim_{N\to\infty} \delta(P|_N) = 0 \,, \tag{2}$$

in which $P|_N$ denotes the first N elements of P. Note that the convergence rate may depend on the ordering.

In the coming sections, we consider notions of denseness, dispersion, and convergence over the function spaces of control signals and trajectories. For these spaces, it might be surprising that there are dense sample sequences for which the dispersion does not converge to zero. In other words, the samples may eventually get arbitrarily close to every point in the space, even though they are not converging to that space. For a simple example of this behavior, let $X = \mathbb{R}$. For any finite set of samples in $\mathbb{R}$, the dispersion is always infinite. Nevertheless, any ordering placed on $\mathbb{Q}$, the set of rationals, yields a sequence that is dense[1] in $\mathbb{R}$. For a bounded space, such as S^1 and the configuration spaces arising in robotics, this behavior does not occur: A dense sequence must drive the dispersion to zero.

Here is a critical question for sampling-based planning with differential constraints: What property must X have to enable convergence? Define X to be *precompact* if and only if for any $\varepsilon > 0$ there exists a finite cover of X with open balls of radius ε. The following lemma answers the question.

Lemma 1. *A sequence P that is dense in X converges to X if and only if X is precompact.*

Proof. We prove necessary and sufficient conditions separately.

Necessary condition: Assume P is convergent, and let ε be greater than zero. Find N_ε such that $\delta(P|_{N_\varepsilon}) < \varepsilon$. Consider a set of open balls of radius ε centered at points of $P|_{N_\varepsilon}$. From the definition of dispersion, it follows that this set of balls is a finite cover of X. Since ε is arbitrary, X is precompact by definition.

Sufficient condition: Suppose, on the other hand, that X is precompact. Thus, for any $\varepsilon > 0$, there exists a finite cover of X with open balls of radius ε. Denote these balls as B_i. Since the sequence P is dense, the intersection of P with each B_i is nonempty. Take the smallest N_ε such that $P|_{N_\varepsilon}$ has at least one element in each B_i. By construction, the dispersion of $P|_{N_\varepsilon}$ in X is bounded by 2ε. Since ε is arbitrary and for $N > N_\varepsilon$ we have $\delta(P|_N) \le \delta(P|_{N_\varepsilon})$, the sequence converges. $\square$

Note that this is a property of the *space* X, and not a particular sample sequence. Our analysis will demonstrate that under general conditions space of all control signals is not precompact. Therefore, it is impossible to achieve convergence in the control space. However, we will show precompactness of the space of finite-time trajectories. In this case, the denseness implies convergence, which means that if a

[1] Here, dense means that the topological closure of $\mathbb{Q}$ is all of $\mathbb{R}$.

solution is not found in a finite number of steps, it either does not exist or the goal set must be smaller than the reached dispersion of the sampled trajectories.

3 Problem Definition

Let the *state space*, $X \subset \mathbb{R}^m$, be a smooth manifold of dimension n. Let U be the *input space*, which is a compact subset of $\mathbb{R}^k$ with $k \leq n$. A given mechanical system is expressed in local coordinates as[2]

$$\dot{x} = f(x,u) \ , \tag{3}$$

in which $\dot{x} = dx/dt$. Also, in the equation above, $x \in X$ and $u \in U$.

It is assumed that f is a *Lipschitz continuous* function in both x and u, which implies that there exists positive real-valued constants L_x and L_u such that

$$\big\|f(x,u) - f(x',u)\big\| \leq L_x\|x - x'\| \quad \text{and} \quad \big\|f(x,u) - f(x,u')\big\| \leq L_u\|u - u'\| \tag{4}$$

for all $x\,, x' \in X$, and $u\,, u' \in U$. The norms used here are defined on the ambient vector spaces $\mathbb{R}^m$ and $\mathbb{R}^k$, respectively. Furthermore, it is assumed that there exists $M > 0$ such that

$$\|f(x,u)\| \leq M \tag{5}$$

for all $x \in X$ and $u \in U$.

Let $\mathscr{U}$ be the set of all measurable functions, $\tilde{u}$, defined on $[0\,,T]$, for all $T \in [0\,,\infty)$, with values in U. Similarly, denote $\mathscr{X}$ to be the set of all Lipschitz continuous functions, $\tilde{x}$, defined on $[0\,,T]$, for $T \geq 0$, with values in X. We require that for all functions in $\mathscr{X}$ the Lipschitz constant is bounded by M. In other words, for any element $\tilde{x} \in \mathscr{X}$ and any given t and t' in the domain of $\tilde{x}$, we have $\|\tilde{x}(t) - \tilde{x}(t')\| \leq M|t - t'|$. Also, define $\tau : \mathscr{U} \cup \mathscr{X} \to [0\,,\infty)$ to return the duration of a control or a trajectory, depending on the argument.

Constraints are imposed on X that account for mechanical limits due to kinematics and dynamics, and also to avoid collisions with static obstacles. Let X_{free} denote an open and bounded subset of X that consists of all states satisfying these constraints. Usually, X_{free} is defined only implicitly via representations of the kinematics and obstacles. Therefore, a *collision detection* algorithm is often needed to evaluate whether states lie in X_{free}.

The *planning problem* is as follows. Assume X, U, and f are given. Furthermore, an *initial state*, $x_{\text{I}} \in X_{\text{free}}$, and open goal set $X_{\text{G}} \subseteq X_{\text{free}}$ are specified. The problem is to compute $\tilde{u} \in \mathscr{U}$ such that for the corresponding $\tilde{x}$, satisfying (3) with given $\tilde{u}$, the following is true: 1) $\tilde{x}(0) = x_{\text{I}}$, 2) $\tilde{x}(\tau(\tilde{u})) \in X_{\text{G}}$, and 3) the image of $\tilde{x}$ lies in X_{free}.

To accomplish this task, we assume the existence of an *integration module* which integrates (3) to produce trajectory segments, and a *collision detection module* which determines whether a trajectory segment lies entirely in X_{free}.

[2] We may consider more general case of time-varying systems $\dot{x} = f(x,u,t)$, without changing further analysis in the paper. We choose the time-invariant case for notational convenience.

4 Preliminary Concepts and Properties

In this section we introduce some preliminary concepts that are necessary to establish basic properties for an algorithm to be resolution complete and convergent. We define metrics on the space of controls and the trajectory space, and show that the relation between controls and trajectories is a well-defined continuous function.

4.1 Designing Metric Spaces on $\mathscr{U}$ and $\mathscr{X}$

The control space, $\mathscr{U}$, can be made into a metric space as follows. Let α be the diameter of the smallest ball that contains U. We call α the *diameter* of U. It is easy to verify $\alpha < \infty$ because U is assumed to be compact. For two controls $\tilde{u}$ and $\tilde{u}'$ in $\mathscr{U}$ define the L_1-type metric

$$\rho_{\mathscr{U}}(\tilde{u},\tilde{u}') = \int_0^T \|\tilde{u}(t) - \tilde{u}'(t)\| \, \mathrm{d}t + \alpha|\tau(\tilde{u}) - \tau(\tilde{u}')| \,, \tag{6}$$

in which $T = \min(\tau(\tilde{u}), \tau(\tilde{u}'))$.

Note that this metric is different from a standard L_1 metric due to variable domain length of functions in $\mathscr{U}$, which is accounted by an additional term in (6). The extra term separates any meaningful controls from the control defined on the zero length time interval, we call it *zero control.*

The choice of the metric is motivated by Figs. 2a and 2b. Consider driving a car around a corner. The trajectory deviates only slightly if steering is applied with small delays. Moreover, the trajectory deviation depends on the delay continuously. The introduced L_1-type metric (6) captures this behavior.

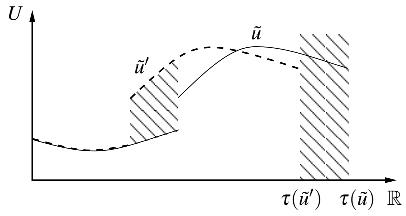
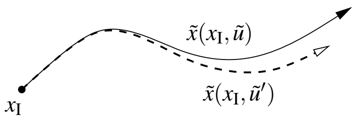


Fig. 2a Control signals. The area of the shaded regions corresponds to $\rho_{\mathscr{U}}(\tilde{u},\tilde{u}')$



$\tilde{x}(x_{\mathrm{I}},\tilde{u})$
$\tilde{x}(x_{\mathrm{I}},\tilde{u}')$
x_{I}

Fig. 2b Corresponding trajectories

To save space, we omit the (tedious) proof of the following lemma:

Lemma 2. *The control space $\mathscr{U}$ is a metric space*[3] *with respect to (6).*

[3] More precisely, it is a pseudometric space [18] because $\rho_{\mathscr{U}}(\tilde{u},\tilde{u}') = 0$ for some $\tilde{u} \neq \tilde{u}'$. However, if two controls are identified in case their distance is zero, then the resulting space is a metric space.

Next we describe a metric on the space of all trajectories, $\mathscr{X}$. For two trajectories $\tilde{x}$ and $\tilde{x}'$ define the L_∞-type metric

$$\rho_{\mathscr{X}}(\tilde{x},\tilde{x}') = \sup_{0\le t\le T}\left\{\|\tilde{x}(t)-\tilde{x}'(t)\|\right\} + M|\tau(\tilde{x})-\tau(\tilde{x}')| \;, \tag{7}$$

in which $T=\min(\tau(\tilde{x}),\tau(\tilde{x}'))$.

On the trajectory space, an L_1-type metric cannot be used because it ignores "spikes" in the trajectory deviation. The two trajectories illustrated in Fig. 3a are "close" in terms of the L_1-type metric. However, they exhibit qualitatively different behavior; the first one does not intersect the obstacle and the second one does. Moreover, it is possible to find two trajectories arbitrary close in L_1-type metric, but with arbitrary large deviation between them. On the other hand, later we will show that two trajectories which are "close" in the L_∞-type metric behave similarly; see Fig. 3b.

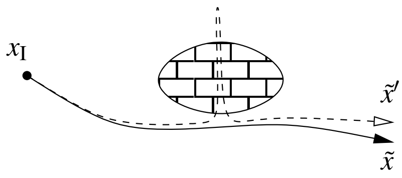


Fig. 3a L_1-type metric

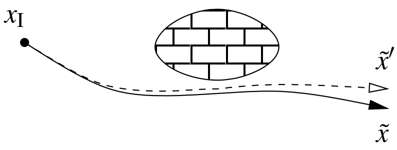


Fig. 3b L_∞-type metric

Lemma 3. *The trajectory space $\mathscr{X}$ is a metric space with respect to (7).*

(The proof is omitted to save space.)

4.2 Relating Controls to Trajectories

Next, we analyze the mapping between a control $\tilde{u}\in\mathscr{U}$ and the corresponding trajectory originating from some $x_0\in X$. Denote the mapping as a function $\tilde{x}(x_0,\tilde{u}):X\times\mathscr{U}\to\mathscr{X}$. Note that for fixed $\tilde{u}$ and x_0, the trajectory $\tilde{x}(x_0,\tilde{u})$ is a function of time and it satisfies the integral equation

$$\tilde{x}(x_0,\tilde{u},t) = x_0 + \int_0^t f(\tilde{x}(x_0,\tilde{u},s),\tilde{u}(s))\,\mathrm{d}s \;, \tag{8}$$

which is equivalent to (3). In the integral equation, $\tilde{x}(x_0,\tilde{u},t)$ denotes the point of the trajectory $\tilde{x}(x_0,\tilde{u})$ at time t.

Lemma 4 (Well-defined trajectories). *For any initial state $x_0\in X$ and any control signal $\tilde{u}\in\mathscr{U}$, the trajectory $\tilde{x}(x_0,\tilde{u})$ belongs to $\mathscr{X}$.*

Proof. For any given $\tilde{u}\in\mathscr{U}$, function $f(x,\tilde{u}(t))$ as a function of x and t satisfies the Caratheodory conditions. Hence, the solution for the differential equation (3), with initial value at any x_0, exists, is unique, and is absolutely continuous on $[0\,,\,\tau(\tilde{u})]$. We now use the integral equation (8) to prove that $\tilde{x}(x_0,\tilde{u},t)$ is Lipschitz continuous as a function of time. Consider

$$\left\|\tilde{x}(x_0,\tilde{u},t)-\tilde{x}(x_0,\tilde{u},t')\right\| \leq \left\|\int_{t'}^{t} f(\tilde{x}(x_0,\tilde{u},s),\tilde{u}(s))\,\mathrm{d}s\right\| \leq M|t-t'|\,. \tag{9}$$

Note that the Lipschitz constant is bounded by M; therefore, $\tilde{x}(x_0,\tilde{u})$ is in $\mathscr{X}$. □

To show the continuity of $\tilde{x}(x_0,\tilde{u})$, we first prove the following lemma.

Lemma 5 (Bounded trajectory deviation).
Let $\tilde{u}$ and $\tilde{u}'$ be two independent controls, with $\tau(\tilde{u})=\tau(\tilde{u}')=T$. Assume further that $\tilde{x}=\tilde{x}(x_0,\tilde{u})$ and $\tilde{x}'=\tilde{x}(x_0',\tilde{u}')$ are the corresponding trajectories. The deviation between trajectories $\tilde{x}$ and $\tilde{x}'$ at any time $t\in[0\,,T]$ is bounded by

$$\|\tilde{x}(t)-\tilde{x}'(t)\| \leq \Big(\|x_0-x_0'\|+L_u\int_0^t \exp(-L_x s)\|\tilde{u}(s)-\tilde{u}'(s)\|\,\mathrm{d}s\Big)\exp(L_x t)\,. \tag{10}$$

Proof. Trajectories $\tilde{x}$ and $\tilde{x}'$ satisfy the integral equation (8), with $\tilde{u}$ and $\tilde{u}'$, respectively. Using the integral equation and the Lipschitz continuity of f, we derive the integral inequality for the trajectory deviation

$$\begin{aligned}\|\tilde{x}(t)-\tilde{x}'(t)\| &= \left\|x_0-x_0'+\int_0^t \Big[f\big(\tilde{x}(s),\tilde{u}(s)\big)-f\big(\tilde{x}'(s),\tilde{u}'(s)\big)\Big]\,\mathrm{d}s\right\| \\ &\leq \|x_0-x_0'\|+\int_0^t L_u\|\tilde{u}(s)-\tilde{u}'(s)\|+L_x\|\tilde{x}(s)-\tilde{x}'(s)\|\,\mathrm{d}s\,.\end{aligned} \tag{11}$$

From the integral form of Gronwall-Bellman inequality [19] a bound on the trajectory deviation at any time $t\in[0\,,T]$ follows as

$$\begin{aligned}\|\tilde{x}(t)-\tilde{x}'(t)\| \leq \|x_0-x_0'\|\exp(L_x t)+L_u\int_0^t \|\tilde{u}(s)-\tilde{u}'(s)\|\,\mathrm{d}s \\ +L_xL_u\int_0^t\int_0^r \exp(L_x(t-r))\|\tilde{u}(s)-\tilde{u}'(s)\|\,\mathrm{d}s\,\mathrm{d}r\,.\end{aligned} \tag{12}$$

The double integral is reduced to a single integral by applying Fubini's theorem [18] to obtain

$$\begin{aligned}&\int_0^t\int_0^r \exp(L_x(t-r))\|\tilde{u}(s)-\tilde{u}'(s)\|\,\mathrm{d}s\,\mathrm{d}r \\ &\qquad=\int_0^t\int_s^t \exp(L_x(t-r))\|\tilde{u}(s)-\tilde{u}'(s)\|\,\mathrm{d}r\,\mathrm{d}s \\ &\qquad\qquad=\frac{1}{L_x}\int_0^t \|\tilde{u}(s)-\tilde{u}'(s)\|\big(\exp(L_x(t-s))-1\big)\,\mathrm{d}s\end{aligned} \tag{13}$$

The combination of the results above finishes the proof. □

The next theorem establishes the continuity of the function $\tilde{x}(x_0, \tilde{u})$ with respect to both x_0 and $\tilde{u}$.

Theorem 1 (Continuity of $\tilde{x}(x_0, \tilde{u})$). *The mapping $\tilde{x}(x_0, \tilde{u})$ is continuous.*

Proof. Take two initial points x_0 and x_0' in X and two control signals $\tilde{u}$ and $\tilde{u}'$ in $\mathscr{U}$. Let $\tilde{x}$ and $\tilde{x}'$ be defined as in Lemma 5, and let $T^* = \min(\tau(\tilde{x}), \tau(\tilde{x}'))$. It follows from Lemma 5 that for all $t \in [0\,, T^*]$

$$\|\tilde{x}(t) - \tilde{x}'(t)\| \leq \Big(\|x_0 - x_0'\| + L_u \int_0^t \exp(-L_x s)\|\tilde{u}(s) - \tilde{u}'(s)\|\,\mathrm{d}s\Big)\exp(L_x t)\,. \quad (14)$$

Take the supremum over the interval $[0\,, T^*]$ on both sides and derive

$$\sup_{0 \leq t \leq T^*} \|\tilde{x}(t) - \tilde{x}'(t)\| \leq \exp(L_x T^*)\Big(\|x_0 - x_0'\| + L_u \int_0^{T^*} \|\tilde{u}(s) - \tilde{u}'(s)\|\,\mathrm{d}s\Big)\,. \quad (15)$$

Using the derivation above, we bound the distance between trajectories in terms of the distances between initial values and control signals

$$\begin{aligned}\rho_{\mathscr{X}}(\tilde{x}, \tilde{x}') &= \sup_{0 \leq t \leq T^*} \|\tilde{x}(x_0, \tilde{u}, t) - \tilde{x}(x_0', \tilde{u}', t)\| + M|T - T'| \\ &\leq \exp(L_x T^*)\Big(\|x_0 - x_0'\| + L_u \int_0^{T^*} \|\tilde{u}(s) - \tilde{u}'(s)\|\,\mathrm{d}s\Big) + \frac{M}{\alpha}\alpha|T - T'| \\ &\leq \exp(L_x T^*)\|x_0 - x_0'\| + \max\Big(L_u \exp(L_x T^*), \frac{M}{\alpha}\Big)\rho_{\mathscr{U}}(\tilde{u}, \tilde{u}')\,. \quad (16)\end{aligned}$$

The inequality above proves continuity of the map. □

Since $\mathscr{X}$ and $\mathscr{U}$ are metric spaces and $\tilde{x}(x_0, \tilde{u})$ is continuous, we may employ topological methods to analyze properties of subsets and sequences in control and trajectory spaces, as well as relations between them.

4.3 Topological Properties

We next address properties of the set of *collision-free* paths connecting x_{I} with X_{G}. Consider $\mathscr{X}_{\mathrm{I}}$, the subset of $\mathscr{X}$ containing all trajectories that originate from x_{I}. Define the induced subset topology on $\mathscr{X}_{\mathrm{I}}$. Also, consider the set $\mathscr{X}_{\mathrm{sol}}$ of solutions to the path planning problem, which consists of all paths in $\mathscr{X}$ that originate from x_{I}, terminate in X_{G}, with the image contained in X_{free} (collision-free). Note that an element of $\mathscr{X}_{\mathrm{sol}}$ may not be necessarily a trajectory governed by the system (3).

Theorem 2. *If X_{free} and X_{G} are open in X, then $\mathscr{X}_{\mathrm{sol}}$ is an open subset of $\mathscr{X}_{\mathrm{I}}$.*

Proof. Let $\tilde{x}$ be in $\mathscr{X}_{\mathrm{sol}}$, and let $T = \tau(\tilde{x})$. According to the definition, the image of $\tilde{x}$ is contained in X_{free} and the terminal point, $\tilde{x}(T)$, is in X_{G}. Since the image of $\tilde{x}$ is compact and the complement of X_{free} is closed, the distance between these two sets,

δ_1, is well-defined and strictly positive. Moreover, the distance from the terminal point to the complement of X_G, δ_2, is also well-defined and positive.

Consider $\delta = \min(\delta_1, \delta_2)$ and any trajectory $\tilde{x}'$ from $\mathscr{X}_I$ such that $\rho_{\mathscr{X}}(\tilde{x}, \tilde{x}') < \delta$. First, we show that the image of $\tilde{x}'$ is in X_{free}. Assume to the contrary that there exists some $t' \in [0\,,\, \tau(\tilde{x}')]$ such that $\tilde{x}'(t') \notin X_{\text{free}}$. Let $t = \min(t', T)$. The bound

$$\begin{aligned} \|\tilde{x}(t) - \tilde{x}'(t')\| &\leq \|\tilde{x}(t) - \tilde{x}'(t)\| + \|\tilde{x}'(t) - \tilde{x}'(t')\| \\ &\leq \|\tilde{x}(t) - \tilde{x}'(t)\| + M|t - t'| \\ &\leq \rho_{\mathscr{X}}(\tilde{x}, \tilde{x}') < \delta \leq \delta_1 \end{aligned} \tag{17}$$

contradicts the definition of δ_1.

Second, we prove that $\tilde{x}'(T') \in X_G$, in which $T' = \tau(\tilde{x}')$. Assume to the contrary that $\tilde{x}'(T') \notin X_G$. Let $T^* = \min(T, T')$ and consider the bound

$$\begin{aligned} \|\tilde{x}(T) - \tilde{x}'(T')\| &\leq \|\tilde{x}(T) - \tilde{x}(T^*)\| + \|\tilde{x}(T^*) - \tilde{x}'(T^*)\| + \|\tilde{x}'(T^*) - \tilde{x}'(T')\| \\ &\leq M|T - T^*| + \|\tilde{x}(T^*) - \tilde{x}'(T^*)\| + M|T^* - T'| \\ &= \|\tilde{x}(T^*) - \tilde{x}'(T^*)\| + M|\max(T', T) - \min(T', T)| \\ &\leq \rho_{\mathscr{X}}(\tilde{x}, \tilde{x}') < \delta \leq \delta_2\,, \end{aligned} \tag{18}$$

which contradicts the definition of δ_2. By definition, $\tilde{x}'$ belongs to $\mathscr{X}_{\text{sol}}$. □

Denote a set of solutions to the motion planning problem as $\mathscr{U}_{\text{sol}}$. Clearly, for any given $\tilde{u}$ in $\mathscr{U}_{\text{sol}}$, the trajectory $\tilde{x}(x_I, \tilde{u})$ belongs to $\mathscr{X}_{\text{sol}}$. Therefore, $\mathscr{U}_{\text{sol}}$ is a preimage of $\mathscr{X}_{\text{sol}}$ with respect to $\tilde{x}(x_I, \tilde{u}) : \mathscr{U} \to \mathscr{X}_I$. From the continuity of $\tilde{x}(x_I, \tilde{u})$ and Theorem 2 it follows that $\mathscr{U}_{\text{sol}}$ is open. We state this result as a separate theorem:

Theorem 3. *Assume all the conditions of Section 3 are met, then $\mathscr{U}_{\text{sol}}$ is open.*

5 Planning Algorithm and Resolution Completeness Conditions

In this section we establish sufficient conditions under which resolution complete algorithms to the motion planning problem exist. A simple resolution complete algorithm is constructed. We also prove the convergence of sampling-based algorithms in the trajectory space.

5.1 Controls via Concatenation of Primitives

Let $\Sigma \subset \mathscr{U}$ be a countable set of *motion primitives*, each defined over a closed and bounded time interval. Borrowing concepts from the theory of computation, Σ can be interpreted as an *alphabet*. If motion primitives are applied in succession, a control that represents their *concatenation* is obtained. For example, if $\sigma_1, \sigma_2 \in \Sigma$, in which $\sigma_1 : [0\,,\, t_1] \to U$ and $\sigma_2 : [0\,,\, t_2] \to U$, are applied in succession, the resulting control, denoted by $\sigma_1\sigma_2$ is

$$(\sigma_1\sigma_2)(t) = \begin{cases} \sigma_1(t) & \text{if } t \in [0, t_1) \\ \sigma_2(t-t_1) & \text{if } t \in [t_1, t_1+t_2] \end{cases}. \tag{19}$$

Allowing any finite number of concatenations, each resulting control can be expressed as a *string*, which is a finite sequence of motion primitives in Σ. Considering this, the set of all controls that can be formed from motion primitives is the *Kleene star* of Σ, which is denoted and defined as

$$\Sigma^* = \{\sigma_1\sigma_2\cdots\sigma_k \mid k \geq 0 \text{ and each } \sigma_i \in \Sigma\}. \tag{20}$$

Note that we do not allow infinite sequences of motion primitives to be applied. The definition of Σ^* allows the empty string, which is assumed to be zero control.

The following lemma establishes a useful property of Σ^* for the purposes of computation.

Lemma 6 (Rectangular enumeration argument). *For any set, Σ, of motion primitives, the set, Σ^*, of all strings is countable.*

Proof. Consider Σ_n^*, which consists of all strings of length not greater than n and composed of any characters of the alphabet $\Sigma|_n$. For example, $\Sigma_1^* = \{\sigma_1\}$, $\Sigma_2^* = \{\sigma_1, \sigma_2, \sigma_1\sigma_1, \sigma_1\sigma_2, \sigma_2\sigma_1, \sigma_2\sigma_2\}$, and so on; see Fig. 4. Verify that each Σ_n^* is finite, and $\Sigma^* = \bigcup_{n=1}^{\infty} \Sigma_n^*$. Hence, Σ^* is countable as a countable union of finite sets. $\square$

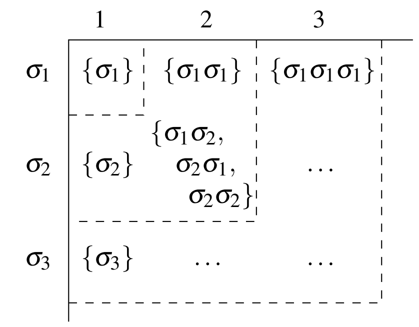


Fig. 4 Even if Σ is countably infinite, Σ^* is countably infinite, as shown by rectangular enumeration argument. We show that $\Sigma^* = \bigcup_{n=1}^{\infty} \Sigma_n^*$, in which sets Σ_n^* correspond to regions bounded by dashed lines and finite.

To facilitate the development of resolution complete planning algorithms, it will be helpful to introduce a set of motion primitives that is straightforward to describe and utilize. Moreover, assuming that all motion primitives in the set Σ are encoded digitally, it follows from the lemma above that all strings in Σ^* are computable.

Suppose that a system is defined as in (3). First, replace U with a countable subset. If U is uncountably infinite, then choose a countable, dense subset $U_\mathrm{d} \subset U$. For example, U_d could be the set of all $u \in U$ for which all coordinates are rational numbers. If U is already countable, then we may simply let $U_\mathrm{d} = U$.

Let $\Sigma_\mathrm{dt} \subset \mathscr{U}$ be called the *discrete-time model*, and be defined as the set of all constant functions, $\tilde{u} : [0, t] \to U_\mathrm{d}$, in which $t = 1/2^i$ for all $i \in \mathbb{N}$. Thus, the duration of motion primitives can be $1/2$, $1/4$, $1/8$, and so on. Any sequence of time intervals that converges to zero may alternatively be used. The set of all strings that utilizes the alphabet Σ_{dt} and the concatenation rule (19) consists of piecewise constant functions. We denote this set of strings as Σ_{dt}^*.

The discrete-time model is just one of numerous possible sets of motion primitives. However, the particular choice of Σ depends heavily on the considered system, the intended application, and the efficiency of the planning algorithm. Virtually any definition of Σ is allowed, provided that Σ^* is dense in $\mathscr{U}$. We will show that this requirement is sufficient for a resolution complete algorithm to exist.

5.2 *Planning Algorithm*

Based on the background results of Section 4, we are now ready to establish the existence of resolution complete algorithms in a very general setting. The existence is demonstrated by construction of a simple string enumeration algorithm; refer to Algorithm 1.

Algorithm 1. String enumeration algorithm

Input : a set of motion primitives Σ, the initial state x_{I}, a goal set X_{G}, a collision detection module, an integration module.
Output : a solution control $\tilde{\sigma} \in \Sigma^*$.

```
1: $n \leftarrow 0$
2: loop
3:     $\varepsilon_{\mathrm{i}} \leftarrow 1/2^n$ ; $n \leftarrow n+1$
4:     $\tilde{\sigma} \leftarrow$ select the $n$th string from $\Sigma^*$
5:     $\hat{x}(x_{\mathrm{I}}, \tilde{\sigma}) \leftarrow$ integrate the trajectory, starting from $x_{\mathrm{I}}$ under the control $\tilde{\sigma}$, with tolerance $\varepsilon_{\mathrm{i}}$
6:     $C_1 \leftarrow$ (true or false) determine whether the error cone around $\hat{x}(x_{\mathrm{I}}, \tilde{\sigma})$ is contained in $X_{\mathrm{free}}$
7:     $C_2 \leftarrow$ (true or false) determine whether the error cone of $\hat{x}(x_{\mathrm{I}}, \tilde{\sigma})$ terminates in $X_{\mathrm{G}}$
8:     if $C_1 \wedge C_2$ then
9:         return $\tilde{\sigma}$
10:    end if
11: end loop
```

The selection operation 4 is required to be *systematic*, which means that strings are selected so that all of Σ^* is enumerated as the number of iterations tends to infinity. This is always possible because Σ^* is countable.

For the integration line 5, validation line 6, and termination line 7, we would ideally like to have them executed in constant time with perfect precision. In practice, however, this is usually not possible. Most often, a *numerical* integration is necessary, which causes errors to propagate to the remaining two operations. Due to these limitations, an alternative will be defined: 1) An exact computation model, and 2) a numerical computation model.

Let the *exact computation model* refer to the case in which all operations are performed exactly in finite time without errors. This model is the simplest to analyze; however, it is also less realistic. Note that for the exact computational model the parameter ε_i is ignored in lines 5, 6, and 7 of the algorithm.

To account for errors arising from a numerical integration error, let the *numerical computation model* be defined as shown in Fig. 5. Assume that the precision of the numerical integration algorithm can be tuned using a parameter $\varepsilon_\mathrm{i} \in (0,\infty)$, and the error of the numerical trajectory is bounded by

$$\|\tilde{x}(x_0,\tilde{u},t) - \hat{x}(x_0,\tilde{u},t)\| \le \varepsilon_\mathrm{i} F(t) \, , \tag{21}$$

in which F is a strictly positive, monotonic function of time, and $\hat{x}(x_0,\tilde{u})$ is the numerical trajectory.

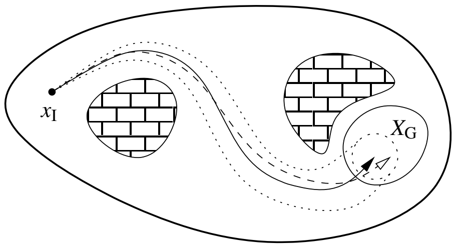


Fig. 5 The numerical computation model. The dotted region is the error cone constructed around the numerically integrated trajectory (solid line), which contains the exact trajectory (dashed line).

The proposed numerical error model allows us to construct an error cone (around the numerical trajectory), which contains the exact trajectory; see Fig. 5. The error cone is then used for validation and termination purposes. It will be assumed that validation and termination algorithms are conservative, that is, the algorithm must *reject* a candidate solution if there is any possibility that the trajectory leaves X_free or fails to terminate in X_G. Moreover, it may also be possible to reject strings under the numerical computation model by determining that *all* possible trajectories either leave X_free or fail to terminate in X_G. However, this will not be considered in detail in this paper.

5.3 Sufficient Conditions for Resolution Completeness

We derive a set of sufficient conditions which guarantee that the proposed algorithm is resolution complete. The result is demonstrated for both exact and numerical computational models. Furthermore, we demonstrate that the discrete-time model satisfies these sufficient conditions.

Theorem 4 (Sufficient conditions for the exact computational model). *If Σ^* is dense in $\mathscr{U}$, then Algorithm 1 is resolution complete under the exact computational model.*

Proof. We have already shown that the set $\mathscr{U}_{\mathrm{sol}} \subset \mathscr{U}$ is open, and in case a solution exists, $\mathscr{U}_{\mathrm{sol}}$ is nonempty. Since Σ^* is dense, its intersection with $\mathscr{U}_{\mathrm{sol}}$ is also nonempty. Hence, the algorithm, selecting systematically strings from Σ^*, eventually finds an element that is also in $\mathscr{U}_{\mathrm{sol}}$. Under the exact computational model, the selected string is accepted. □

Theorem 5 (Sufficient conditions for the numerical computational model). *If Σ^* is dense in $\mathscr{U}$, then Algorithm 1 is resolution complete under the numerical computational model.*

Proof. Let $\tilde{u} \in \mathscr{U}_{\mathrm{sol}}$ be a solution to the motion planning problem and ε be a positive constant. Furthermore, let $\delta = \varepsilon / \big(2 \max \left[L_u \exp(L_x \tau(\tilde{u})), M/\alpha\right]\big)$ and $N \in \mathbb{N}$ be such that $1/2^N < \varepsilon / \big(4F(\tau(\tilde{u}) + \delta/\alpha)\big)$. Find $\tilde{\sigma} \in \Sigma^*$, such that its number is greater than N and $\rho_{\mathscr{U}}(\tilde{u}, \tilde{\sigma}) < \delta$. It is always possible to do so because Σ^* is assumed to be dense in $\mathscr{U}$. With such choice of $\tilde{\sigma}$, it follows that $|\tau(\tilde{u}) - \tau(\tilde{\sigma})| < \delta/\alpha$, and, following the proof of Theorem 1, the distance $\rho_{\mathscr{X}}(\tilde{x}(x_{\mathrm{I}}, \tilde{u}), \tilde{x}(x_{\mathrm{I}}, \tilde{\sigma})) < \varepsilon/2$.

Consider the distance between the exact trajectory, $\tilde{x}(x_{\mathrm{I}}, \tilde{u})$, and the numerically integrated trajectory, $\hat{x}(x_{\mathrm{I}}, \tilde{\sigma})$:

$$\begin{aligned}\rho_{\mathscr{X}}(\tilde{x}(x_{\mathrm{I}}, \tilde{u}), \hat{x}(x_{\mathrm{I}}, \tilde{\sigma})) &\leq \rho_{\mathscr{X}}(\tilde{x}(x_{\mathrm{I}}, \tilde{u}), \tilde{x}(x_{\mathrm{I}}, \tilde{\sigma})) + \rho_{\mathscr{X}}(\tilde{x}(x_{\mathrm{I}}, \tilde{\sigma}), \hat{x}(x_{\mathrm{I}}, \tilde{\sigma})) \\ &\leq \frac{\varepsilon}{2} + \varepsilon_{\mathrm{i}} F(\tau(\tilde{\sigma})) \leq \frac{\varepsilon}{2} + \frac{\varepsilon F(\tau(\tilde{\sigma}))}{4F(\tau(\tilde{u}) + \delta/\alpha)} \leq \frac{\varepsilon}{2} + \frac{\varepsilon}{4} \leq \frac{3\varepsilon}{4}\,. \end{aligned} \tag{22}$$

Furthermore, note that the error cone around $\hat{x}(x_{\mathrm{I}}, \tilde{\sigma})$ stays in $\varepsilon/4$-neighborhood of the numerical trajectory. Hence, the error cone is contained in the ε-neighborhood of $\tilde{x}(x_{\mathrm{I}}, \tilde{u})$. By adjusting ε, which is arbitrary, we can ensure that the error cone is in $\mathscr{X}_{\mathrm{sol}}$. Therefore, $\tilde{\sigma}$ is accepted by Algorithm 1. □

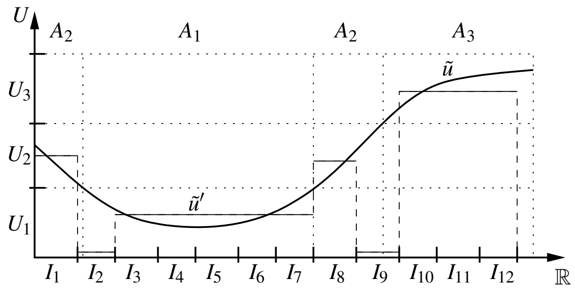
Now that we have a simple test to determine whether Algorithm 1 is resolution complete, we apply it to the discrete-time model.

Theorem 6 (Denseness theorem). *The set Σ^*_{dt} is dense in $\mathscr{U}$.*

Proof. Intuitively, the proof follows from the well known fact that piecewise constant functions are dense in the space of measurable functions [20]. Unfortunately, the discrete-time model is only a proper subset of the set of all piecewise constant functions. Here we outline our proof that overcomes this difficulty. Refer to Fig. 6 for details.

Consider any $\tilde{u}$ in $\mathscr{U}$ and ε greater than zero. Assume that U is partitioned with a collection of measurable sets $\{U_i\}$ with nonempty interior such that the diameter of each set is less than $\varepsilon / \big(2\tau(\tilde{u})\big)$. Let A_i be a preimage of U_i with respect to $\tilde{u}$. Since $\tilde{u}$ is a measurable function and U_i is a measurable set, A_i is measurable.

Next, consider the approximation of A_is from within by intervals of length $1/2^N$; denote these intervals as $I_j = [(j-1)/2^N, j/2^N]$. Assume the tolerance of the approximation is less than $\varepsilon/(2\alpha)$, collectively for all A_is. This means that the measure of the difference between A_i and all intervals which are subsets of A_i does not exceed $\varepsilon/(2\alpha)$, collectively for all sets A_i.



U
A_2
A_1
A_2
A_3
$\tilde{u}$
U_3
U_2
$\tilde{u}'$
U_1
I_1 I_2 I_3 I_4 I_5 I_6 I_7 I_8 I_9 I_{10} I_{11} I_{12}
$\mathbb{R}$

Fig. 6 Collection $\{U_i\}$ is a partition of U. Sets A_is are preimages of U_is under the map $\tilde{u}$. Intervals I_js approximate A_is from within. The function $\tilde{u}'$ is a piecewise constant approximation of $\tilde{u}$, defined on the collection $\{I_j\}$.

Finally, define u_i to be any element of U_d that is also in U_i (it is possible to find such u_i because U_i has nonempty interior and U_d is dense in U), and let u_0 to be any in U_d. Construct the approximation

$$\tilde{u}'(t) = \begin{cases} u_i & \text{if } \exists\, i \text{ and } \exists\, j \text{ such that } t \in I_j \subseteq A_i \\ u_0 & \text{otherwise} \end{cases} . \tag{23}$$

By construction, $\tilde{u}'$ is in Σ^*_{dt}. Now, compute the distance

$$\rho_{\mathscr{U}}(\tilde{u},\tilde{u}') \le {\sum_j}' \int_{I_j} \|\tilde{u}(t) - u_i\|\,\mathrm{d}t + \alpha\frac{\varepsilon}{2\alpha} \le \tau(\tilde{u})\frac{\varepsilon}{2\tau(\tilde{u})} + \frac{\varepsilon}{2} = \varepsilon\,. \tag{24}$$

Here, $\sum'$ denotes the summation over j such that $I_j \subseteq A_i$ for some i.

In conclusion, we note that for any $\tilde{u} \in \mathscr{U}$ and $\varepsilon > 0$ we found $\tilde{u}' \in \Sigma^*_{\mathrm{dt}}$ such that $\rho_{\mathscr{U}}(\tilde{u},\tilde{u}') \le \varepsilon$. Therefore, Σ^*_{dt} is dense in $\mathscr{U}$. □

Finally, we have established the main result of the paper:

Theorem 7. *Under exact and numerical computation models, there exists a set of motion primitives and a resolution complete planning algorithm.*

Proof. The result follows from Theorems 4, 5, and 6. □

Perhaps surprisingly, the algorithm may be resolution complete without actually causing convergence in dispersion. As discussed in Section 2, a dense sample sequence on a precompact set converges. However, $\mathscr{U}$ is generally not precompact. Therefore, our proposed algorithm does not converge in $\mathscr{U}$. Nevertheless, it converges in the trajectory space for the bounded time problem, assuming X_free is bounded:

Theorem 8 (Convergence of Finite Length Trajectories). *Assuming that execution time is bounded by T, the space $\mathscr{X}_{\mathrm{I},T} = \{\tilde{x} \in \mathscr{X}_\mathrm{I} \mid \tau(\tilde{x}) \le T\}$ is precompact. Hence, a dense sequence of controls corresponds to a dense sequence of trajectories that converges to the set of all feasible trajectories.*

Proof. For the set $\mathscr{X}_{\mathrm{I},T}$, the following two conditions hold:

1. **Uniform boundedness:** There exists $C_1 > 0$ such that $\|\tilde{x}(t)\| \le C_1$ for all $\tilde{x}$ in $\mathscr{X}_{\mathrm{I},T}$. The condition is easy to verify. Let $C_1 = MT + \|x_\mathrm{I}\|$ and consider

$$\|\tilde{x}(t)\| \le \|\tilde{x}(t) - \tilde{x}(0)\| + \|\tilde{x}(0)\| \le Mt + \|x_\mathrm{I}\| \le C_1 \; ; \tag{25}$$

2. **Uniform equicontinuity:** There exists $C_2 > 0$ such that for all $\tilde{x} \in \mathscr{X}_{\mathrm{I},T}$ and any t and t' in $[0\,,\,\tau(\tilde{x})]$, we have $\|\tilde{x}(t) - \tilde{x}(t')\| \le C_2|t - t'|$. The above follows directly from the definition of $\mathscr{X}$, by letting $C_2 = M$.

Using the Arzelà–Ascoli[4] theorem [20], it follows that $\mathscr{X}_{\mathrm{I},T}$ is precompact. The convergence of sampled trajectories follows from Lemma 1. □

6 Conclusions

In summary, we have considered the problem of existence of resolution complete algorithms for motion planning with differential constraints. To develop the set of sufficient conditions, the most important step was to map the space of controls $\mathscr{U}$ to the trajectory space $\mathscr{X}$ through the system dynamics. By forming metric spaces we have induced topologies on $\mathscr{U}$ and $\mathscr{X}$ and shown that the map is continuous, as illustrated in the diagram, which updates Fig. 1 with the precise terminology from the paper:

$$\begin{tikzcd} \Sigma \arrow[d] & U \arrow[d] & X \arrow[d] \\ \Sigma^* \arrow[r, "\subseteq"] & \mathscr{U} \arrow[r, "\tilde{x}(x_I,\tilde{u})"] & \mathscr{X}\,. \end{tikzcd} \tag{26}$$

We have relied on the fact that $\mathscr{U}$ can be sufficiently sampled using primitives Σ to obtain Σ^*, yielding a computational approach. Using this formulation, we have introduced a resolution-complete algorithm for motion planning with differential constraints in the most general setting known to date, requiring only Lipschitz continuity of the system.

We believe that metric space formulations and the resulting functional analysis framework may be practically useful for a broader class of systems and motion planning algorithms. For example, consider a system which is symmetric under Lie group transformations (e.g., [10]) and suppose the dispersion of a given set of motion primitives is limited by $\varepsilon > 0$ in the set of all trajectories that have the final time bounded by Δt. Consider further the set of all strings of length N composed of these primitives and the set of all trajectories with the final time bounded by $N\Delta t$. It follows from Lemma 5 that the dispersion of the former set in the later set is bounded by $K\varepsilon$, in which K depends only on N, Δt, and L_x; however, it is independent of any particular trajectory (we leave it to the reader to verify this fact). Hence, to reach the desired dispersion in the reachable set, it suffices to build primitives that approximate short-time trajectories with a given resolution. We also find our study

[4] In the provided reference the theorem is called Arzelà's theorem, and the synonymous term "relatively compact" is used instead of "precompact".

to be complementary to path pruning techniques (e.g., [9, 11]). Path pruning methods improve the path diversity to reduce the branching factor of search methods, whereas our analysis provides a bound on the dispersion of the set of pruned paths to guarantee convergence.

Acknowledgements. This work was supported in part by NSF grants 0904501 (IIS Robotics), NSF grant 0535007 (IIS Robotics) and 1035345 (Cyberphysical Systems), DARPA SToMP grant HR0011-05-1-0008, and MURI/ONR grant N00014-09-1-1052.

References

1. Laumond, J.-P., Sekhavat, S., Lamiraux, F.: Guidelines in nonholonomic motion planning for mobile robots. In: Laumond, J.-P. (ed.) Robot Motion Planning and Control, pp. 1–53. Springer, Heidelberg (1998)
2. Donald, B.R., Xavier, P.G., Canny, J., Reif, J.: Kinodynamic planning. Journal of the ACM 40, 1048–1066 (1993)
3. Bobrow, J.E., Dubowsky, S., Gibson, J.S.: Time-optimal control of robotic manipulators along specified paths. International Journal of Robotics Research 4(3), 3–17 (1985)
4. Hollerbach, J.: Dynamic scaling of manipulator trajectories, tech. rep., MIT A.I. Lab Memo 700 (1983)
5. Slotine, J.-J.E., Yang, H.S.: Improving the efficiency of time-optimal path-following algorithms. IEEE Transactions on Robotics & Automation 5(1), 118–124 (1989)
6. LaValle, S.M.: Planning Algorithms. Cambridge University Press, Cambridge (2006), `http://planning.cs.uiuc.edu/`
7. Reif, J.H.: Complexity of the mover's problem and generalizations. In: Proceedings IEEE Symposium on Foundations of Computer Science, pp. 421–427 (1979)
8. Latombe, J.-C.: Robot Motion Planning. Kluwer, Dordrecht (1991)
9. Branicky, M.S., Knepper, R.A., Kuffner, J.J.: Path and trajectory diversity: Theory and algorithms. In: Proceedings IEEE International Conference on Robotics & Automation, pp. 1359–1364 (2008)
10. Frazzoli, E., Dahleh, M.A., Feron, E.: Robust hybrid control for autonomous vehicles motion planning, Tech. Rep. LIDS-P-2468, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology (1999)
11. Erickson, L.H., LaValle, S.M.: Survivability: Measuring and ensuring path diversity. In: Proceedings IEEE International Conference on Robotics and Automation (2009)
12. Bhatia, A., Frazzoli, E.: Sampling-based resolution-complete algorithms for safety falsification of linear systems. In: Egerstedt, M., Mishra, B. (eds.) HSCC 2008. LNCS, vol. 4981, pp. 606–609. Springer, Heidelberg (2008)
13. Haghverdi, E., Tabuada, P., Pappas, G.J.: Bisimulation relations for dynamical, control, and hybrid systems. Theoretical Computer Science 342, 229–261 (2005)
14. Bertsekas, D.P.: Convergence in discretization procedures in dynamic programming. IEEE Transactions on Automatic Control 20, 415–419 (1975)
15. LaValle, S.M., Konkimalla, P.: Algorithms for computing numerical optimal feedback motion strategies. International Journal of Robotics Research 20, 729–752 (2001)
16. Choset, H., Lynch, K.M., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, L.E., Thrun, S.: Principles of Robot Motion: Theory, Algorithms, and Implementations. MIT Press, Cambridge (2005)

17. Niederreiter, H.: Random Number Generation and Quasi-Monte-Carlo Methods. Society for Industrial and Applied Mathematics, Philadelphia (1992)
18. Royden, H.: Real Analysis. The Macmillman Company, Collier-Macmillman Limited, London (1988)
19. Bardi, M., Capuzzo-Dolcetta, I.: Optimal Control and Viscosity Solutions of Hamilton-Jacobi-Bellman Equation. Birkhäuser, Basel (2008)
20. Kolmogorov, A.N., Fomin, S.V.: Introductory Real Analysis. Dover Publications, Inc., New York (1975)

Grasp Invariance

Alberto Rodriguez and Matthew T. Mason

Abstract. This paper introduces a principle to guide the design of finger form: invariance of contact geometry over some continuum of varying shape and/or pose of the grasped object in the plane. Specific applications of this principle include scale-invariant and pose-invariant grasps. Under specific conditions, the principle gives rise to spiral shaped fingers, including logarithmic spirals and straight lines as special cases. The paper presents a general technique to solve for finger form, given a continuum of shape or pose variation and a property to be held invariant. We apply the technique to derive scale-invariant and pose-invariant grasps for disks, and we also explore the principle's application to many common devices from jar wrenches to rock-climbing cams.

1 Introduction

What principles should guide the design of finger form? It depends on context—the specific application, the hand design philosophy, and in particular on the function assigned to the fingers. In this paper we explore the possible role of the fingers in adapting to variations in object shape and pose. One common design approach is to adapt to object shape and pose by control of several actuators per finger. But for simpler hands, with one actuator per finger, or even one actuator driving several fingers, the job of gracefully adapting to shape and pose variations may fall on the finger form. This work explores grasp invariance over shape and/or pose variation as a principle for finger form design.

Alberto Rodriguez
Carnegie Mellon University, Pittsburgh, USA
e-mail: albertor@cmu.edu

Matthew T. Mason
Carnegie Mellon University, Pittsburgh, USA
e-mail: matt.mason@ri.cmu.edu

We begin with a geometry problem:

Scale-invariant contact problem: Given O an object shape in the plane, p a point on the boundary of O, and c the location of the finger's revolute joint, find the finger shape that makes contact at p despite scaling of O.

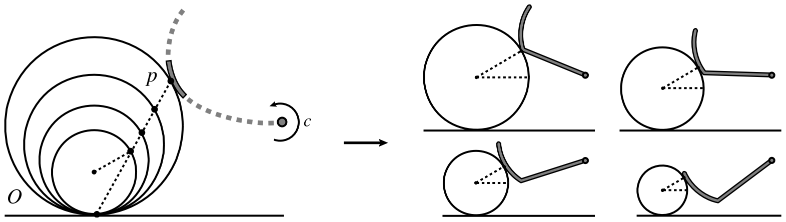


Fig. 1 The scale-invariant contact problem (*left*), solved in Sect. 3.2, yields a finger whose contact with the object O is preserved for different sizes of O (*right*)

If O is a smooth shape, a solution to the previous problem would preserve contact normal as well as contact location. As a consequence, many properties governing the mechanics of grasping and manipulation would also be preserved. For example, as we shall see later in this paper, if O is a disk of varying scale in the palm of a two-fingered hand, the solution of the scale-invariant contact problem yields identical equilibrium grasps despite the variation in scale.

This paper generalizes the scale-invariant contact problem, and explores its implications for finger design for grasping and fixturing problems. Scale-invariant contact is just one example of a broader class of problems where a continuum of constraints guides the design of finger shape. In the specific case of scale invariance, the set of constraints is generated by scaling the object.

In Sect. 3 we show how, under certain conditions, the problem can be mathematically formulated and admits a unique solution as an integral curve of a vector field. In Sect. 4 we find the analytical expression of the integral curve for the most simple and common cases. The integration yields spiral shaped fingers—in special cases, a logarithmic spiral, a shape long noted for its scale invariant properties [11]. Finally, in Sects. 5 and 6, we show that the principle applies to fixture and finger designs seen in many common devices from jar wrenches to rock-climbing cams.

2 Related Work on Finger Design

This paper grew out of our interest in simple hands, focused on enveloping grasps of objects with uncertain pose and shape [9]. The actual forms of the phalanges and the palm become important when the locations of contacts are not carefully planned and controlled. Simple hands adapt to varying shapes and poses by the emergent interaction of hand with object, rather than by actively driving the hand shape to conform to a known object shape and pose.

Several other approaches are evident in previous hand design literature. The enveloping grasps we employ are often contrasted with fingertip grasps, where the details of phalange form are not usually considered. The dichotomy between enveloping grasps and fingertip grasps corresponds roughly to the distinction between power and precision grasps in Cutkosky and Wright's grasp taxonomy [5].

While much of the analysis of grasp has focused on local stability and in-hand manipulation with fingertip grasps [3], hand design research has explored various grasp types more broadly. The hand design leading to the Barrett Hand [14, 1] explored a single finger probe, a pinch grasp, both cylindrical and spherical enveloping grasps, a two-fingered fingertip grasp, and a hook grasp. The design of the DLR-II likewise was guided by the desire to produce both power and precision grasps [4].

Unlike the work cited so far, our approach does not change the shape of the fingers to adapt to an object pose or shape. Instead, adaptation emerges from the interaction of fixed finger shapes with the object. In that respect the closest work is Trinkle and Paul's [13] work on enveloping grasps, Dollar and Howe's [7] work on hands with compliantly coupled joints, and Theobald et al.'s [10] design of a gripper for acquiring rocks. None of the above focus on the actual finger form. Dollar and Howe [6] survey 20 different designs of compliant and underactuated hands, and all employ cylindrical or straight fingers, occasionally employing some additional shape features, but with no particular principle described to guide their design.

The closest work to this paper is found not in robotics research, but in the design of various tools and hardware. All the devices in Fig. 2 adapt to shape and pose

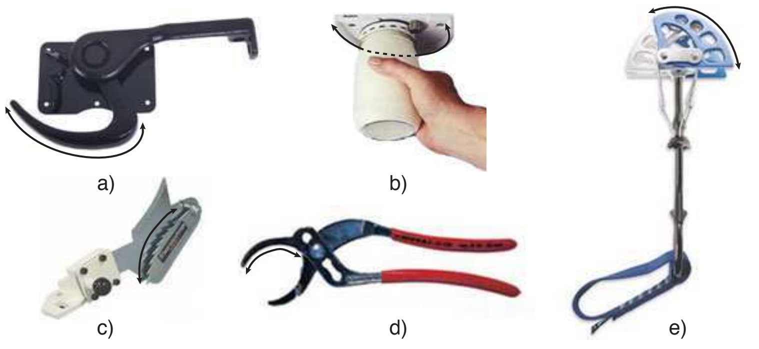


Fig. 2 Curved shapes used in: **a)** Truck door lock; **b)** Jar opener; **c)** Anti-kickback device for table saw; **d)** Pliers; **e)** Climbing cam

variations without changing their shape. Instead, the adaptation arises from the curved shape. In the cases of the rock-climbing cam [8] and the anti-kickback device [2] the shape was derived theoretically. The others, to our knowledge, are product of human intuition. In Sec. 6 we analyze their shape under the grasp invariance principle.

3 Problem Formulation

Let $H(x,s)$ be a parametrized transformation of O, i.e. for all values of the transformation parameter s, $H(O,s)$ is a warped version of the object. We are interested in designing a finger that makes first contact with O at a given point p, and with points $p_s = H(p,s)$ for all warped versions of O. Locally, making first contact is equivalent to the object and finger being tangent. Under this formulation, the scale-invariant contact problem becomes:

> *Transformation-invariant contact problem*: Find the shape of the finger that is always tangent to the set of objects $\{H(O,s)\}_s$ when contacting them at points $\{p_s\}_s$.

Let *contact curve* l describe the travel of the point p, and let *contact vector* v be the tangent to the object at point p, Fig. 3.

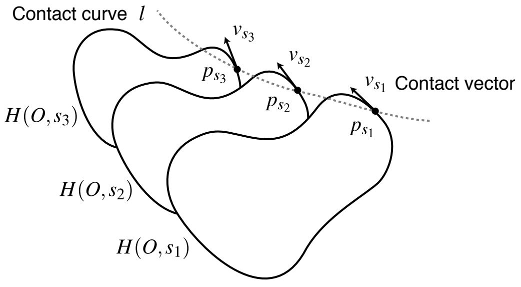


Fig. 3 The motion of the object O defines the travel of the point p and with it, the contact curve l and the motion of the contact vector v

Every point l_s in the contact curve corresponds to an instance of the object, $H(O,s)$. As a consequence it induces a constraint at one point in the finger. This establishes a relationship between points in the contact curve and points in the finger: Whenever the finger crosses the contact curve, i.e. contacts the object at p_s, it must always do it tangent to the corresponding contact vector v_s, i.e. tangent to the object at p_s. In the next section we see that this relation leads to a reformulation of the problem in terms of vector fields.

3.1 Geometric Reformulation of the Problem

The relationship between the finger and the contact curve described in the last section allows us to reformulate the transformation-invariant contact problem as:

> *Transformation-invariant contact problem:* Find the finger form that always crosses the contact curve l tangent to the contact vector v.

Every time the finger crosses the contact curve, its tangent must satisfy one constraint. Let r be the distance between the center of rotation, c, and that contact point. As shown in Fig. 4a, while the finger rotates around c, the same tangential constraint is propagated along an arc of radius r and center c. By repeating the same procedure

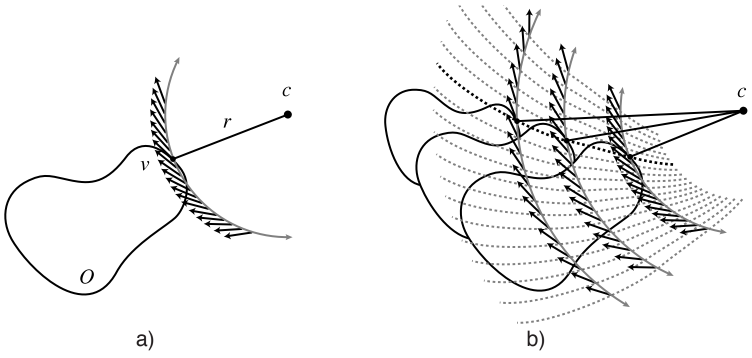


Fig. 4 a) Parallel transport of v radially from c. **b)** Extending the same construction to all values of s, gives us the vector field

for all constraints, or equivalently for all points in the contact curve, we obtain a vector field V defined on an annulus in the plane.

The vector field V is well defined if and only if the construction does not impose inconsistent constraints, that is, if and only if it does not impose two different constraints at the same point. To avoid inconsistent constraints, the distance between c and the contact curve must be strictly monotonic, because different points along the contact curve will impose constraints on different concentric arcs. In practice, given a parametrized transformation $H(O,s)$, we avoid inconsistency of constraints by restricting to an interval of s where such distance is strictly monotonic. We will refer to a problem that avoids inconsistent constraints as a *proper problem*.

By construction, any integral curve of the vector field V of a proper problem must satisfy all contact curve constraints. Hence, if we shape the finger following the integral curve, it shall contact the object with the expected geometry, and therefore must be a solution to the transformation-invariant contact problem.

By virtue of the the theorem of existence and uniqueness of maximal integral curves, Theorem 1, the existence of the integral curve depends on the smoothness of the vector field V. If the contact curve l is smooth and v changes smoothly along it, the vector field V will also be smooth because V is the parallel transport of v along concentric arcs centered at c, which is a continuous and differentiable operation. If that is the case, the solution is guaranteed to exist for the restricted interval of s where the problem is a proper problem.

Theorem 1 (Existence and Uniqueness of Maximal Integral Curves). *Let X be a smooth vector field on an open set $U \in \mathbb{R}^{n+1}$ and let $p \in U$. Then there exists a unique and maximal integral curve $\alpha(t)$ of X such that $\alpha(0) = p$.*

The theorem is a direct consequence of the fundamental existence and uniqueness theorem for solutions of systems of differential equations [12].

3.2 General Solution Recipe

The derivation of the previous section suggests a general procedure for obtaining the shape of the finger:

1. Given the parametrized transformation of the object, $H(O,s)$, construct the contact curve l and the set of contact vector constraints v.
2. Obtain the vector field V, by rotating l and v around the rotation center.
3. Find the shape of the finger by integrating V.

Figure 5 shows an example of the procedure applied to the scale-invariant contact problem for a disk-shaped object resting in a planar palm. When scaling the disk, the contact curve l becomes a line and the contact vector v is constant along l.

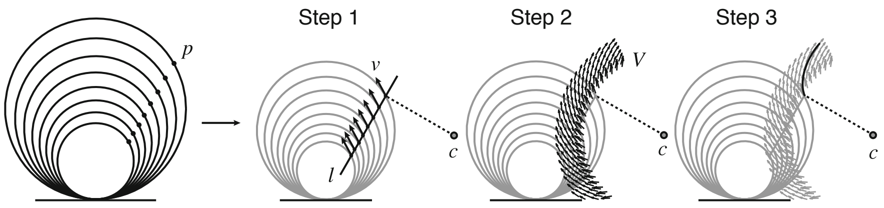


Fig. 5 Step by step execution of the general recipe to obtain the form of the finger for the scale-invariant contact problem. **Step 1** Construction of the contact curve l and contact vectors v. **Step 2** Vector field V. **Step 3** Integral curve of V

4 Analytic Solution

In the general case we can find the shape of the finger using the recipe in Sect. 3.2 and numerically integrating the vector field. In some specific cases, namely when the contact curve l is a line and the contact vector v is constant along it, we can also integrate analytically the vector field. This is the case of linear scaling of the object, translation of the object, or any linear combination of scaling and translation.

Let (x,y) and (r,θ) be the cartesian and polar coordinates in the plane. The shape of the finger is the solution to the system of first order differential equations:

$$\begin{aligned} \dot{x} &= V_x \\ \dot{y} &= V_y \end{aligned} \tag{1}$$

where $V = (V_x, V_y)$ is the vector field obtained in Sect. 3.2. The identity (2) relates the derivatives of the cartesian coordinates to the derivatives of the polar coordinates:

$$\frac{\mathrm{d}y}{\mathrm{d}x} = \frac{r'(\theta)\sin\theta + r(\theta)\cos\theta}{r'(\theta)\cos\theta - r(\theta)\sin\theta} \tag{2}$$

With it, we can rewrite the cartesian system (1) as the single polar differential equation (3) that we will solve in the next subsections.

$$\frac{V_y}{V_x} = \frac{r'(\theta)\sin\theta + r(\theta)\cos\theta}{r'(\theta)\cos\theta - r(\theta)\sin\theta} \tag{3}$$

Without loss of generality we suppose the contact curve l to be parallel to the Y axis and the center of rotation c to lie on the X axis. Let α_0 be the constant angle between l and the contact vector v. Depending on the relative location of c and l we distinguish three cases: (I) c lies on l; (II) c is at finite distance from l; and (III) c is at infinity, i.e. the finger translates rather than rotates.

4.1 Case I: Rotation Center on the Contact Curve

We assume l to be the Y axis and c to be located at the origin as in Fig. 6.

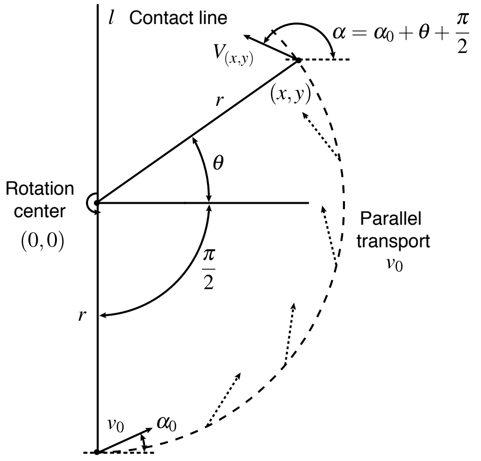


Fig. 6 Parallel transport of v_0 along the arc with center at c gives us $V_{(x,y)}$, the tangent vector to the finger at (x,y)

In this case, the differential equation (3) becomes:

$$\frac{V_y}{V_x} = \tan\left(\alpha_0 + \theta + \frac{\pi}{2}\right) = \frac{r'(\theta)\sin\theta + r(\theta)\cos\theta}{r'(\theta)\cos\theta - r(\theta)\sin\theta} \tag{4}$$

Solving for $r'(\theta)$:

$$r'(\theta) = -r(\theta)\tan\alpha_0 \tag{5}$$

and integrating, we obtain:

$$r(\theta) = \mathrm{C}\, e^{-\theta\tan\alpha_0} \tag{6}$$

where C is the integration constant. Solution (6) is the equation of a logarithmic spiral with pitch $\frac{\pi}{2} + \tan^{-1}(\cot\alpha_0)$. The solution could have been anticipated from the diagram on Fig. 6 because the angle between the radial line and the vector tangent to the curve is constant. This is characteristic of logarithmic spirals and gives them their scale invariant properties [11], as we will further see in Sect. 6.

4.2 Case II: Rotation Center at Finite Distance from Contact Curve

Without loss of generality, let c be the origin and let l be parallel to the Y axis, crossing the X axis at $(1,0)$, as in Fig. 7.

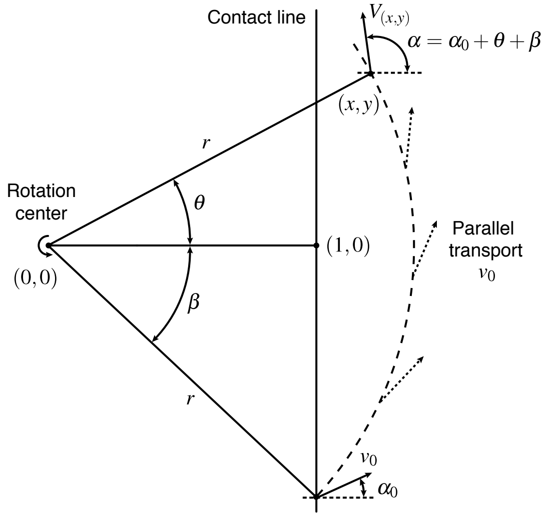


Fig. 7 Same construction as in Fig. 6. Parallel transport of v_0 along the arc with center at c gives $V_{(x,y)}$, the tangent vector to the finger at (x,y)

If we apply (3) to the construction of Fig. 7 we obtain:

$$\frac{V_y}{V_x} = \tan(\alpha_0 + \theta + \beta) = \tan\left[\alpha_0 + \theta + \cos^{-1}\left(\frac{1}{r}\right)\right] = \frac{r'(\theta)\sin\theta + r(\theta)\cos\theta}{r'(\theta)\cos\theta - r(\theta)\sin\theta} \tag{7}$$

With some trigonometric algebra, (7) can be solved for $r'(\theta)$ as:

$$r'(\theta) = r(\theta) \cdot \frac{\cos\alpha_0 - \sqrt{r(\theta)^2 - 1}\sin\alpha_0}{\sin\alpha_0 + \sqrt{r(\theta)^2 - 1}\cos\alpha_0} \tag{8}$$

Equation (8) is a separable differential equation with form $\frac{\mathrm{d}r}{\mathrm{d}\theta} = g(r)$ and can be solved as $\int \mathrm{d}\theta = \int \frac{1}{g(r)}\mathrm{d}r$:

$$\theta = \int \mathrm{d}\theta = \int \frac{1}{r(\theta)} \cdot \frac{\sin\alpha_0 + \sqrt{r(\theta)^2 - 1}\cos\alpha_0}{\cos\alpha_0 - \sqrt{r(\theta)^2 - 1}\sin\alpha_0}\mathrm{d}r = \left[\begin{array}{c}\text{change}\\ t \to \sqrt{r(\theta)^2 - 1}\end{array}\right]$$

$$
\begin{aligned}
&= \int \frac{t}{t^2+1} \cdot \frac{(\sin\alpha_0 + t\cos\alpha_0)}{(\cos\alpha_0 - t\sin\alpha_0)} \mathrm{d}t = \\
&= -\int \frac{1}{t^2+1} + \frac{\cos\alpha_0}{\sin\alpha_0 t - \cos\alpha_0} \mathrm{d}t = \\
&= -\tan^{-1}(t) - \frac{\ln(|\sin\alpha_0 t - \cos\alpha_0|)}{\tan\alpha_0} + \mathrm{C} = \\
&= -\tan^{-1}\left(\sqrt{r^2-1}\right) - \frac{\ln\left(|\sin\alpha_0\sqrt{r^2-1} - \cos\alpha_0|\right)}{\tan\alpha_0} + \mathrm{C} \qquad (9)
\end{aligned}
$$

where C is the integration constant determined by the initial conditions. As shown in Fig. 8, the solution is a family of spirals valid for all α_0, except when $\tan\alpha_0 = 0$. In that specific case, a similar derivation yields a different spiral:

$$
\theta(r) = -\tan^{-1}\left(\sqrt{r^2-1}\right) + \sqrt{r^2-1} + \mathrm{C} \qquad (10)
$$

4.3 Case III: Rotation Center at Infinity

A center of rotation at infinity corresponds to the case where the finger does not rotate but translates in a particular direction v_c, i.e. the finger joint is prismatic rather than revolute. The vector field V is defined, then, by the translation of the contact line in the same direction. As we are limiting the analysis to the case where the contact vector is constant along the contact line, the vector field V is constant on the plane. The integral curve of V, and shape of the finger, is consequently a line segment aligned with the constant contact vector v_0.

The vector field is well defined except in the case when the translation of the finger v_c is aligned with the contact line. In this degenerate situation, the vector field is only defined on top of the contact line, and the solution is still a line.

4.4 Family of Solutions

In cases I and II, the integral curve of the vector field is a spiral. In case III, we always obtain a line segment. Figure 8 shows a summary of the solutions we obtain when changing α_0. Note that the vector fields obtained with α_0 and $\alpha_0 + \pi$ always have the same magnitude and opposite direction, hence it suffices to analyze the range $\left[\frac{\pi}{2}, -\frac{\pi}{2}\right]$. In Fig. 8 we show a long section of the spiral. However, for practical reasons, the actual shape of the finger is just the initial portion.

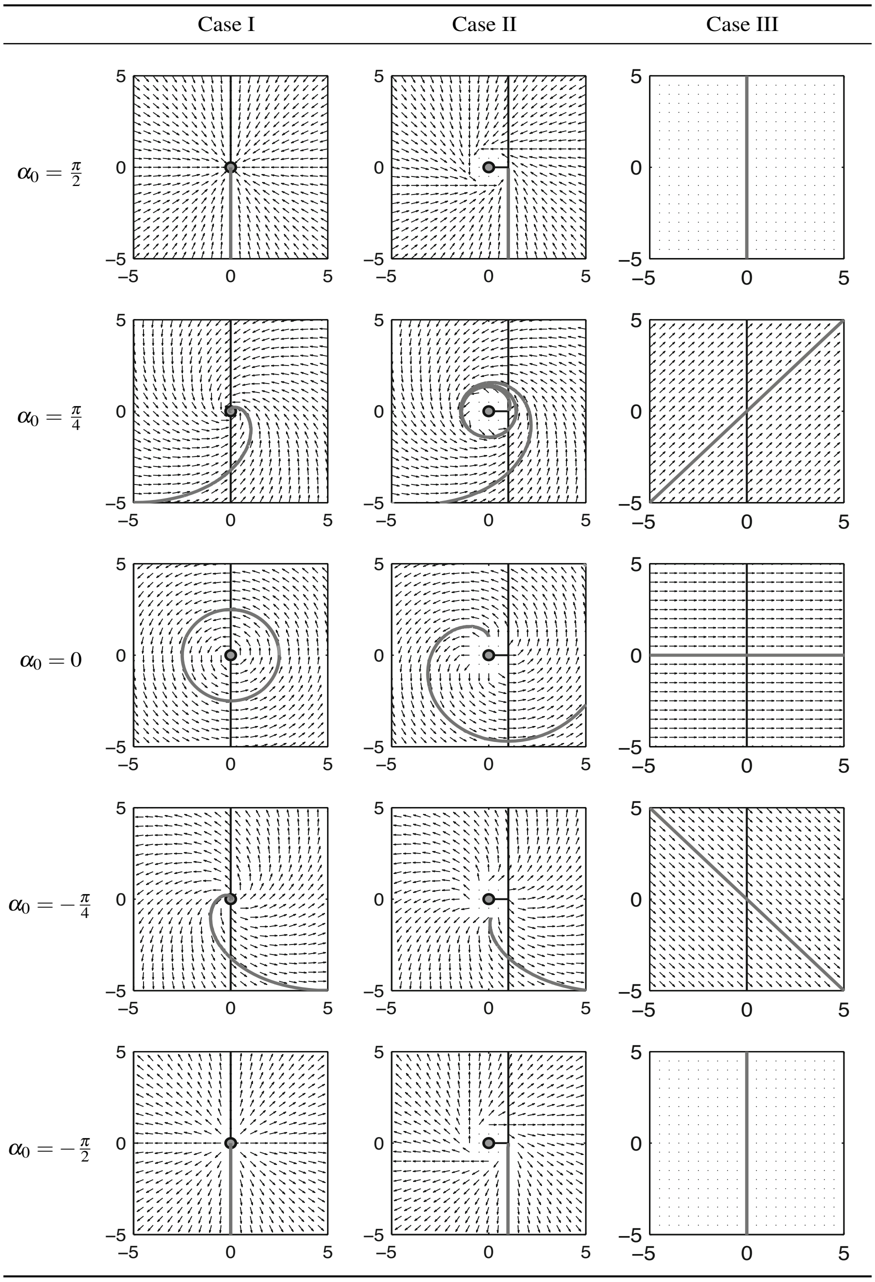


Fig. 8 Plots show the contact curve (*vertical bold line*), the rotation center (*grey dot*), and the finger solution (*grey curves*) for different values of α_0. In all cases, the finger and the contact curve cross at the lower half of the contact curve with constant angle, equal to the corresponding α_0. **Case I** Rotation center lying on the contact curve. **Case II** Rotation center at finite distance from the contact curve. **Case III** Rotation center at infinity

5 Applications to Grasping

5.1 Scale Invariant Grasping

The solution to the scale-invariant contact problem in Fig. 1 shows a finger that contacts a disk of varying size with constant geometry. This same idea can be used to design a gripper whose equilibrium grasps are geometrically invariant with scale. Suppose we aim for a triangular grasp between two fingers and a palm. Fig. 9 shows the corresponding contact line l, contact vectors v, and the induced normalized diagram. The solution belongs to the family of spirals obtained in Case II in Sect. 4.2, where the center of rotation c is at finite distance from the contact line l.

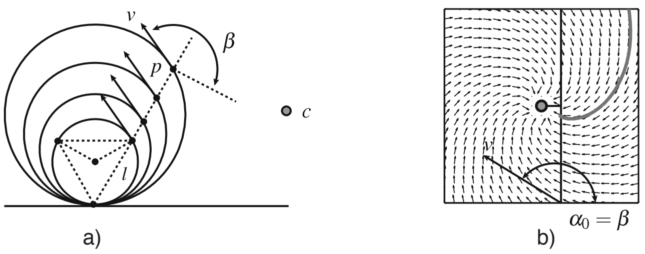


Fig. 9 a) Contact curve, l, and contact vector, v, of the scale invariant grasping application. **b)** Corresponding normalized diagram with $\alpha_0 = \beta$

The contact line, the contact vector, and the form of the finger, depend on the object O and the type of contact we aim for. Therefore, the form of the finger depends on the task to solve. For the specific example of task on Fig. 9, we can construct a scale-invariant gripper by combining two identical but symmetric fingers as in Fig. 10.

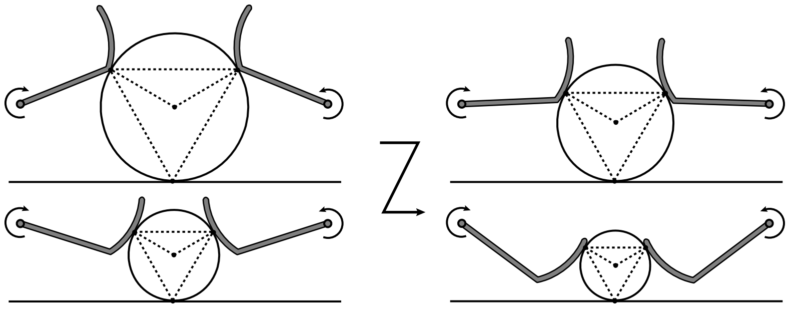


Fig. 10 Gripper with scale invariant fingers for grasping spheres with a regular triangular grasp

5.2 Pose Invariant Grasping

Here we aim to design a hand whose grasps are invariant with respect to the location of a given object rather than scale. Suppose again that we want to grasp a disk of a given size with a triangular grasp, against a planar palm. Now, the disk can be located anywhere along the palm, and we want the grasp geometry to be invariant with respect to that displacement. Figure 11 shows the corresponding contact line l, contact vectors v, and normalized diagram. The solution belongs to the family of spirals in Case II in Sect. 4.2.

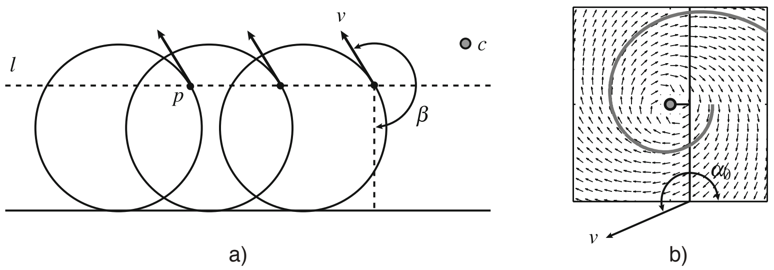


Fig. 11 a) Contact curve, l, and contact vector, v, of the pose invariant grasping application. **b)** Corresponding normalized diagram with $\alpha_0 = \beta$

Again, the contact line, the contact vector and consequently the form of the finger depend on the object O and the type of contact desired. The diagram corresponds to case II in Sect. 4.2 with the rotation center at a finite distance from the contact line. In Fig. 12 two identical but symmetric fingers compose a pose-invariant gripper.

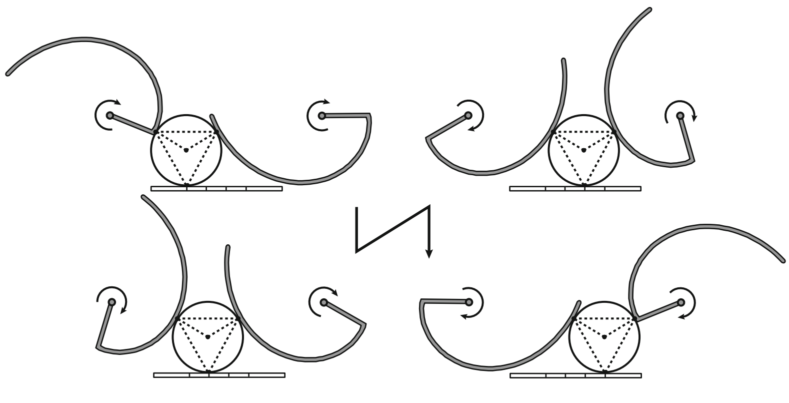


Fig. 12 Invariant grasping geometry for different locations of a sphere

5.3 Pick-Up Tool

Suppose we are to design a gripper with two rigid fingers to pick up an object from the ground. The object needs to slide along the length of the fingers while it is being lifted, similar to Trinkle and Paul's work on dexterous manipulation with sliding contacts [13]. Because of the critical role that contact geometry plays in the sliding motion, complex lift plans can be simplified if the contact geometry between finger and object were to be invariant with respect to the lifting motion. With that in mind, we can use the grasp invariance principle to find the finger shape that preserves a contact suitable for sliding. Figure 13 shows a gripper designed to pick up disks.

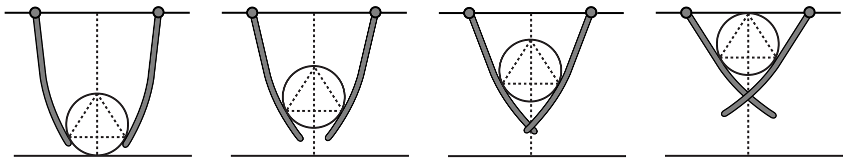


Fig. 13 Invariant contact geometry of the pick-up tool when lifting an object from the ground

6 Applications to Common Devices

6.1 Rock-Climbing Cam

A spring loaded cam is a safety device used in rock climbing to secure anchor points in cracks in the rock face. The device uses the mechanical advantage of the wedge effect to convert pulling force into huge friction forces, Fig. 14.

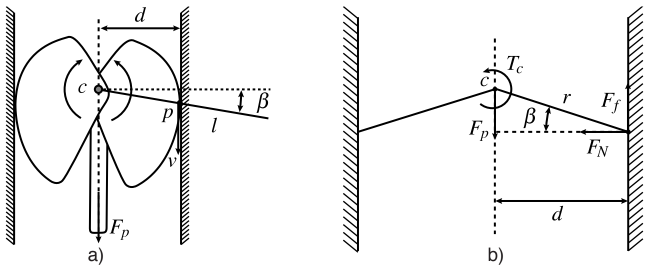


Fig. 14 a) Spring loaded cam as used in rock-climbing. **b)** Static force diagram

The diagram on Fig. 14b reveals an important relationship between the coefficient of friction μ and the cam angle β. In static equilibrium (zero torque at c):

$$\begin{aligned} & r \cdot F_f \cos\beta = r \cdot F_N \sin\beta \\ [F_f \le \mu F_N] \quad & F_N \tan\beta \le \mu F_N \\ & \beta \le \tan^{-1}\mu \qquad (11) \end{aligned}$$

where r is the distance from the cam rotation center c to the contact point, F_N is the normal force caused as a reaction to the pulling force F_p, and F_f is the frictional force. To design a cam whose loading pattern is invariant with the size of the crack, d, we can integrate the constraint (11) so that β remains constant despite variations in d. The rotation center c lies on top of the contact line l and hence the solution is a logarithmic spiral with $\alpha_0 = \beta$ as shown in Fig. 15, example of Case I in Sect. 4.1.

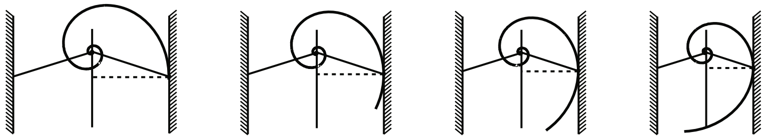


Fig. 15 Invariant loading pattern for different wall openings when using a logarithmic spiral shape

The logarithmic spiral has been used for decades in the design of climbing cams. The invention of modern rock-climbing cams is attributed to Raymond Jardine [8]. His invention used a logarithmic spiral, with camming angle $\beta = 13.5°$.

6.2 Anti Kickback Device for Table Saws

Table saw kickback happens when the blade catches the workpiece and violently throws it back to the front of the saw, towards the operator. An anti kickback device is a passive device that only allows forward motion of the workpiece. Among several options to introduce the asymmetry, one option is to use a spiral like rotating part that wedges a workpiece trying to move backwards, Fig. 16.

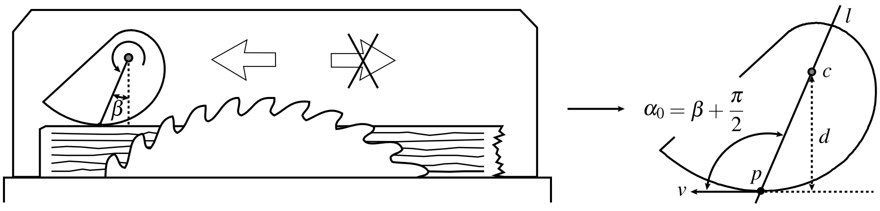


Fig. 16 Anti kickback device as used in a wood table saw. The contact between the retaining part and the workpiece is at an optimum angle β

Experiments have determined that the optimal contact between the workpiece and the anti kickback device is reached when $\beta = 8°$ [2]. To make the contact invariant with the thickness of the workpiece, we can use the grasp invariance principle to design the shape of the device. The center of rotation lies on top of the contact line l and consequently the optimal solution is a logarithmic spiral, with $\alpha_0 = \beta + \frac{\pi}{2}$. The patent of the device [2] proposed a logarithmic spiral as the optimal contour.

6.3 Jar Wrench

The jar wrench in Fig. 2b can open jars of varying sizes. Figure 17 shows the basic operating principle. The lid contacts the inner disk and the outer contour at different places for different sized lids. The mechanics of the opening procedure, and specifically the amount of friction available, depend of the value of the angle β.

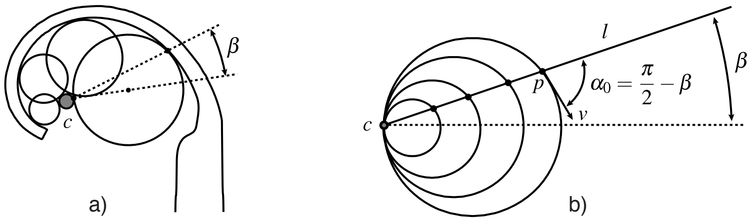


Fig. 17 a) Jar wrench. **b)** Simplified diagram where the discs are fixed and the wrench rotates

If we know the optimal value of β, and want the mechanics to be invariant across the range of jar lid sizes, we can use the grasp invariance principle to design the contour. To simplify the analysis we suppose, as shown in Fig. 17b, that the lids are fixed and resting always against the same contact point, while the wrench rotates to contact the lid. The center of rotation is on top of the contact line l. Consequently the optimal shape of the jar wrench is a logarithmic spiral, with $\alpha_0 = \frac{\pi}{2} - \beta$.

7 Discussion

In this paper we have introduced the grasp invariance principle and have given a recipe to apply it to design the form of rotating fingers and fixtures, given a continuum of shape or pose variation and a property to be held invariant. We have shown that, in simple cases, the finger has a spiral form, and have applied the technique to design scale-invariant and pose-invariant grippers for disks and a pick-up tool. Finally we have analyzed the shape of few common devices under the principle.

There are lots of extensions, generalizations and applications that we lacked either the space to discuss or the time to explore. Here we mention a few of them:

- Open design questions: Where should we put the center of rotation c? How do we choose the transformation $H(x,s)$ function? The finger shape depends both on c and $H(x,s)$. Those parameters can be chosen to satisfy additional properties.
- This paper covers design issues of rotating and prismatic fingers. There are other types of finger motion worth considering, for example those involving linkages.
- 2D designs are readily adapted to design grasp invariant 3D grippers, for example by arranging three fingers symmetrically around a circular palm. However we can also think about a deeper 3D generalization where fingers become 2D surfaces.
- What happens when we deal with non-smooth boundaries? Can we extend the approach to include contact at vertices?

- The grasp invariance principle applies to one-dimensional variations of shape or pose of the object, e.g. scale-invariant or pose-invariant grippers. How can we trade off various objectives to address generalization across objects and tasks?

Acknowledgements. This work was supported by NSF, ARL and Caja Madrid. This work is sponsored in part by the U.S. Army Research Laboratory, under the Robotics Collaborative Technology Alliance. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies of the Army Research Laboratory or the U.S. Government. Alberto Rodriguez is the recipient of a fellowship sponsored by Caja Madrid.

References

1. Barrett Technologies: The Barrett hand (2001), `http://www.barrett.com/robot/products-hand.htm`
2. Berkeley, J.: Anti-Kickback System. US Patent 4,615,247 (1986)
3. Bicchi, A., Kumar, V.: Robotic grasping and contact: A review. In: International Conference on Robotics and Automation, pp. 348–353. IEEE, Los Alamitos (2000)
4. Butterfass, J., Grebenstein, M., Liu, H., Hirzinger, G.: DLR-Hand II: next generation of a dextrous robot hand. In: International Conference on Robotics and Automation, pp. 109–114. IEEE, Los Alamitos (2001)
5. Cutkosky, M.: On grasp choice, grasp models, and the design of hands for manufacturing tasks. Transactions on Robotics and Automation 5(3), 269–279 (1989)
6. Dollar, A., Howe, R.: Joint coupling design of underactuated grippers. In: 30th Annual Mechanisms and Robotics Conference, pp. 903–911. Harvard University (2006)
7. Dollar, A., Howe, R.: The SDM Hand: A Highly Adaptive Compliant Grasper for Unstructured Environments. International Journal of Robotics Research, 1–11 (2010)
8. Jardine, R.: Climbing Aids. US Patent 4,184,657 (1980)
9. Mason, M., Srinivasa, S., Vazquez, A., Rodriguez, A.: Generality and simple hands. International Journal of Robotics Research In review (2010)
10. Theobald, D., Hong, W., Madhani, A., Hoffman, B., Niemeyer, G., Cadapan, L., Slotine, J., Salisbury, J.: Autonomous rock acquisition. In: Forum on Advanced Developments in Space Robotics, AIAA (1996)
11. Thompson, D.W.: On Growth and Form. Cambridge University Press, Cambridge (1961)
12. Thorpe, J.: Elementary Topics in Differential Geometry, pp. 8–10. Springer, Heidelberg (1979)
13. Trinkle, J., Paul, R.: Planning for dexterous manipulation with sliding contacts. The International Journal of Robotics Research 9(3), 24–48 (1990)
14. Ulrich, N., Paul, R., Bajcsy, R.: A medium-complexity compliant end effector. In: International Conference on Robotics and Automation, pp. 434–436. IEEE, Los Alamitos (1988)

Path Planning on Manifolds Using Randomized Higher-Dimensional Continuation

Josep M. Porta and Léonard Jaillet

Abstract. Despite the significant advances in path planning methods, problems involving highly constrained spaces are still challenging. In particular, in many situations the configuration space is a non-parametrizable variety implicitly defined by constraints, which complicates the successful generalization of sampling-based path planners. In this paper, we present a new path planning algorithm specially tailored for highly constrained systems. It builds on recently developed tools for Higher-dimensional Continuation, which provide numerical procedures to describe an implicitly defined variety using a set of local charts. We propose to extend these methods to obtain an efficient path planner on varieties, handling highly constrained problems. The advantage of this planner comes from that it directly operates into the configuration space and not into the higher-dimensional ambient space, as most of the existing methods do.

1 Introduction

Many problems require to determine a path between two points, fulfilling a given set of constraints. In Robotics, this appears for instance in parallel manipulators [35], robot grasping [25], constraint-based object positioning [24], surgery robots [1], and humanoid robots [20]. This situation also appears in Biochemistry when searching for low energy paths between different molecular conformations [38]. In all these cases, the constraints expressed as a set of equations reduce the configuration space to a variety composed by one or more manifolds embedded in a higher-dimensional ambient space, defined by the variables involved in the equations. Approaches that try to directly describe these manifolds exist, but they are either too complex to be applied in practice [6], or limited to particular architectures [28]. The adaptation of sampling-based planning methods is also cumbersome since, sampling in the ambient space, the probability of the samples to lay on the configuration space is null.

Josep M. Porta · Léonard Jaillet
Institut de Robòtica i Informàtica Industrial, CSIC-UPC
e-mail: {porta,ljaillet}@iri.upc.edu

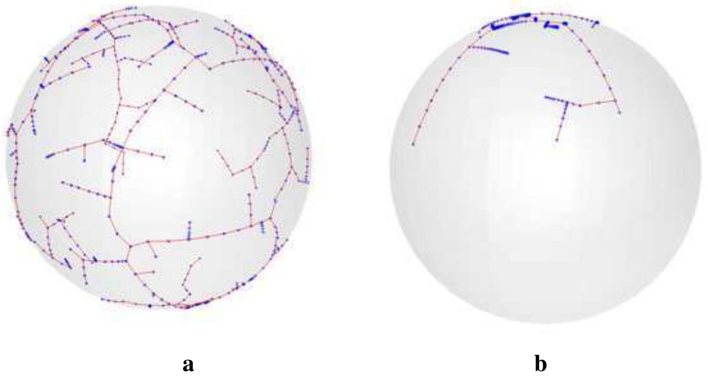


Fig. 1 RRTs with 500 samples. Blue crosses represent the tree nodes and red lines the connections between them. **a** When the ambient space is a box tightly enveloping the sphere, the exploration is relatively homogeneous. **b** When the box is elongated along the vertical axis, an unwanted bias penalize the exploration.

Consequently, several methods have been devised to find points of the configuration space from points of the ambient space.

The Kinematics-based Roadmap method [12] samples a subset of variables and uses inverse kinematics to find all the possible values for the remaining ones. This strategy is only valid for particular families of mechanisms, and although some improvements have been proposed [8], the probability of generating invalid samples is significant. Moreover, the presence of singularities in the subset of variables solved via inverse kinematics complicates the approach [11].

An alternative strategy to get a valid configuration is to use numerical iterative techniques, either implementing random walks [39], or the more efficient Jacobian pseudo inverse method [2, 9, 30]. All these approaches only perform properly when the ambient and the configuration spaces are similar. If the constraints define one or several complex surfaces with many folds, a uniform distribution of samples in the ambient space will not translate to a uniform distribution in the configuration space and this heavily reduces the efficiency of the sampling approaches. This problem may appear even in simple cases such as the one described in Fig. 1, where a Rapidly-exploring Random Tree (RRT) is built on a sphere from points sampled in a 3D ambient space. If the sphere is not centered in, and tightly enveloped by the ambient space, the sampling process is biased and the result is a poor exploration of the solution variety. The lack of prior knowledge on the variety structure makes it hard to forecast whether or not a sampling-based approach would be successful.

One way to limit the problems of mismatching between the two spaces is to focus the sampling on a subset of the ambient space around the configuration space [42]. However, even in the case where the ambient and the configuration spaces are somehow similar, samples are thrown in the ambient space that can be of much higher dimensionality than the configuration space. Um *et al* [36] sketch a RRT scheme

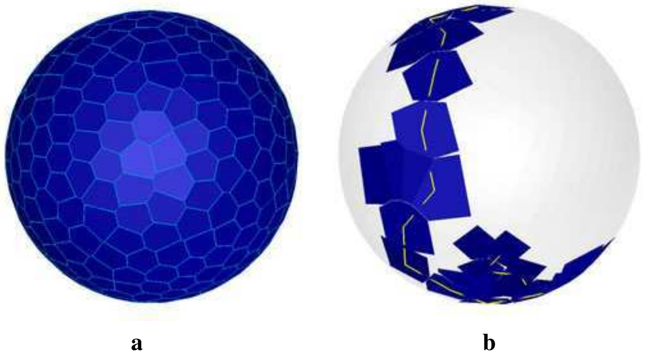


Fig. 2 Atlas of the sphere obtained by Higher-dimensional Continuation. Each polytope is a chart that locally parametrizes the sphere. **a** The full atlas includes about 500 charts. **b** The part of the atlas explored with our approach for connecting the two poles. Only about 30 charts are generated. The solution path is shown as a yellow line.

where the tree is defined in the tangent of the configuration space, which is of the same dimensionality as the variety. However, the overlap between tangent spaces at different points can lead to an inappropriate sampling bias and points in the tangent space do not actually fulfill the equations defining the variety. Ideally, one would like to sample directly on the configuration space. A uniform sampling over this space typically relies on a global parametrization. In some families of mechanism distance-based formulations provide this parametrization [13, 31], some approaches try to infer it from large sets of samples [14], and task-space planners assume that a subset of variables related with the end-effector are enough to parametrize the configuration space [40, 27]. However, it is in general not possible to obtain a global isometric parametrization of the configuration space.

From differential geometry, it is well known that a variety can be described by an atlas containing a collection of charts, each chart providing a local parametrization of the variety [22]. Higher-dimensional Continuation techniques (see [17] for a survey) provide principled numerical tools to compute the atlas of one of the connected components of an implicitly defined variety, departing from a point and avoiding overlap between neighboring charts. For instance, Fig. 2a shows the atlas obtained with the most recent of these techniques [15] in the toy problem of the sphere. One-dimensional continuation methods (also known as path following, homotopy or bootstrap methods), have been strongly developed in the context of Dynamical Systems [19], whereas in Robotics, they have been mainly used for solving problems related to Kinematics [26, 29]. To our knowledge, Higher-dimensional Continuation tools have not been used in Robotics.

In this paper, we extend the tools developed for Higher-dimensional Continuation to the context of path planning. We define the concept of partial atlas connecting two configurations, dealing with the presence of obstacles. We also introduce the random exploration of a variety focused towards a target configuration. As a result, we obtain a Higher-dimensional Continuation planner (HC-planner) for highly constrained systems that clearly outperforms existing approaches. Figure 2b shows an example of path found with our approach for the sphere toy problem. Note how only a small subset of all the atlas charts is needed to find a path connecting the two query points.

Next section provides a description of the tools for High-dimensional Continuation. Section 3 proposes an extension of these tools to the context of path planning. Section 4 compares the performance of the planner with respect to existing methods for several benchmarks. Finally, Section 5 summarizes the contributions of this work and indicates points that deserve further attention.

2 Higher-Dimensional Continuation

Next, we describe the main algorithmic tools appearing in [15]. By generalizing the one-dimensional pseudo-arclenght procedure, these tools allow the generation of an atlas for describing a k-dimensional smooth variety implicitly defined by a system of equations

$$\mathbf{F}(\mathbf{x}) = \mathbf{0}\,, \tag{1}$$

with $\mathbf{F} : \mathbb{R}^n \to \mathbb{R}^{n-k}$, $n > k > 0$. Figure 3 illustrates the main idea on which relies the approach. Given a point $\mathbf{x}_i$ on the variety, we can define $\mathbf{\Phi}_i$, an orthonormal basis of the tangent space of the variety at this point. This is the $n \times k$ matrix satisfying

$$\begin{pmatrix} \mathbf{J}(\mathbf{x}_i) \\ \mathbf{\Phi}_i^\top \end{pmatrix} \mathbf{\Phi}_i = \begin{pmatrix} \mathbf{0} \\ \mathbf{I} \end{pmatrix}\,, \tag{2}$$

with $\mathbf{J}(\mathbf{x}_i)$ the Jacobian of $\mathbf{F}$ evaluated at $\mathbf{x}_i$ and $\mathbf{I}$ the identity matrix. The pair $(\mathbf{x}_i, \mathbf{\Phi}_i)$ defines a chart, $\mathcal{C}_i$, that locally approximates the variety. To extend the atlas, the root point, $\mathbf{x}_j$, of a new chart, $\mathcal{C}_j$, can be obtained by first generating a point, $\hat{\mathbf{x}}_j$, using the tangent space of $\mathcal{C}_i$

$$\hat{\mathbf{x}}_j = \mathbf{x}_i + \mathbf{\Phi}_i\,\mathbf{u}_i^j\,, \tag{3}$$

with $\mathbf{u}_i^j$ a k-dimensional vector of parameters. Then $\mathbf{x}_j$ is the orthogonal projection of $\hat{\mathbf{x}}_j$ into the variety. This projection is obtained by solving the system [23]

$$\begin{aligned} \mathbf{F}(\mathbf{x}_j) &= \mathbf{0}\,, \\ \mathbf{\Phi}^\top(\mathbf{x}_j - \hat{\mathbf{x}}_j) &= \mathbf{0}\,, \end{aligned} \tag{4}$$

using a Newton procedure where $\mathbf{x}_j$ is initialized to $\hat{\mathbf{x}}_j$ and where at each iteration $\mathbf{x}_j$ is updated with the increment $\Delta\mathbf{x}_j$ fulfilling

$$\begin{pmatrix} \mathbf{J}(\mathbf{x}_i) \\ \mathbf{\Phi}_i^\top \end{pmatrix} \Delta\mathbf{x}_j = -\begin{pmatrix} \mathbf{F}(\mathbf{x}_i) \\ \mathbf{\Phi}^\top(\mathbf{x}_j - \hat{\mathbf{x}}_j)\,. \end{pmatrix}. \tag{5}$$

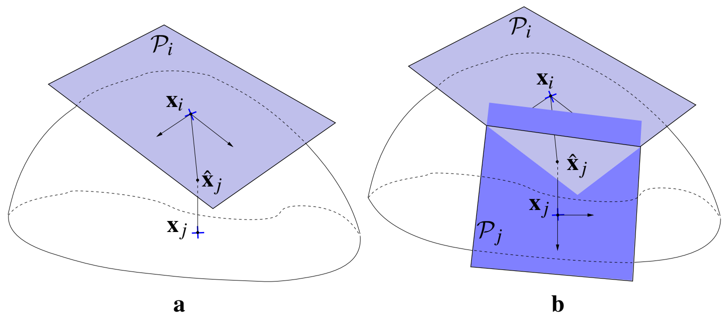


Fig. 3 Higher-dimensional Continuation method applied to a two-dimensional manifold embedded in a 3D ambient space. **a** A chart is defined from the tangent space at a given start point $\mathbf{x}_i$. The area of applicability of the chart is denoted as $\mathcal{P}_i$. A point $\hat{\mathbf{x}}_j$ defined using the tangent space at $\mathbf{x}_i$ is orthogonally projected to the manifold to determine $\mathbf{x}_j$, the root point of the next chart. **b** The new chart locally parametrizes a new region of the manifold. The area of applicability of the new chart is $\mathcal{P}_j$, that does not overlap with $\mathcal{P}_i$.

The update is applied until the norm of the right-hand side of the previous system becomes negligible or for a maximum number of iterations. When a valid $\mathbf{x}_j$ is determined, we can define a new chart $\mathcal{C}_j$, as shown in Fig. 3b. The intersection between tangent spaces marks the boundaries of applicability of the corresponding charts, denoted as $\mathcal{P}_i$ and $\mathcal{P}_j$, respectively. When $\mathcal{C}_i$ is fully surrounded by other charts, $\mathcal{P}_i$ becomes a convex polytope. Note that $\mathcal{P}_i$ is defined in the tangent space associated with $\mathcal{C}_i$ and, thus, it is a polytope in a k-dimensional space and not in the much larger n-dimensional ambient space.

The algorithm proposed in [15] gives a systematic way to define new charts and to generate the associated polytopes. In this work, $\mathcal{P}_i$ is initialized as an hypercube enclosing a ball, $\mathcal{B}_i$, of radius r, as illustrated in Fig. 4a. The polytope is represented using a set of faces that intersect defining a set of vertices [7]. A vertex $\mathbf{v}$ of $\mathcal{P}_i$ external to $\mathcal{B}_i$, can then be used to generate a new chart. From this vertex, a point $\hat{\mathbf{x}}_j$ on the surface of $\mathcal{B}_i$ is defined using Eq. (3) and

$$\mathbf{u}_i^j = \frac{r}{\|\mathbf{v}\|}\,\mathbf{v}\,. \tag{6}$$

If $\mathcal{C}_i$ and the new $\mathcal{C}_j$ generated from $\mathbf{u}_i^j$ are too far or too different, i.e., if

$$\|\mathbf{x}_j - \hat{\mathbf{x}}_j\| > \sigma\,, \tag{7}$$

$$\|\Phi_i^\top \Phi_j\| < 1-\sigma\,, \tag{8}$$

the new chart is discarded and a new attempt of chart generation is performed from a set of parameters $\mathbf{u}_i^j$ closer to $\mathbf{x}_i$. When $\mathcal{C}_j$ is valid, it is used to refine $\mathcal{P}_i$ from the intersection between $\mathcal{B}_i$ and $\hat{\mathcal{C}}_j$, the projection into the tangent space of $\mathcal{C}_i$ of the part of the variety covered by $\mathcal{C}_j$. This projection is approximated by a ball, $\tilde{\mathcal{B}}_j$, included

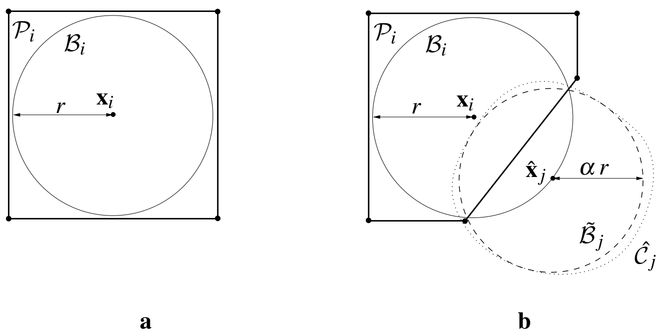


Fig. 4 Polytope-based chart construction. **a** The initial polytope $\mathcal{P}_i$ is a box including a ball of radius r around x_i. **b** The polytope is refined using a ball $\tilde{\mathcal{B}}_j$ that approximates $\hat{\mathcal{C}}_j$, the projection of a neighboring chart into the current chart.

in $\hat{\mathcal{C}}_j$, as shown in Fig. 4b. For this approximation to be conservative, the radius of $\tilde{\mathcal{B}}_j$ is scaled by a factor α, $0 < \alpha < 1$, depending on the angle between sub-spaces spanned by Φ_i and Φ_j.

The hyperplane defined by the intersection of $\mathcal{B}_i$ and $\tilde{\mathcal{B}}_j$ can be computed by subtracting the equations for the two balls. As shown in Fig. 4b, this plane defines a new face of $\mathcal{P}_i$ that eliminates some of its vertices (in particular the one used to generate $\mathcal{C}_j$) and generates new ones. $\mathcal{P}_j$, the polytope associated to $\mathcal{B}_j$, is cropped using the projection of $\mathcal{C}_i$ into $\mathcal{C}_j$.

When all the vertices of the polytope of a chart are inside the associated ball, the chart cannot be further expanded as the domain for this chart is fully bounded. This process of chart expansion continues as far as there are open charts. At the end, the connected component of the variety containing the initial point is fully covered by a set of chats whose area of validity is bounded by the corresponding polytopes (see Fig. 2a). To fully characterize the connected component of the variety, higher-dimensional continuation tools need to consider configuration space singularities, where the variety bifurcates [16]. Here, we consider only the case were the Jacobian of $\mathbf{F}$ at singularities is of rank $n-k-1$. In this case, the singularities form a zero-measure set that can be located by monitoring an indicator function, $\chi(\mathbf{x})$, whose value is different for two points on opposite sides of the singularity set [4]. When located, singular points are accurately determined using a dichotomic search and the additional vector of the null space of the Jacobian at the singular point is used to define a point on the other branch of the variety. This point is used as a root for a new chart from where to start the coverage of the additional branch of the variety.

The cost of the algorithm at each step is dominated by the cost of two searches among the set of charts: one to find an open chart and another to find the potential neighbors of a new chart. The first search can be saved keeping the open charts in a list. The performance of the second search can be increased using a kd-tree storing the root points of the charts.

3 Path-Planning on Manifolds

Using the tools described in the previous section, a graph can be built where nodes are the chart roots and edges represent the neighboring relations between charts. Thus, the shortest connecting two given points can be easily computed using a standard graph search method such as A* considering only the collision-free transitions between the chart roots. This procedure defines an optimal, resolution complete path planner, but it is only practical for low dimensional varieties, specially if we have to use charts with small applicability areas. If we define charts with large radius, the presence of obstacles becomes an issue. If we use a coarse resolution in an environment with many obstacles, most of the transitions between chart centers will be in collision and it will not be possible to find ways out among obstacles.

Herein, we propose modifications to the Higher-order Continuation procedures to deal with the curse of dimensionality and the presence of obstacles. First, we take advantage of that path planning is only concerned with the path between two given configurations and not with the full atlas generation, which allows to save the construction of many unnecessary charts. Second, to deal with the presence of obstacles, we randomize the process of atlas extension and adapt the generation of charts to the presence of obstacles.

3.1 Chart Selection: Focusing on the Path to the Goal

As aforementioned, the atlas structure can be represented by a graph where nodes are the charts and edges are the neighboring relations between charts. To guide the search toward the goal, we use a Greedy Best-First search where the chart to expand is the one with minimum expected cost to reach the goal. The cost for a chart $\mathcal{C}_i$ is heuristically evaluated as

$$h(i) = \beta^{n_i} \left\| \mathbf{x}_i - \mathbf{x}_g \right\| , \tag{9}$$

where $\mathbf{x}_g$ is the goal configuration, $\beta > 1$ a fixed parameter, and n_i is the number of times a chart failed to expand. Thus, the term β^{n_i} prevents the search to get stuck in dead ends. As soon as the goal is connected to the rest of the atlas, the search is stopped.

Observe that using a Greedy Best-First search, we do not necessary generate all the neighbors of the chart under expansion. The generation of children charts proceeds only while the children have higher cost than the parent. This largely reduces the generation of charts.

Finally, note that due to the use of a Greedy Best-First algorithm the final path is not necessarily optimal. As mentioned, the generation of a (resolution) optimal path would require the use of an A* algorithm, the generation of all the neighbors of the node/chart under expansion, and the use of a small resolution to deal with the presence of obstacles. In general, this implies to generate too many charts, hindering the practical applicability of the approach.

3.2 Chart Expansion: Generating Random Directions

When the chart to be expanded is selected, the expansion point for the atlas is selected at random. This is achieved by sampling a point uniformly on the surface of the ball associated with the atlas and checking if this point is inside the associated polytope. If it is the case, the generation of the new atlas proceeds as detailed in Sect. 2.

The uniform generation of random points, $\mathbf{u}_i^j$, on a the surface of a k-dimensional ball is done by generating the point elements according to a normalized one-dimensional Gaussian and scaling the resulting vector to norm r [10].

To verify if the point is inside the associated chart's polytope, we exploit the fact that convex polytopes are defined as the intersection of k-dimensional hyperplanes. Thus, for a point $\mathbf{u}_i^j = \{u_1, \ldots, u_k\}$ to be inside the polytope $\mathcal{P}_i$ made of m_i faces, it must fulfill

$$\gamma_0^t + \sum_{s=1}^{k} \gamma_s^t u_s \geq 0 \,, \tag{10}$$

for all the faces $f^t = (\gamma_0^t, \ldots, \gamma_k^t)$, $t = 1, \ldots, m_i$ defining $\mathcal{P}_i$.

If the point is inside the polytope, it is approached through small incremental steps of size δ. The intermediate points are successively projected on the manifold using Eqs. (4) and (5) and then checked for collision. The last collision-free configuration is used as a root for a new chart. This adjusts the distribution of charts to the free configuration space. If no progress at all can be done towards the target point, the expansion is declared as failure and the chart under expansion is penalized by increasing n_i. Since collisions are not checked between intermediate points, δ has to be set small enough so that only minor interpenetrations could occur.

Observe that while the chart to extend is selected greedily, the exact expanding direction is selected randomly, favoring the exploration of alternative paths in the presence of obstacles.

3.3 Algorithm

Algorithm 1 corresponds to the HC-Planner, implementing the path planning approach introduced in this paper. The algorithm takes $\mathbf{x}_s$ and $\mathbf{x}_g$ as start and goal configurations respectively, and tries to connect them with a path on the variety implicitly defined by a given set of constraints $\mathbf{F}$, as expressed in Eq. (1). The process begins by initializing two charts associated to the two query configurations (lines 1 and 2). Each chart includes the root point $\mathbf{x}$, the base of the tangent space $\mathbf{\Phi}$, the ball $\mathcal{B}$ and the polytope $\mathcal{P}$ limiting the area of applicability of the chart. The two charts are then included in the initial atlas, A (line 3). To efficiently determine the chart with the minimum expected cost, charts are organized into a binary heap. Thus, the cost-to-goal of the start configuration is evaluated (line 4) and used to initialize the heap (line 5). In lines 6 to 16, a greedy search is performed as described in Sect. 3.1, while the two query configurations are disconnected. At each iteration, we extract C_i, the most promising chart from the heap (line 7) and if the polytope $\mathcal{P}_i$ of

Algorithm 1: High-dimensional Continuation path planner.

```
   HC-Planer($\mathbf{x}_s, \mathbf{x}_g, \mathbf{F}$)
   input  : A couple of samples to connect $\mathbf{x}_s$, $\mathbf{x}_g$, a set of constraints $\mathbf{F}$.
   output: A path connecting the two samples
1  $\mathcal{C}_s \leftarrow$ InitChart($\mathbf{x}_s, \mathbf{F}$)          // $\mathcal{C}_s = \{\mathbf{x}_s, \Phi_s, \mathcal{B}_s, \mathcal{P}_s\}$
2  $\mathcal{C}_g \leftarrow$ InitChart($\mathbf{x}_g, \mathbf{F}$)          // $\mathcal{C}_g = \{\mathbf{x}_g, \Phi_g, \mathcal{B}_g, \mathcal{P}_g\}$
3  $A \leftarrow \{\mathcal{C}_s, \mathcal{C}_g\}$
4  $h(s) \leftarrow \|\mathbf{x}_s - \mathbf{x}_g\|$
5  $H \leftarrow$ InitHeap($\mathcal{C}_s, h(s)$)
6  while not Connected($A, \mathcal{C}_s, \mathcal{C}_g$) do
7      $\mathcal{C}_i \leftarrow$ ExtractMin($H$)          // $\mathcal{C}_i = \{\mathbf{x}_i, \Phi_i, \mathcal{B}_i, \mathcal{P}_i\}$
8      if $\mathcal{P}_i \nsubseteq \mathcal{B}_i$ then
9          $\mathcal{C}_j \leftarrow$ GenerateNewChart($\mathcal{C}_i, \mathbf{F}$)
10         if $\mathcal{C}_j = \emptyset$ then
11             $h(i) \leftarrow \beta\, h(i)$
12             $H \leftarrow$ AddToHeap($\mathcal{C}_i, h(i)$)
13         else
14             $A \leftarrow A \cup \{\mathcal{C}_j\} \cup$ SingularChart($\mathcal{C}_i, \mathcal{C}_j$)
15             $h(j) \leftarrow \|\mathbf{x}_j - \mathbf{x}_g\|$
16             $H \leftarrow$ AddToHeap($\mathcal{C}_j, h(j)$)
17 Return(Path($A, \mathcal{C}_s, \mathcal{C}_g$))
```

this chart still has vertices outside the ball $\mathcal{B}_i$ (line 8), we try to extend the atlas with a new chart (line 9). If the extension fails (line 10), the current chart is penalized so that its chance to be selected for future extension decreases (line 11), and the chart is added to the heap with the updated cost (line 12). If the atlas extension succeeds, the new chart is added to the atlas, updating the neighboring relations between charts (line 14). Next, the heuristic-to-goal is initialized for the new chart (line 15) and added to the atlas (line 16). When the goal is reached, a graph search procedure can be used to extract the path linking the query configurations via the roots of some of the charts in the atlas. Every time a new chart is added to the atlas, we check whether the line connecting the roots of the parent and the child charts crosses a singularity. If so, the singular point is located and an additional chart is defined such that its root is at the singularity and its tangent space is aligned with the branch of the variety that does not contain $\mathcal{C}_i$ and $\mathcal{C}_j$. Function SingularChart (line 14) implements this process and returns the new chart or and empty set if there is no singularity between $\mathcal{C}_i$ and $\mathcal{C}_j$.

The generation of a new chart from a previous one is presented in Algorithm 2. We select a point $\mathbf{u}_i^j$ on surface of the ball defined on the tangent space of the input chart (line 2), as described in Sect. 3.2. If the point is inside the polytope (line 3), i.e., the point is in the area of influence of the current chart, we proceed to determine a point, $\mathbf{x}_j$, adequate to generate a new chart. This point is searched from the root of

Algorithm 2: Generation of a new chart.

```
GenerateNewChart($\mathcal{C}_i,\mathbf{F}$)
input : A chart to expand $\mathcal{C}_i=\{\mathbf{x}_i,\Phi_i,\mathcal{B}_i,\mathcal{P}_i\}$, a set of constraints $\mathbf{F}$.
output: A new chart $\mathcal{C}_j$
1  $\mathcal{C}_j \leftarrow \emptyset$
2  $\mathbf{u}_i^j \leftarrow$ RandomInBall($\mathcal{B}_i$)
3  if $\mathbf{u}_i^j \in \mathcal{P}_i$ then
4      $e \leftarrow$ True      // Small error with respect to $\mathcal{C}_i$
5      $c \leftarrow$ True      // Collision-free
6      $t \leftarrow$ True      // Tangent space similar to $\mathcal{C}_i$
7      $s \leftarrow \delta$
8      while $s \leq r$ and $e$ and $c$ and $t$ do
9          $\hat{\mathbf{x}}_j \leftarrow \mathbf{x}_i + \Phi_i\, s\, \mathbf{u}_i^j/r$
10         $\mathbf{x}_j \leftarrow$ Project($\mathcal{C}_i,\hat{\mathbf{x}}_j,\mathbf{F}$)
11         if $\|\hat{\mathbf{x}}_j-\mathbf{x}_j\| > \sigma$ then
12             $e \leftarrow$ False
13         else
14             if Collision($\mathbf{x}_j$) then
15                 $c \leftarrow$ False
16             else
17                 $\Phi_j \leftarrow$ TangentSpace($\mathbf{x}_j,\mathbf{F}$)
18                 if $\|\Phi_i^\top \Phi_j\| < 1-\sigma$ then
19                     $t \leftarrow$ False
20                 else
21                     $\mathcal{C}_j \leftarrow$ InitMap($\mathbf{x}_j,\mathbf{F}$)      // $\mathcal{C}_j=\{\mathbf{x}_j,\Phi_j,\mathcal{B}_j,\mathcal{P}_j\}$
22                     $s \leftarrow s+\delta$
23 Return($\mathcal{C}_j$)
```

the chart under expansion, progressively moving to the target point with incremental steps of size δ. At each step, we project the point from the tangent space to the manifold (lines 9-10), implementing Eqs. (4) and (5). If the projection converges to a point in the manifold, we check whether the obtained point is too far away from the tangent space (line 11), whether it is in collision (line 14), and whether the tangent space at the new point, computed using Eq. (2), and that of $\mathcal{C}_i$ are too different (line 18). In any of these cases, the progress towards the new point is stopped (lines 12, 15, and 19) and we return the chart for the last valid point (line 23), if any.

The main operations of the HC-planner scale as follows. The initialization of a chart scales with $O(n^3+2^k)$, with n the dimensionality of the ambient space and k the dimensionality of the configuration space, since we use a QR decomposition to identify a base of the kernel of the Jacobian of $\mathbf{F}$ and we have to define a box with 2^k vertices. The initialization of the heap is $O(1)$ and the extraction and removal of its minimum element is $O(k)$. The generation of a new chart scales with $O(n^3+k\,2^k)$

since it includes the five following steps: 1) the generation of a random number on a k-dimensional ball that is $O(k)$; 2) the check to determine if a point is inside a k-dimensional polytope, that scales as $O(k\,2^k)$ since each face is defined by a $k+1$ dimensional vector and the number of neighbors of a chart grows with the kissing number that is $O(2^k)$; 3) the projection of a point from the tangent space to the manifold, a Newton process with a bounded number of iterations, where at each iteration we use a QR decomposition that is $O(n^3)$; 4) the determinant of a matrix of size k, $O(k^3)$, that comes from the product of $n \times k$ matrices that is $O(k^2\,n)$; and, finally, 5) the initialization of a chart, $O(n^3+2^k)$. When adding a chart to the atlas, we have to look for neighboring charts. This can be done in $O(k)$ since it is logarithmic with the number of charts that, in the worst case scales exponentially with k. For the neighboring charts, we have to crop the corresponding polytopes. This operation scales with the number of vertices of those polytope which is $O(2^k)$. Finally, the addition of an element to the heap is $O(k)$, in agreement with the cost of determining the neighboring relations between charts.

Summarizing, if l is the number of charts needed to connect the start and the goal the overall algorithm scales with $O(l\,(n^3+k\,2^k))$. In very constrained problems, as the ones we consider, $k \ll n$ and the cost is dominated by $O(l\,n^3)$. In the worst case, the final atlas might include all the possible charts for a given manifold and l is exponential in k and independent of n. However, as we show in next section, many problems require in practice a limited number of charts to connect the start and goal configurations.

Note that the planner is resolution complete, in the sense that by taking a radius r small enough we can ensure to find a solution path if it exists. In particular, in problems involving narrow passages of minimum width υ, setting $r < \upsilon/2$ would ensure a solution. However, in practice, much larger radius can be used safely.

4 Experiments

We implemented in C the higher dimensional continuation tools[1] described in Sect. 2 and the HC-planner described in Sect. 3, including the treatment of configuration space singularities. These tools were integrated as modules of our position analysis toolbox [32] using SOLID [3, 34] as a collision detector and the GNU Scientific Library for the lineal algebra operations. Our position analysis toolbox is based on a formulation that yields a system of simple equations only containing linear, bilinear, and quadratic monomials, and trivial trigonometric terms for the helical pair only [21]. The simplicity of the final system of equations makes it advantageous for continuation methods [37]. For the purpose of comparison, we also implemented the RRT for constrained spaces presented in [9]. In this RRT, points are sampled in the ambient space and the nearest sample on the variety is progressively extended towards the random sample. At each extension step, the points are projected to the variety using the Jacobian pseudo inverse method. In our implementation, the nearest-neighbor queries use the kd-tree described in [41]. The maximum

[1] An implementation of these tools tailored for dynamical systems is available in [33].

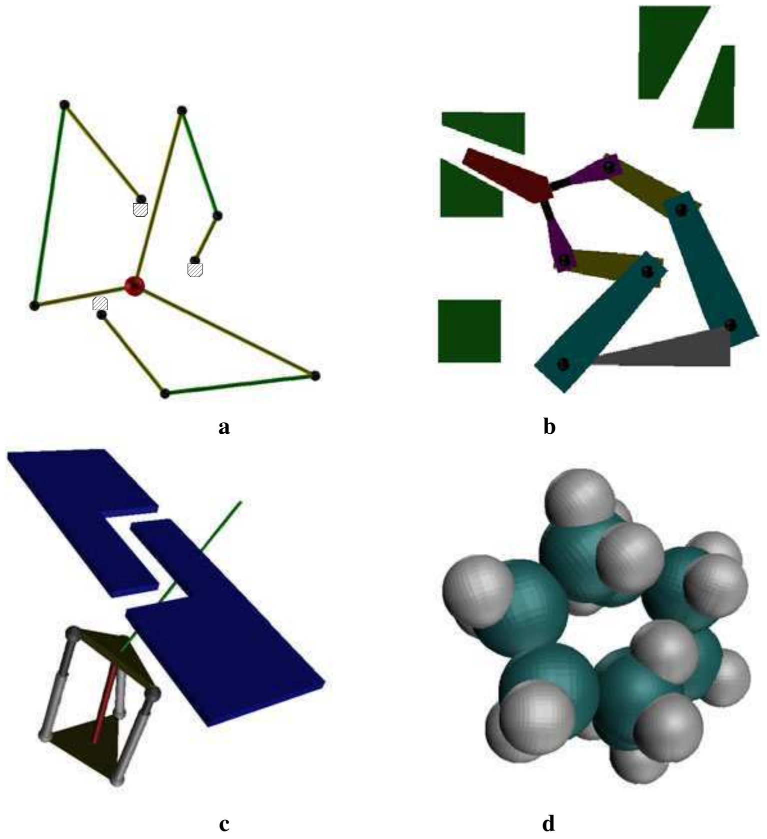


Fig. 5 The four benchmarks used. **a** Star-shaped planar manipulator with three fixed extremities. **b** Two-arms manipulator moving an object from one gap to another. **c** Rotational-only parallel manipulator. **d** Cyclooctane molecule.

number of nodes in the RRT is set to 75000. Experiments were executed on a Intel Core 2 at 2.4 Ghz running Linux. Finally, the algorithm parameters were set to $r = 0.4$, $\delta = 0.04$, $\sigma = 0.1$, and $\beta = 1.1$ for all the experiments.

Figure 5 shows the four benchmarks used in this paper. The first one is a planar star-shaped manipulator also used in [28]. In this case, obstacles are not considered. The second problem involves a system where two arms have to cooperate to move an object from one gap to another. This problem previously appears in [11]. The movement between the start and goal configurations requires to traverse actuator singularities, which makes the problem unsolvable by basic Kinematics-based Roadmap approaches [8, 12]. The third example, kindly provided by Juan Cortés, is a parallel platform with rotation motion only. The task here is to move a stick attached to the robot across some obstacles. The last benchmark is the cyclooctane,

Table 1 Dimensionality of the ambient and configuration spaces, execution times and number of samples/charts used by a RRT and the HC-planner

			RRT		HC		RRT/HC	
Benchmarks	k	n	Time	Samples	Time	Charts	T/T	S/C
Star-shaped	5	18	3.87	5515	0.47	199	8.23	27.71
Two-arms	3	10	22.94	27130	0.35	391	65.54	69.38
Parallel	3	27	36.97	13400	0.92	276	40.18	48.55
Cyclooctane	2	17	6.49	6813	0.28	152	23.17	44.82

a molecule whose kinematics is a 8-revolute loop. Here, we have to find a path between two conformations that avoids self-collisions involving carbon and hydrogen atoms (depicted in cyan and white in the figure, respectively).

Table 1 shows the performance comparison, averaged over 100 runs, between RRT and the HC-planner. For each of the four benchmarks, the table gives the dimensionality of the configuration space (k), the dimensionality of the ambient space (n), the execution times and the number of samples or charts used for each method. The table also shows execution time ratios (T/T) and the ratio between the number samples used in RRT and the number of charts used with the HC-planner (S/C). Note that the RRT in the Two-arms test case is unable to find a solution for 25% of the cases. In Table 1, the RRT results for this problem correspond to averages for the successful tests only.

The results show that, for this set of problems, the execution time of the RRT is more than one order of magnitude higher than that of the HC-planner, except for the Star-shaped problem where the HC-planner is only about a 8 times faster. This is true despite the generation of samples being much faster than the generation of charts. This is so because charts are more powerful since they do not only describe the variety on a single point but on a local neighborhood of a point. Thus, the HC-algorithm uses in average 40 times less charts than samples used by RRT.

The advantage of the HC-planner search strategy with respect to a more optimal strategy is evaluated by applying a standard A* algorithm, implemented as described in the introduction of Sect. 3. A* can not solve the Star-shaped problem with less than 10000 charts. This is due to the curse of dimensionality: in a 5-dimensional configuration space the number of neighboring charts for each chart is about 40 and this results in a large exponential growth of the number of chart to generate even for simple problems. The Two-arms test can not be solved with $R = 0.4$ since with such a coarse resolution, almost all the transitions between chart centers are in collision. The problem can only be solved with R below 0.25. However, at this resolution the number of charts to generate increases to the point that the A* execution time is about 20 seconds. Our approach dynamically adapts the root of the charts to the distribution of obstacles, avoiding the generation of charts on blocked regions. Finally, both the Parallel and the Cyclooctane problems can be solved with A*, but at a higher cost than the one used by RRT. In the cases that can be solved, though, A* returns a path that, in average, is about half the length of those obtained both with the HC-planner or with RRT. However, the path length for both the HC-planner and the RRT could be improved with a smoothing post-process, which is not yet implemented in our system. Finally, note that A* is able to detect if two samples can not be connected because they are in different connected components of the variety even in the presence of obstacles: in the worst case the atlas for the connected component including the start sample will be completed and if it does not include the goal sample the planning can be declared as a failure. The HC-planner trades off this completeness for efficiency introducing a randomness that prevents an atlas to be completed in the presence of obstacles.

5 Conclusions

In this paper, we extended the use of High-dimensional Continuation algorithmic tools for path planning applications. Using these tools, we defined a randomized path planner for highly constrained systems. The presented planner directly works on the configuration space, trying to connect any pair of query configurations with a small collection of local charts. The algorithm performance is highly independent of the relation between the configuration space and the ambient space. This is in contrast with existing sampling algorithms for constrained problems that generate samples in the ambient space. The experiments show that our approach can be more than one order of magnitude faster than state of the art algorithms.

The worst case cost of the algorithm introduced in this paper is exponential with the dimension of the configuration space, which is in agreement with the cost of the best complete path planners [6]. Thus, the algorithm would not scale gracefully to high-dimensional problems. Despite this, the use of a greedy search strategy together with the randomization allow to solve problems with moderate complexity (at least up to dimension 5 in the examples) embedded in even higher-dimensional spaces. Problems slightly more complexes than this are also likely to be addressable with the presented planner and this includes many interesting problems in Robotics and in Molecular Biology [5]. To scale to problems with even larger dimensionality we could rely on charts with larger area of influence. However this is likely to be valid only in almost lineal problems, where the error between the tangent space and the solution variety remains small over large areas. Moreover, the use of large charts limits the set of problems that can be solved since environments densely populated with obstacles typically require small charts. We would like to explore the possibility to define variants of the HC-planner where the role of the resolution is minimized in the same way as probabilistic roadmaps overcome the resolution limitations of cell decomposition methods. Another possibility to explore is to define a cost function over the configuration space so that the exploration could be limited to areas with low cost [18]. All these points deserve a more careful evaluation. It is also our future endeavor to perform a more thorough experimental and theoretical analysis of the proposed algorithm, focusing on the performance of the algorithm for different obstacle settings.

Acknowledgements. We would like to thank L. Ros for pointing us to the Higher-dimension Continuation tools and for fruitful discussions during the elaboration of this work. This work has been partially supported by the Spanish Ministry of Science and Innovation under project DPI2007-60858.

References

1. Ballantyne, G., Moll, F.: The da Vinci telerobotic surgical system: Virtual operative field and telepresence surgery. Surgical Clinics of North America 83(6), 1293–1304 (2003)
2. Berenson, D., Srinivasa, S.S., Ferguson, D., Kuffner, J.J.: Manipulation planning on constraint manifolds. In: IEEE International Conference on Robotics and Automation, pp. 1383–1390 (2009)
3. van den Bergen, G.: Efficient collision detection of complex deformable models using AABB trees. Journal of Graphics Tools 2(4), 1–13 (1997)
4. Beyn, W.J., Champneys, A., Doedel, E., Govarets, W., Kuznetsov, U.A., Yu, A., Sandstede, B.: Numerical Continuation, and Computation of Normal Forms. In: Handbook of Dynamical Systems, vol. 2, pp. 149–219. Elsevier, Amsterdam (2002)
5. Brown, W.M., Martin, S., Pollock, S.N., Coutsias, E.A., Watson, J.P.: Algorithmic dimensionality reduction for molecular structure analysis. Journal of Chemical Physics 129(6), 64, 064,118 (2008)
6. Canny, J.: The Complexity of Robot Motion Planing. MIT Press, Cambridge (1988)
7. Chen, P.C., Hansen, P., Jaumard, B.: On-line and off-line vertex enumeration by adjacency lists. Operation Research Letters 10, 403–409 (1991)
8. Cortés, J., Siméon, T., Laumond, J.P.: A random loop generator for planning the motions of closed kinematic chains using PRM methods. In: IEEE International Conference on Robotics and Automation, pp. 2141–2146 (2002)
9. Dalibard, S., Nakhaei, A., Lamiraux, F., Laumond, J.P.: Whole-body task planning for a humanoid robot: a way to integrate collision avoidance. In: IEEE-RAS International Conference on Humanoid Robots, pp. 355–360 (2009)
10. Fishman, G.F.: Monte Carlo: Concepts, Algorithms, and Applications. Springer, Heidelberg (1996)
11. Gharbi, M., Cortés, J., Siméon, T.: A sampling-based path planner for dual-arm manipulation. In: IEEE/ASME International Conference on Advanced Intelligent Mechatronics, pp. 383–388 (2008)
12. Han, L., Amato, N.M.: A kinematics-based probabilistic roadmap method for closed chain systems. In: Algorithmic and Computational Robotics - New Directions (WAFR 2000), pp. 233–246 (2000)
13. Han, L., Rudolph, L.: Inverse kinematics for a serial chain with joints under distance constraints. In: Robotics: Science and Systems, pp. 177–184 (2005)
14. Havoutis, I., Ramamoorthy, S.: Motion synthesis through randomized exploration of submanifolds of configuration spaces. In: Baltes, J., Lagoudakis, M.G., Naruse, T., Ghidary, S.S. (eds.) RoboCup 2009. LNCS(LNAI), vol. 5949, pp. 92–103. Springer, Heidelberg (2010)
15. Henderson, M.E.: Multiple parameter continuation: Computing implicitly defined k-manifolds. International Journal of Bifurcation and Chaos 12(3), 451–476 (2002)
16. Henderson, M.E.: Multiparameter parallel search branch switching. International Journal of Bifurcation and Chaos in Applied Science and Engineering 15(3), 967–974 (2005)

17. Henderson, M.E.: Higher-Dimensional Continuation. In: Numerical continuation methods for dynamical systems: path following and boundary value problems. Springer, Heidelberg (2007)
18. Jaillet, L., Cortés, J., Siméon, T.: Sampling-based path planning on configuration-space costmaps. IEEE Transactions on Robotics 26(4), 635–646 (2010)
19. Krauskopf, B., Osinga, H.M., Galán-Vioque, J.: Numerical continuation methods for dynamical systems: path following and boundary value problems. Springer, Heidelberg (2007)
20. Ott, C., Eiberger, O., Friedl, W., Bauml, B., Hillenbrand, U., Borst, C., Albu-Schafer, A., Brunner, B., Hirschmuller, H., Hirzinger, G.: A humanoid two-arm system for dexterous manipulation. In: IEEE-RAS International Conference on Humanoid Robots, pp. 276–283 (2006)
21. Porta, J.M., Ros, L., Thomas, F.: A linear relaxation technique for the position analysis of multiloop linkages. IEEE Transactions on Robotics 25(2), 225–239 (2009)
22. Pressley, A.: Elementary Differential Geometry. Springer, Heidelberg (2001)
23. Rheinboldt, W.C.: MANPACK: A set of algorithms of computations on implicitly defined manifolds. Computers and Mathematics with Applications 32(12), 15–28 (1996)
24. Rodríguez, A., Basañez, L., Celaya, E.: A relational positioning methodology for robot task specification and execution. IEEE Transactions on Robotics 24(3), 600–611 (2008)
25. Rosales, C., Ros, L., Porta, J.M., Suárez, R.: Synthesizing grasp configurations with specified contact regions. International Journal of Robotics Research (2010)
26. Roth, B., Freudenstein, F.: Synthesis of path-generating mechanisms by numerical methods. ASME Journal of Engineering for Industry 85, 298–307 (1963)
27. Shkolmik, A., Tedrake, R.: Path planning in 1000+ dimensions using a task-space voronoi bias. In: IEEE International Conference on Robotics and Automation, pp. 2892–2898 (2009)
28. Shvlab, N., Liu, G., Shoham, M., Trinkle, J.C.: Motion planning for a class of planar closed-chain manipulators. International Journal of Robotics Research 26(5), 457–473 (2007)
29. Sommese, A.J., Wampler, C.W.: The Numerical Solution of Systems of Polynomials Arising in Engineering and Science. World Scientific, Singapore (2005)
30. Stilman, M.: Task constrained motion planning in robot joint space. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 3074–3081 (2007)
31. Tang, X., Thomas, S., Coleman, P., Amato, N.M.: Reachable distance space: Efficient sampling-based planning for spatially constrained systems. International Journal of Robotics Research 29(7), 916–934 (2010)
32. The CUIK project web page:
`http://www.iri.upc.edu/research/webprojects/cuikweb`
33. The MultiFario project web page, `http://multifario.sourceforge.net`
34. The SOLID web page, `http://www.dtecta.com`
35. Tsai, L.W.: Robot Analysis: The Mechanics of Serial and Parallel Manipulators. John Wiley and Sons, Chichester (1999)
36. Um, T.T., Kim, B., Suh, C., Park, F.C.: Tangent space RRT with lazy projection: An efficient planning algorithm for constrained motions. In: Advances in Robot Kinematics, pp. 251–260 (2010)
37. Wampler, C., Morgan, A.: Solving the 6R inverse position problem using a generic-case solution methodology. Mechanism and Machine Theory 26(1), 91–106 (1991)
38. Wedemeyer, W.J., Scheraga, H.: Exact analytical loop closure in proteins using polynomial equations. Journal of Computational Chemistry 20(8), 819–844 (1999)

39. Yakey, J.H., LaValle, S.M., Kavraki, L.E.: Randomized path planning for linkages with closed kinematic chains. IEEE Transactions on Robotics and Automation 17(6), 951–959 (2001)
40. Yao, Z., Gupta, K.: Path planning with general end-effector constraints: Using task space to guide configuration space search. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 1875–1880 (2005)
41. Yershova, A., LaValle, S.M.: Improving motion planning algorithms by efficient nearest neighbor searching. IEEE Transactions on Robotics 23(1), 151–157 (2007)
42. Yershova, A., LaValle, S.M.: Motion planning for highly constrained spaces. In: Robot Motion and Control. Lecture Notes on Control and Information Sciences, vol. 396, pp. 297–306 (2009)

Algorithms and Analytic Solutions Using Sparse Residual Dipolar Couplings for High-Resolution Automated Protein Backbone Structure Determination by NMR

Anna Yershova[⋆], Chittaranjan Tripathy[⋆], Pei Zhou, and Bruce Randall Donald[⋆⋆]

Abstract. Developing robust and automated protein structure determination algorithms using nuclear magnetic resonance (NMR) data is an important goal in computational structural biology. Algorithms based on global orientational restraints from residual dipolar couplings (RDCs) promise to be quicker and more accurate than approaches that use only distance restraints. Recent development of analytic expressions for the roots of RDC equations together with protein kinematics has enabled exact, linear-time algorithms, highly desirable over earlier stochastic methods. In addition to providing guarantees on the number and quality of solutions, exact algorithms require a minimal amount of NMR data, thereby reducing the number of NMR experiments. Implementations of these methods determine the solution structures by explicitly computing the intersections of algebraic curves representing discrete RDC values. However, if additional RDC data can be measured, the algebraic curves no longer generically intersect. We address this situation in the paper and show that globally optimal structures can still be computed analytically as points closest to all of the algebraic curves representing the RDCs. We present new algorithms that expand the types and number of RDCs from which analytic solutions are computed. We evaluate the performance of our algorithms on NMR data for four proteins: human ubiquitin, DNA-damage-inducible protein I (DinI), the Z domain of staphylococcal protein A (SpA), and the third IgG-binding domain of Protein G

Anna Yershova · Chittaranjan Tripathy
Department of Computer Science, Duke University, Durham, NC 27707, USA

Pei Zhou
Department of Biochemistry, Duke University Medical Center, Durham, NC 27707, USA

Bruce Randall Donald
Department of Computer Science, Duke University, and Department of Biochemistry, Duke University Medical Center, Durham, NC 27707, USA
e-mail: `brd+wafr2010@cs.duke.edu`

[⋆] These authors contributed equally.

[⋆⋆] Corresponding author.

(GB3). The results show that our algorithms are able to determine high-resolution backbone structures from a limited amount of NMR data.

1 Introduction

Understanding the structures of biologically important proteins is one of the long-term goals in biochemistry. While automation has advanced many other aspects of biology, three-dimensional (3D) protein structure determination remains a slower, harder, and more expensive task. The speed at which protein structures are being discovered today is, for example, several orders of magnitude slower than that of gene sequencing.

Two established experimental approaches for protein structure determination are X-ray crystallography and nuclear magnetic resonance (NMR) spectroscopy. While crystallography can provide high resolution structures when good quality crystals can be grown, NMR spectroscopy is the only experimental technique currently capable of measuring atomic-resolution geometric restraints and dynamics of proteins in physiologically-relevant solution state conditions. This sets a challenging goal in computational structural biology: design efficient automated structure determination techniques that exploit geometric restraints collected using NMR.

Even with recent advances in reducing the acquisition time for NMR data [1] it still remains advantageous to be able to determine protein structure from fewer experiments [2]. Therefore, to design successful computational approaches the interplay between all of the components of the structure determination process should be taken into account: the time and cost of NMR data acquisition, the types of NMR data and their information content, the algorithmic complexity of the problem of computing the 3D structure, and the accuracy of the obtained solution.

Previous algorithms for structure determination problem by NMR can be divided into three categories: *stochastic search, systematic search*, and *exact* algorithms. The first category includes many widely-used approaches [3, 4, 5, 6, 7, 8], which perform stochastic search over possible structures, scored according to their agreement with experimental data. These methods suffer from well-known pitfalls, such as local minima, undersampling, non-convergence, and missed solutions. While stochastic methods may perform adequately in data-rich settings, collecting a large number of NMR spectra increases the time and cost of the experiments, and still provides no guarantee on the quality of solutions.

The second category involves systematic grid search over possible structures [9, 10, 11, 12] and scoring according to the experimental data fit. Excessive computational cost and undersampling due to insufficient resolution are the limitations of these methods.

The development of sampling methodology was influenced by the computational complexity of the structure determination problem using the *nuclear Overhauser effect* (NOE) data. The recent introduction of *residual dipolar couplings* (RDCs) [13, 14] has enabled novel attacks on the problem. In contrast to NOEs, which represent local distance restraints, RDCs measure the global orientation of internuclear vectors. While many RDC-based methods still use stochastic search [15, 16, 9, 17, 18, 12, 19, 10] despite its pitfalls, exact algorithms [20, 21, 2] have recently emerged. These methods explicitly represent the RDC equations as well as protein kinematics in algebraic form to compute structures that optimize the fit to the RDC data. The exact algorithms not only guarantee completeness and polynomial running time, they also use a sparse set of RDC measurements (e.g. only two or three RDCs per residue), which reduces the time and cost of collecting experimental data.

Implementations of these methods determine the solution structures by explicitly computing the intersections of algebraic curves representing discrete RDC values and protein kinematics [20, 21, 2]. However, if additional RDC data is measured, the algebraic curves representing the RDCs no longer generically intersect. Even though collecting additional experimental data may improve convergence of stochastic structure determination methods, development of efficient new exact methods is needed to handle this scenario. In this paper, we present algorithms that expand the types and number of RDC data from which analytic solutions are computed. When additional orientational restraints can be measured, the globally optimal structures are calculated as points closest to all of the algebraic curves representing the RDC constraints. Therefore, the structures that optimally agree with the collected experimental data are determined analytically.

Moreover, with additional RDC data our algorithms can compute structures of *loops*, more challenging regions in a protein for structure determination compared to *secondary structure elements (SSEs)*, such as α-helices and β-sheets. Loops increase computational complexity of the exact algorithms, because there is less physical constraints on the structures that are possible for loops to assume, as is reflected in their Ramachandran statistics [22]. Therefore, we expand the domain of applicability of exact algorithms to structure determination problems which in practice only stochastic methods could handle before. This extends the benefits of analytic solutions into a new and important domain.

In summary, the following contributions are made in this paper:

- We present a general framework for computing protein backbone structures from sparse RDC measurements.
- We describe new algorithms that handle two particular types of RDC data: two RDCs per residue in one or two alignment media, and multiple RDCs per residue in multiple media.
- We present the analysis of our methods in terms of the upper bound on the number of optimal structures the algorithm can generate.
- We evaluate the performance of our methods on NMR data sets for four proteins: human ubiquitin, DNA-damage-inducible protein I (DinI), the Z domain of staphylococcal protein A (SpA), and the third IgG-binding domain of Protein G (GB3).

We start with background on RDCs in Section 2. We formally define the protein structure determination problem in Section 3. Section 4 presents the general framework for our exact algorithms, as well as detailed explanations of the methods that handle two different types of RDC data (Sections 4.1 and 4.2). We present a study on the performance of our methods in Section 6 and conclusions in Section 7.

2 Background

The Physics of RDCs. During an NMR experiment, a protein in solution is placed in a static magnetic field. The magnetic field interacts with the nuclear spins of atoms of the protein, which allows collecting various NMR measurements.

Consider a vector **v** between two NMR-active nuclear spins, *a* and *b*, of two atoms, and a vector representing the gradient of the magnetic field *B* in Figure 1. An RDC experiment detects the interaction between the nuclear spins. This interaction is measured in units of Hertz and can be expressed as

$$D = \frac{\mu_0 \gamma_a \gamma_b \hbar}{4\pi^2 \left\langle r_{a,b}^3 \right\rangle} \left\langle \frac{3\cos^2\theta - 1}{2} \right\rangle, \tag{1}$$

in which D is the residual dipolar coupling measurement, μ_0 is the magnetic permeability of vacuum, $\hbar$ is the reduced Planck's constant, γ is the gyromagnetic ratio, $r_{a,b}$ is the distance between the two spins, θ is the angle between the internuclear vector, **v**, and B. The angle brackets represent an average over time and an ensemble of proteins in solution.

Intuitively, as the protein tumbles in solution, the internuclear vector tumbles with the protein. The RDC measurement represents the time average over such movements. This measurement is of little use if the protein tumbles freely (isotropically), since the average is zero and hence no RDC value is detected. An anisotropic solution is used to constrain the motions of the protein by adding an *alignment medium* to the solution. The resulting RDC measurement represents the relationship between the internuclear vector and the set of parameters associated with the alignment medium, known as the *Saupe matrix* [23], or *alignment tensor*.

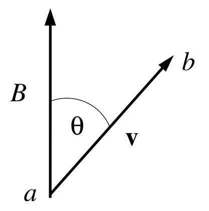


Fig. 1 For an internuclear vector **v** between two NMR-active nuclear spins *a* and *b*, and a vector representing the gradient of the magnetic field *B*, an RDC experiment detects the interaction between the nuclear spins. This interaction is dependent on the internuclear distance $r_{a,b}^3$ as well as the angle θ between vectors **v** and B.

The Tensor Formulation of the RDC Equation. A more convenient form of the RDC equation (1) for computing the geometric structure of a protein can be obtained after a series of algebraic manipulations [2]:

$$D = D_{\mathrm{max}} \mathbf{v}^{\mathrm{T}} \mathbf{S} \mathbf{v}, \tag{2}$$

in which D_{max} is the dipolar interaction constant, **v** is the unit internuclear vector, and **S** is the Saupe matrix [23] corresponding to the alignment medium used in the NMR experiment.

To derive Equation (2) from (1), a transformation to the *molecular coordinate frame* associated with the protein internuclear vector **v** is made [20, 21]. In such a coordinate frame, **v** is static, and the magnetic field vector, B, tumbles around **v**. The time and ensemble average over such tumbling of B is represented by the Saupe matrix, which we discuss later in this section.

Note that Equation (2) directly relates an RDC value to the orientation of **v** in the molecular coordinate frame. A sufficient number of such equations allows computation of all internuclear vector orientations in a protein with respect to the molecular coordinate frame. In the rest of the paper, by denoting $r = D/D_{max}$, we will work with the *normalized* RDC equation:

$$r = \mathbf{v}^{\mathrm{T}} \mathbf{S} \mathbf{v}. \tag{3}$$

Alignment Tensors. The Saupe matrix in Equations (2) and (3) is a 3×3 symmetric and traceless matrix. It contains 5 degrees of freedom, 3 of which correspond to a 3D rotation, and 2 are eigenvalues.

Each alignment tensor can be diagonalized as $\mathbf{S} = \mathbf{R}^{\mathrm{T}} \hat{\mathbf{S}} \mathbf{R}$, in which **R** is a 3D rotation matrix, and the traceless diagonal matrix $\hat{\mathbf{S}}$ has eigenvalues (S_{xx}, S_{yy}, S_{zz}), $S_{zz} = -S_{xx} - S_{yy}$. Then, the RDC equation (3) becomes

$$r = S_{xx}x^2 + S_{yy}y^2 + S_{zz}z^2, \tag{4}$$

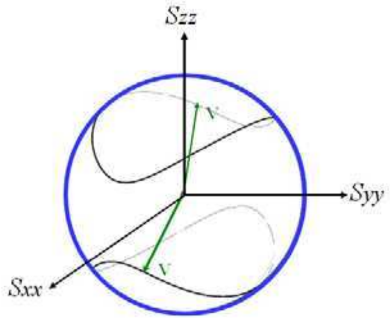


Fig. 2 The RDC sphero-quartic curves in the principal order frame for the alignment tensor (S_{xx}, S_{yy}, S_{zz}). The unit internuclear vector **v** that satisfies the RDC equation (4) is constrained to the sphero-quartic curves on the unit sphere.

in which $\mathbf{Rv} = (x, y, z)$. The coordinate frame that diagonalizes the Saupe matrix is called the *principal order frame (POF)* of the alignment medium.

To design exact algorithms, we consider algebraic representations of solutions. Figure 2 shows the solutions to the RDC equation for (x, y, z) as *sphero-quartic* curves on a sphere in the principal order frame. These curves are the intersection of the unit sphere $x^2 + y^2 + z^2 = 1$ and a hyperboloid representing the RDC equation (4).

Protein Geometry and Assumptions on Dynamics. A protein is modeled as a collection of peptide planes. Each peptide plane contains the bond vectors: C^{α}-C', C'-N, C'-O, N-H^{N} and N-C^{α}. Peptide planes are joined through the bond vectors N-C^{α} and C^{α}-C', with two torsional degrees of freedom. *Dihedral angles*, ϕ and ψ, parametrize these joints (see Figure 3). The bond vectors C^{α}-H^{α} and C^{α}-C^{β} do not belong to peptide planes and form a fixed tetrahedral geometry with both N-C^{α} and C^{α}-C'. Often, it is convenient to divide a protein into *residues*. Most of the residues (excluding glycine and proline with different geometry) contain consecutive atoms H, N, C^{α}, H^{α}, C^{β}, C' and O, as shown in Figure 3. The *backbone* of a protein is a sequence of repeating C^{α}, C', and N atoms.

There exist NMR techniques to measure RDCs on the C^{α}-H^{α}, C^{α}-C', C^{α}-C^{β}, N-H^{N}, and C'-N bond vectors as well as on C'-H^{N} internuclear vector.

Crucial assumptions about the *dynamics* of the protein are often necessary to solve for the large number of unknown variables in RDC equations. Initially, neither internuclear vectors $\mathbf{v}$ nor Saupe matrices $\mathbf{S}$ are known in the structure determination process. The assumption that the protein fluctuates in a small ensemble about one principal mode, that is, the protein is more or less rigid, leads to a simplified dynamics model in which the gradient vector B of the magnetic field is assumed to tumble similarly with respect to each internuclear vector. We work with this model, because it is applicable to many proteins. However, when the protein is known to be

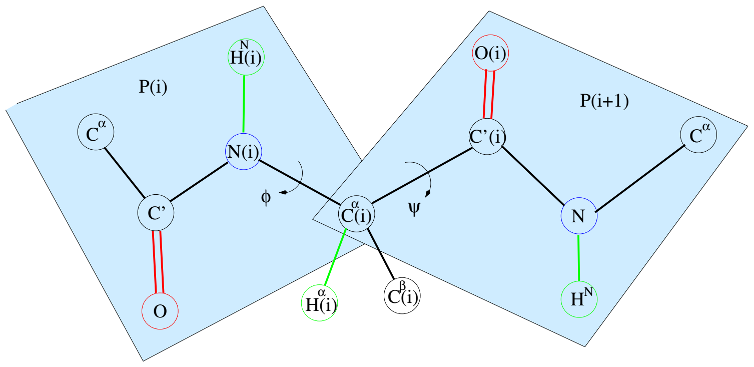


Fig. 3 Protein geometry. Two peptide planes P(i) and P(i+1) and the atoms of the corresponding residue i are shown. Two torsional degrees of freedom of the backbone are represented by dihedral angles ϕ and ψ.

Table 1 A ϕ-defining RDC is used to compute the backbone dihedral ϕ, and a ψ-defining RDC is used to compute the backbone dihedral ψ.

ϕ-defining	C^{α}-H^{α}, C^{α}-C', C^{α}-C^{β}
ψ-defining	N-H^N, C'-N, C'-H^N

more flexible in solution, additional parameters representing this flexibility must be integrated into the model.

3 The Protein Structure Determination Problem

The protein structure determination problem considered in this paper is: given sparse RDC data for a protein, determine the alignment tensor(s) and dihedral angles of the backbone that produce the best fit to the experimental data as well as represent a valid protein model.

Sparse RDC Data. We consider the case of experimental data, containing as few as two RDCs per residue in one or two media. To ease the presentation of our methods, we differentiate between RDC measurements based on their location within a residue. Denote RDC measurements on internuclear vectors C^{α}-H^{α}, C^{α}-C', and C^{α}-C^{β} as *ϕ-defining* RDCs, and N-H^N, C'-N, and C'-H^N as *ψ-defining* RDCs (Table 1). In this paper we design exact algorithms for the following types of RDC data: (a) One ϕ-defining RDC and one ψ-defining RDC per residue in one or two media; (b) Multiple ϕ-defining RDCs and multiple ψ-defining RDCs per residue in one or more media.

In general, it may not be possible to record a complete set of such RDC data for the entire protein. When we describe our methods in Section 4, we assume that all of the RDC values are present for a *protein fragment*. We describe how we deal with some cases of incomplete data in Section 5.

Data fit. The data fit function that we use is RDC root mean square deviation (RMSD). Denote $\mathbf{S}_j, j = 1, \dots, m$, all of the alignment tensors involved, and $\phi_i, \psi_i, i = 1, \dots n$, the dihedral angles for residues 1 through n in the protein backbone. The RDC RMSD for given alignment tensors and dihedral angles is computed using the equation:

$$\sigma(\{\mathbf{S}_j\}_{j=1}^m, \{\phi_i, \psi_i\}_{i=1}^n) = \sqrt{\frac{1}{l}\sum_{k=1}^{l}(r_k^b - r_k^e)^2}, \quad (5)$$

in which l is the total number of RDCs for all residues, r_k^e is the experimental RDC, and r_k^b is the RDC value back-computed from the alignment tensors and the structure defined by the dihedral angles computed using Equation (3).

Validation. To ensure that the solution structure is biologically meaningful, we validate it according to two criteria: Ramachandran regions [22], and van der Waals packing [24].

Next we proceed to the methods which solve the protein structure determination problem defined in this section.

4 Methods

The key to our methods is the use of explicit representation of the protein kinematics incorporated into the RDC equations.

At this point, it is useful to introduce several notations. Without loss of generality, we choose the principal order frame of $\mathbf{S}_1$ (POF$_1$) as the global coordinate frame. Within this coordinate frame, $\mathbf{S}_1$ is diagonal, with eigenvalues $S_{1,xx}$, $S_{1,yy}$, and $(-S_{1,xx} - S_{1,yy})$. We denote the diagonal components of any other alignment tensor, $\mathbf{S}_j$, as $S_{j,xx}$ and $S_{j,yy}$. $\mathbf{R}_{j,O}$ denotes the orientation of the principle order frame of alignment tensor $\mathbf{S}_j$ in POF$_1$.

Within a protein fragment, there will be several coordinate frames associated with different internuclear vectors in the portion, and related to each other through the protein kinematics. Our algorithms keep representations of these frames in POF$_1$.

Consider a coordinate frame defined at the peptide plane P_i with z-axis along the bond vector $\mathrm{N}(i) \to \mathrm{H}^{\mathrm{N}}(i)$ of residue i, in which the notation $a \to b$ means a vector from the nucleus a to the nucleus b. The y-axis is on the peptide plane i and the angle between the y-axis and the bond vector $\mathrm{N}(i) \to \mathrm{C}^{\alpha}(i)$ is $29.14°$ as described in [20]. The x-axis is defined based on right-handedness. Let $\mathbf{R}_{i,P}$ denote the orientation (rotation matrix) of P_i with respect to POF$_1$. Then, $\mathbf{R}_{1,P}$ denotes the relative rotation matrix between the coordinate system defined at the first peptide plane of the current protein portion and the principal order frame of the alignment tensor $\mathbf{S}_1$. We call it the *orientation of the first peptide plane*.

Our methods follow the following framework. First, the diagonal and rotational components of the alignment tensors and the orientation of the first peptide plane, $\mathbf{R}_{1,P}$, are estimated using methods described in Section 5. Next, the branch-and-bound tree search algorithm shown in Figure 4 is called, which computes the dihedral angles of the protein portion backbone. The tree encodes all the solutions to the system of RDC equations together with the kinematic equations that relate consecutive internuclear vectors in the protein backbone. Depending on the types of RDC data, the branch-and-bound search performs different computations. We cover these differences in Sections 4.1 and 4.2.

4.1 Exact Solutions for Peptide Plane Orientations from One ϕ-Defining RDC and One ψ-Defining RDC

In this section, consider the case of experimental RDC data that only contains one ϕ-defining and one ψ-defining RDC measurement per residue. This covers, for example, the case of having RDC measurements for C^{α}-H^{α}, and N-H^{N} in one medium. It also covers having C^{α}-H^{α} RDC in one medium and N-H^{N} RDC in second medium. Next, we describe the inductive step of the branch-and-bound tree search algorithm in Figure 4.

```
BRANCH-AND-BOUND ($\mathbf{S}_1,\ldots,\mathbf{S}_m$, $\mathbf{R}_{i,P}$)
Input: Global orientation of the coordinate frame for the current internuclear vector,
the RDC data for the next internuclear vector, and the alignment tensor(s)
Output: Those dihedral angles that represent valid protein structures and
best fit to the experimental data
 Branch:  Use methods from Sections 4.1 and 4.2 to solve RDC equations.
          Each solution is a dihedral that represents a child node.
 Bound:   Prune invalid children nodes based on:
          RDC RMSD, Ramachandran regions, and van der Waals packing
 Recurse: Compute global orientation of the coordinate frame for each valid
          child node, $\mathbf{R}_{i+1,P}$, call BRANCH-AND-BOUND ($\mathbf{S}_1,\ldots\mathbf{S}_m$, $\mathbf{R}_{i+1,P}$)
```

Fig. 4 The outline of the branch-and-bound tree search algorithm. The tree encodes all the solutions to the system of RDC equations together with the kinematic equations that relate consecutive internuclear vectors in the protein backbone. The algorithm systematically searches through these solutions and prunes those that do not satisfy the bound conditions. Different types of RDC data require different branch-and-bound criteria, which we cover in Sections 4.1 and 4.2.

The algorithm uses $\mathbf{R}_{i,P}$ to derive $\mathbf{R}_{i+1,P}$ inductively after it computes the backbone dihedral angles ϕ_i and ψ_i. $\mathbf{R}_{i+1,P}$ is then used to compute the dihedrals of the $(i+1)^{st}$ peptide plane. The angles ϕ_i and ψ_i are computed using the following propositions.

Proposition 1. *Given the diagonalized alignment tensor components $S_{1,xx}$ and $S_{1,yy}$, the orientation of the i^{th} peptide plane $\mathbf{R}_{i,P}$, and a ϕ-defining RDC, r, for the corresponding internuclear vector, $\mathbf{v}$, there exist at most 4 possible values of the dihedral angle ϕ_i that satisfy the RDC. The possible values of ϕ_i can be computed exactly and in closed form by solving a quartic equation.*

Proof. Let the unit vector $\mathbf{v}_0 = (0,0,1)^T$ represent the N-H$^{\mathrm{N}}$ bond vector of residue i in the local coordinate frame defined on the peptide plane P_i. Let $\mathbf{v} = (x,y,z)^T$ denote the internuclear vector for the ϕ-defining RDC for residue i in the principal order frame. We can write the forward kinematics relation between $\mathbf{v}$ and $\mathbf{v}_0$ as

$$\mathbf{v} = \mathbf{R}_{i,P}\,\mathbf{R}_l\,\mathbf{R}_z(\phi_i)\,\mathbf{R}_r\,\mathbf{v}_0, \tag{6}$$

in which $\mathbf{R}_l$ and $\mathbf{R}_r$ are constant rotation matrices that describe the kinematic relationship between $\mathbf{v}$ and $\mathbf{v}_0$. $\mathbf{R}_z(\phi_i)$ is the rotation about the z-axis by ϕ_i.

Let c and s denote $\cos\phi_i$ and $\sin\phi_i$, respectively. Using this while expanding Equation (6) we have

$$x = A_0 + A_1 c + A_2 s,\ y = B_0 + B_1 c + B_2 s,\ z = C_0 + C_1 c + C_2 s, \tag{7}$$

in which A_i, B_i, C_i for $0 \le i \le 2$ are constants. Using Equation (7) in the RDC equation (4) and simplifying we have

$$K_0 + K_1 c + K_2 s + K_3 cs + K_4 c^2 + K_5 s^2 = 0, \tag{8}$$

in which K_i, $0 \le i \le 5$ are constants. Using half-angle substitutions

$$u = \tan(\frac{\phi_i}{2}),\ c = \frac{1-u^2}{1+u^2},\ \text{and}\ s = \frac{2u}{1+u^2} \tag{9}$$

in Equation (8) we obtain

$$g(u) = L_0 + L_1 u + L_2 u^2 + L_3 u^3 + L_4 u^4 = 0, \tag{10}$$

in which L_i, $0 \le i \le 4$ are constants. Equation (10) is a quartic equation which can be solved exactly and in closed form. Let $\{u_1, u_2, u_3, u_4\}$ denote the set of four real solutions (at most) of Equation (10). For each u_i the corresponding dihedral angle ϕ_i can be computed using Eq. (9). □

Proposition 2. *Given the diagonalized alignment tensor components $S_{1,xx}$ and $S_{1,yy}$, the orientation of the i^{th} peptide plane $\mathbf{R}_{i,P}$, the dihedral ϕ_i, and a ψ-defining RDC, r, for the corresponding internuclear vector, $\mathbf{v}'$, on the peptide plane P_{i+1}, there exist at most 4 possible values of the dihedral angle ψ_i that satisfy the RDC. The possible values of ψ_i can be computed exactly and in closed form by solving a quartic equation.*

Proof. After representing the internuclear vector $\mathbf{v}'$ through $\mathbf{v}_0$ using protein kinematics:

$$\mathbf{v}' = \mathbf{R}_{i,P}\,\mathbf{R}_l\,\mathbf{R}_z(\phi_i)\,\mathbf{R}_r\,\mathbf{R}'_l\,\mathbf{R}_z(\psi_i)\,\mathbf{R}'_r\,\mathbf{v}_0, \tag{11}$$

the proof is similar to that in Proposition 1, since the value of ϕ_i is known. □

Proposition 3. *Given the diagonalized alignment tensor components, the orientation of the i^{th} peptide plane $\mathbf{R}_{i,P}$, a ϕ-defining RDC and a ψ-defining RDC for ϕ_i and ψ_i, respectively, there exist at most 16 orientations,$\mathbf{R}_{i+1,P}$, of the peptide plane P_{i+1} that satisfy the RDCs.*

Proof. This follows from the direct application of Propositions 1 and 2. □

Proposition 4. *Given the diagonalized alignment tensor components $S_{1,xx}$ and $S_{1,yy}$ for medium 1, $S_{2,xx}$ and $S_{2,yy}$ for medium 2, a relative rotation matrix $\mathbf{R}_{2,O}$, the orientation of the i^{th} peptide plane $\mathbf{R}_{i,P}$, a ϕ-defining RDC in medium 1 and a ψ-defining RDC in medium 2 for ϕ_i and ψ_i, respectively, there exist at most 16 orientations, $\mathbf{R}_{i+1,P}$, of the peptide plane P_{i+1} that satisfy the RDCs, which can be computed exactly and in closed form by solving two quartic equations.*

Proof. It follows the proof of Proposition 3, once the transformation to the principal order frame of medium 2 is made by $v' = \mathbf{R}_{2,O} v$ to compute the value of ψ_i. □

4.2 Exact Minima for Peptide Plane Orientations from Multiple ϕ-Defining RDCs and Multiple ψ-Defining RDCs

Now, consider the case when additional RDC data has been collected, and more than one ϕ and ψ-defining RDC measurements are available in one or more media. This covers, for example, the case of having RDCs for C^{α}-H^{α}, C^{α}-C', N-H^{N}, and C'-N in one medium. It also covers the case of having C^{α}-H^{α} and N-H^{N} RDCs in two media. The inductive step of the tree search in Figure 4 is performed using the following propositions.

Proposition 5. *Given the diagonalized alignment tensor components $S_{j,xx}$ and $S_{j,yy}$, the rotations between principal order frames, $\mathbf{R}_{j,O}$, the orientation of the i^{th} peptide plane $\mathbf{R}_{i,P}$, and multiple ϕ-defining RDC for the corresponding internuclear vector* $\mathbf{v}$ *of residue i, the global minimum of the RDC RMSD function for* $\mathbf{v}$ *can be computed exactly. There exist at most 4 possible values of the dihedral angle ϕ_i that minimize the RDC RMSD function, and such values of ϕ_i can be computed exactly.*

Proof. Let l be the number of RDC equations available for the internuclear vector $\mathbf{v}$. The RDC RMSD function for $\mathbf{v}$ is a univariate function of ϕ:

$$\sigma(\phi) = \sqrt{\frac{1}{l}\sum_{k=1}^{l}(r_k^b - r_k^e)^2}, \tag{12}$$

in which r_k^b is the back computed RDC value, $r_k^b = \mathbf{v}^T\mathbf{S}_j\mathbf{v}$, for the appropriate alignment medium $\mathbf{S}_j$. Similarly to the proof of Proposition 1, $\mathbf{v}$ can be represented as

$$\mathbf{v} = \mathbf{R}_{i,P}\,\mathbf{R}_l\,\mathbf{R}_z(\phi_i)\,\mathbf{R}_r\,\mathbf{v}_0. \tag{13}$$

After denoting $\cos\phi_i$ and $\sin\phi_i$ as c and s, respectively, Equation (13) becomes

$$x = A_0 + A_1c + A_2s,\; y = B_0 + B_1c + B_2s,\; z = C_0 + C_1c + C_2s, \tag{14}$$

in which A_i, B_i, C_i for $0 \le i \le 2$ are constants. Substituting x, y, and z into each RDC term of Equation (12) and using half-angle substitutions we obtain:

$$\sigma(u) = \sqrt{\frac{1}{2}\sum_{k=1}^{2}(g_k(u))^2}, \tag{15}$$

in which $g_k(u)$ are quartic polynomials for each medium k as in Equation 10.

Equation (15) defines a univariate function of u that can be minimized exactly, by finding zeroes of its derivative function. Let $\{u_1, u_2, u_3, u_4\}$ denote the set of four minima (at most) of Equation (15). For each u_i the corresponding dihedral angle ϕ_i can be computed using Eq. (9). □

Proposition 6. *Given the diagonalized alignment tensor components $S_{j,xx}$ and $S_{j,yy}$, the rotations between principal order frames, $\mathbf{R}_{j,O}$, the orientation of the i^{th} peptide* *plane $\mathbf{R}_{i,P}$, the dihedral ϕ_i, and multiple ψ-defining RDCs for the corresponding internuclear vector $\mathbf{v}'$ on peptide plane P_{i+1}, the global minima of the RDC RMSD function for $\mathbf{v}'$ can be computed exactly. There exist at most 4 possible values of the dihedral angle ψ_i that minimize the RDC RMSD function, and such values of ψ_i can be computed exactly.*

Proof. The proof is similar to that in Proposition 5, after the transformation as in Proposition 2 is used. □

Proposition 7. *Given the alignment tensors $\{\mathbf{S}_j\}_{j=1}^{m}$, the orientation of the peptide plane P_i, multiple ϕ-defining RDC and multiple ψ-defining RDC for ϕ_i and ψ_i, respectively, there exist at most 16 orientations of the peptide plane P_{i+1} with respect to P_i that minimize the RDC RMSD functions for each of the internuclear vectors.*

Proof. This follows from the direct application of Propositions 5 and 6. □

Note that the case of data described in this section allows comparing RDC RMSD for the branches of the search tree in Figure 4. This enables reducing the size of the search tree by pruning the branches based on RDC RMSD, which was not possible for the data described in Section 4.1 since RDC RMSD was always 0. Pruning based on Ramachandran regions and steric clashes alone is not always enough to compute the structures of protein loops. In Section 6 we show that if RDCs in second medium are measured, pruning based on RDC RMSD allows computation of loops.

5 Alignment Tensors, Orientation of the First Peptide Plane, and Packing

In our current implementation we estimate alignment tensor(s) similarly to [21, 20], by using singular value decomposition (SVD) [25] method to fit experimental RDC data to the corresponding vectors of an α-helix with ideal geometry. After that, the alignment tensor(s) are iteratively refined by using the computed helix structures by our exact algorithms. Once the values of the alignment tensor(s) are estimated, other fragments of the protein are computed using these values.

We can use uniform samples over rotation matrices to obtain the orientation of the first peptide plane in a protein fragment that result in structures with the best fit to the RDC data. For certain types of RDC data, however, the orientation of the first peptide plane in a fragment can be computed analytically.

The resulting running time complexity of our algorithms is linear, and the analysis is similar to [21]. As described in [21], we use a divide-and-conquer approach in which the protein is first divided into $O(n)$ fragments of constant length (typically 5-14 residues) based on their secondary structure type (α-helix, β-sheet, loop), and then our algorithms from Section 4 are applied to determine the orientations and conformations of these fragments. In contrast to [21], in which an algebraic geometry approach for finding the structure that minimizes the RDC fit for various RDC data was described, in this paper we have presented algorithms that achieve the same goal, but are simple and practical to implement.

In some cases reliable computation of the structure of certain fragments in a protein is not possible from sparse RDC alone. This may happen due to missing RDCs for certain residues. It may also happen due to the large size of the set of possible solutions returned by methods above. To overcome this problem, we use a sparse set of distance restraints (NOEs) to assemble the fragments. Our packing method [30] considers all possible discrete translations of the fragments over a three dimensional grid (within a parametrized resolution) that satisfy these sparse NOEs.

We also incorporate sparse unambiguous NOEs to pack β-sheets. We use rotamers from the Richardsons' Rotamer Library [49], and model the side-chain NOEs to pack the strands while they are being computed using the methods we described in the previous sections. A composite scoring scheme is used as a bound criterion in the tree search is based on a combination of (1) RDC RMSD, (2) RMS dihedral deviation from an ideal strand, (2) hydrogen bond score, i.e., a combination of RMS deviation of proton-acceptor distance and RMS deviation of donor-proton-acceptor angle violation, (3) score from the steric checker, and (4) NOE RMSD.

However, when a sufficient number of RDCs is measured for a protein, the divide-and-conquer approach is applied to the neighboring fragments sequentially, and, therefore, packing of the fragments does not require NOEs.

6 Results

We implemented our algorithms in a software package called RDC-ANALYTIC. Table 2 shows the results of the application of RDC-ANALYTIC on datasets for human ubiquitin (PDB id: 1ubq [26], DNA-damage-inducible protein I (DinI, PDB id: 1ghh [27]), Z-Domain of Staphylococcal Protein A (SpA, PDB id: 1q2n [28]), and the third IgG-binding domain of Protein G (GB3, PDB id: 1p7e [29]). For these proteins, the experimental NMR data has been taken from the Biological Magnetic Resonance Data Bank (BMRB). For ubiquitin, our program estimates the alignment tensors for different sets of RDCs and computes the helix (Ile23-Lys33) conformations. The results show that the corresponding alignment tensor components computed from different sets of RDCs in one medium or two media agree fairly well with those computed from ubiquitin NMR structure (PDB id: 1d3z). As shown in Table 2, the backbone RMSDs of the helices compared to the X-ray structure (PDB id: 1ubq) and NMR reference structure (PDB id: 1d3z) are small. The global folds of ubiquitin, DinI and SpA computed from different sets of RDCs and sparse sets of NOEs are shown in Figure 5. The results are summarized in Table 2. For ubiquitin and DinI we used sparse sets of NOEs which involve only amide and methyl protons obtainable from ^1H-^{13}C-ILV methyl labeling. We also used C^{α}-C' and N-H^N RDCs. For ubiquitin, the backbone RMSDs between the structures computed our algorithm and the reference structures (Model 1 of 1d3z, and 1ubq) is < 1.28 Å. For, DinI we computed the backbone fold using C^{α}-C' and N-H^N RDCs. Compared to the reference structure (Model 1 of 1ghh) the backbone RMSD was 1.11 Å.

For SpA we performed three runs of our program. In the first two runs, we used C^{α}-H^{α} and N-H^N RDCs and selected different sets of parameters. For the first run,

Table 2 Results of RDC-ANALYTIC**.** (a) Experimental RDC data for ubiquitin (PDB id: 1d3z), DinI (PDB id: 1ghh), SpA (PDB id: 1q2n), and GB3 (PDB id: 1p7e) are taken from the Biological Magnetic Resonance Data Bank (BMRB). The SSE backbones are computed for different combinations of RDCs in one or two media. If RDC measurements in two media are collected for a bond vector, we denote it by 2x in the table (e.g. 2xC^{α}-H^{α}). For ubiquitin the computed SSEs are compared with both the X-ray structure (PDB id: 1ubq) and the NMR structure (PDB id: 1d3z, Model 1). For DinI, SpA and GB3, since only the NMR structures are available, we compare our SSEs with Model 1 of the respective ensemble. (b) Simulated RDCs obtainted from the reference structure are used. (c) Simultaneous structure computation and assembly of β-strands into β-sheets of ubiquitin and DinI are done using 13 and 6 NOEs, respectively, which involve only amide and methyl protons obtainable using ^{1}H-^{13}C-ILV methyl labeling. (d) For ubiquitin, DinI and SpA we used 5, 10 and 10 C^{α}-C^{α} distance restraints, respectively, to pack the SSEs and obtain the maximum likelihood backbone folds. (e) Simulated RDCs from 1p7e Model 1 are used only for the missing RDC values in the experimental data.

Protein	RDCs[a] used & RMSD (Hz)	Alignment Tensor(s) (S_{yy}, S_{zz})	Backbone RMSD (Å) vs. X-ray/NMR structure
Ubiquitin[c,d] α:23-33	C^{α}-H^{α}: 1.11, N-H^{N}: 0.740	15.230, 24.657	1.276
β:2-7, 12-17, 41-45, 65-70	C^{α}-C': 0.129, N-H^{N}: 0.603	14.219, 25.490	1.172
DinI[c,d] α:18-32,58-72 β:2-8, 39-44, 49-53	C^{α}-C': 0.483, N-H^{N}: 1.203	10.347, 33.459	1.111
SpA[d]	(run1) C^{α}-H^{α}: 0.458,N-H^{N}: 2.11	8.008, 23.063	1.063
α:8-17, 24-36, 41-54	(run2) C^{α}-H^{α}: 0.678,N-H^{N}: 0.543	8.146, 24.261	1.577
	[b](run3)C^{α}-C': 1.237,N-H^{N}: 1.049	7.676, 22.961	0.834
Ubiquitin α:25-31	2xC^{α}-H^{α}: 0.93, 2xN-H^{N}: 0.32	16.9, 23.2; 7.0, 52.4	0.403
loop:54-58	2xC^{α}-H^{α}: 2.2, 2xN-H^{N}: 0.7	16.9, 23.2; 7.0, 52.4	0.409
loop:59-64	2xC^{α}-H^{α}: 1.9, 2xN-H^{N}: 1.2	16.9, 23.2; 7.0, 52.4	0.652
loop/β:64-70	2xC^{α}-H^{α}: 3.1, 2xN-H^{N}: 1.2	16.9, 23.2; 7.0, 52.4	0.49
β:2-7	2xC^{α}-H^{α}: 2.6, 2xN-H^{N}: 1.4	16.9, 23.2; 7.0, 52.4	0.64
β:11-17	2xC^{α}-H^{α}: 2.6, 2xN-H^{N}: 1.5	16.9, 23.2; 7.0, 52.4	0.50
β:41-55	2xC^{α}-H^{α}: 2.2, 2xN-H^{N}: 0.8	16.9, 23.2; 7.0, 52.4	0.44
GB3[(e)] α/loop:23-39	2xC^{α}-H^{α}: 1.7, 2xN-H^{N}: 1.6	47.0, 19.2; 23.8, 12.6	0.35
loop/β:39-51	2xC^{α}-H^{α}: 0.9, 2xN-H^{N}: 1.3	16.9, 23.2; 7.0, 52.4	0.49
loop/β:51-55	2xC^{α}-H^{α}: 0.7, 2xN-H^{N}: 0.5	16.9, 23.2; 7.0, 52.4	0.54

the backbone fold computed by our algorithm is within 1.1 Å of the reference structure (Model 1 of 1q2n). For the second run, we used a narrow sampling interval for N-H^{N} RDCs, and as a result the N-H^{N} RDC RMSD of the structure computed was better than that for the structure computed in the first run. However, when we packed the SSEs computed from the second run and then compared the resulting backbone fold with the reference structure (Model 1 of 1q2n), the backbone RMSD was 1.58 Å, slightly higher than that from the first run. We found that when the first two helices (Glu24-Asp36, Ser41-Ala54) are compared with the reference structure, the backbone RMSD was 0.72 Å. The N-H^{N} RDCs are missing for Glu8 and Gln9 that define the first two residues of the helix Glu8-Leu17, which probably led to somewhat poor conformation for this helix. We then simulated the C^{α}-C' and N-H^{N} RDCs using 1q2n Model 1. Using these simulated RDCs, we computed the global fold of SpA during the third run. When compared with the reference structure, the backbone RMSD was 0.83 Å.

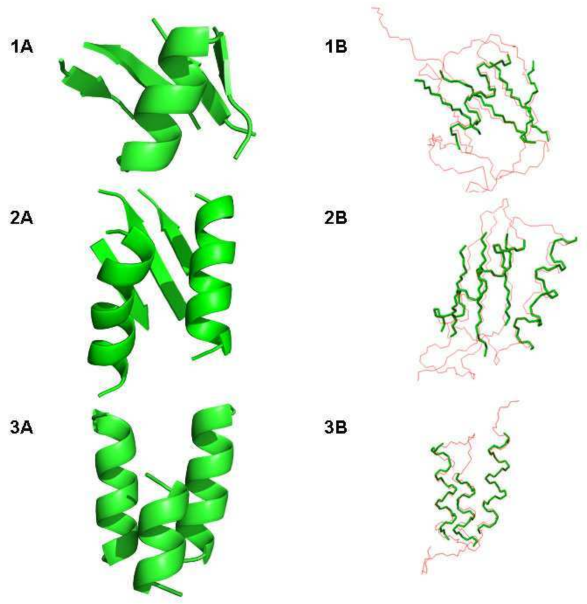


Fig. 5 The global folds of ubiquitin (1A), DinI (2A) and SpA (3A), computed by RDC-ANALYTIC, using C^{α}-C' and N-H^{N} RDCs and a sparse set of NOEs. For ubiquitin and DinI experimental C^{α}-C' and N-H^{N} RDCs are used. For SpA simulated C^{α}-C' and N-H^{N} RDCs are used. (1B) Overlay of the ubiquitin global fold (green) computed by RDC-ANALYTIC with the X-ray structure (red). The backbone RMSD is 1.17 Å. (2B) Overlay of the global fold of DinI (green) computed by RDC-ANALYTIC with the Model 1 (red) of the reference structure (PDB id: 1ghh). The backbone RMSD is 1.11 Å. (3B) The global fold of SpA computed by RDC-ANALYTIC is overlaid on the Model 1(red) of the reference structure (PDB id: 1q2n). The backbone RMSD is 0.83 Å.

For both ubiquitin and GB3 we applied RDC-ANALYTIC to compute portions (including helices, loops and β-strands) from C^{α}-H^{α} and N-H^{N} RDCs in two media. The results for portions of ubiquitin are reported in Table 2. The resulting overlay of the residues 23-55 of GB3 backbone (red, green, and black) computed by RDC-ANALYTIC with the NMR reference structure (PDB id:1p7e Model 1 [29]) is shown in Figure 6. Since the experimental data for GB3 is not complete (28 out of 132 C^{α}-H^{α} and N-H^{N} RDCs in two media are missing for for residues 23-55), we simulated the missing RDCs using the NMR reference structure. We then computed several consecutive portions of the protein in a divide-and-conquer fashion. The overall backbone RMSD with the reference structure was 1.47 Å.

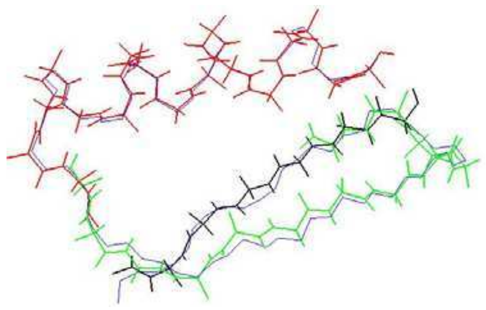


Fig. 6 Overlay of the residues 23-55 of GB3 backbone (red, green, and black) computed by RDC-ANALYTIC with the NMR reference structure PDB id:1p7e Model 1 [29] (blue). Several portions of the protein were computed in a divide-and-conquer fashion. The backbone RMSD of the portions are the following residues: 23-39(red) - 0.35 Å, 39-51(green) - 0.49 Å, 51-55(black) - 0.54 Å. The overall backbone RMSD with the reference structure is 1.47 Å.

The above tests on both real NMR data and simulated data demonstrate the capability of RDC-ANALYTIC to determine high-quality backbone fold. As part of our future work, we plan to apply our algorithms to determine backbone structures from NMR data collected for larger proteins.

7 Conclusions

We developed algorithms for protein structure determination using residual dipolar couplings collected by solution-state NMR. Our algorithms take into account different aspects of the structure determination process, such as the time and cost of NMR data acquisition, the types of NMR data and their information content, the algorithmic complexity of extracting the 3D structure, and the accuracy of the obtained solution. By using RDCs, we reduce the algorithmic complexity of the structure determination problem to linear time. To reduce the cost of NMR data acquisition, our methods use sparse RDC data, specifically, as little as two RDC measurements per residue in a protein. We develop exact algorithms to compute analytic solutions that optimally fit the NMR data producing high quality structures.

The key to our algorithms is the explicit representations of the RDC equations together with the protein kinematics. Geometrically, this representation results in algebraic curves on a unit sphere that may or may not intersect, depending on the type and number of RDC measurements collected for a single internuclear vector in a residue. Our algorithms find the points on the unit sphere located closest to all of the algebraic curves exactly. These points correspond to the dihedral angles of the protein that optimally fit the RDC data.

We tested our algorithms on NMR data for several proteins: human ubiquitin, DinI, SpA, and GB3. Previous versions of our algorithms [30] (which were not exact for as many types of RDC data as the new algorithms presented in this paper) have been extensively tested on NMR datasets for different proteins, as well as used in a prospective study to solve the structure of the FF Domain 2 of human transcription elongation factor CA150 (PDB id: 2kiq [30]). We plan to do more extensive experimental tests on different NMR data using the algorithms proposed in this paper. We also plan to apply our algorithms to solve other new protein structures in our future work.

Acknowledgements. We thank the members of Donald, Zhou and Richardson labs for their assistance and helpful comments. This work is supported by the following grants from National Institutes of Health: R01 GM-65982 to B.R.D. and R01 GM-079376 to P.Z.

References

1. Coggins, B., Venters, R., Zhou, P.: Progr NMR Spectr. (2010)
2. Donald, B.R., Martin, J.: Progr. NMR Spectr. 55(2), 101–127 (2009)
3. Clore, G.M., Gronenborn, A.M., Tjandra, N.: J. Magnet Res. 131, 159–162 (1998)
4. Güntert, P.: Progr. NMR Spectr. 43, 105–125 (2003)
5. Mumenthaler, C., Güntert, P., Braun, W., Wüthrich, K.: J. Biomol. NMR 10(4), 351–362 (1997)
6. Gronwald, W., Moussa, S., Elsner, R., Jung, A., Ganslmeier, B., Trenner, J., Kremer, W., Neidig, K.-P., Kalbitzer, H.R.: J. Biomol. NMR 23, 271–287 (2002)
7. Kuszewski, J., Schwieters, C.D., Garrett, D.S., Byrd, R.A., Tjandra, N., Clore, G.M.: J. Am. Chem. Soc. 126(20), 6258–6273 (2004)
8. Huang, Y.J., Tejero, R., Powers, R., Montelione, G.T.: Proteins: Structure Function and Bioinformatics 62(3), 587–603 (2006)
9. Delaglio, F., Kontaxis, G., Bax, A.: J. Am. Chem. Soc. 122, 2142–2143 (2000)
10. Andrec, M., Du, P., Levy, R.M.: J. Biomol. NMR 21(4), 335–347 (2001)
11. Rienstra, C.M., Tucker-Kellogg, L., Jaroniec, C.P., Hohwy, M., Reif, B., Mcmahon, M.T., Tidor, B., Lozano-Pérez, T., Griffin, R.G.: Proceedings of the National Academy of Sciences of the United States of America 99(16), 10260–10265 (2002)
12. Tian, F., Valafar, H., Prestegard, J.H.: J. Am. Chem. Soc. 123, 11791–11796 (2001)
13. Tolman, J.R., Flanagan, J.M., Kennedy, M.A., Prestegard, J.H.: Proceedings of the National Academy of Sciences USA 92, 9279–9283 (1995)
14. Tjandra, N., Bax, A.: Science 278, 1111–1114 (1997)
15. Brünger, A.T.: X-PLOR, version 3.1. A system for X-ray crystallography and NMR. Yale University Press, New Haven (1992)
16. Schwieters, C.D., Kuszewski, J.J., Tjandr, N., Clore, G.M.: J. Magnet. Res. 160, 65–73 (2003)
17. Rohl, C.A., Baker, D.: J. Am. Chem. Soc. 124, 2723–2729 (2002)
18. Hus, J.-C., Marion, D., Blackledge, M.: J. Am. Chem. Soc. 123, 1541–1542 (2001)
19. Giesen, A., Homans, S., Brown, J.: J. Biomol. NMR 25, 63–71 (2003)
20. Wang, L., Donald, B.R.: J. Biomol. NMR 29(3), 223–242 (2004)
21. Wang, L., Mettu, R.R., Donald, B.R.: J. Comp. Bio. 13(7), 1276–1288 (2006)
22. Lovell, S.C., Davis, I.W., Arendall, W.B., de Bakker, P.I., Word, J.M., Prisant, M.G., Richardson, J.S., Richardson, D.C.: Proteins: Structure, Function, and Genetics 50(3), 437–450 (2003)
23. Saupe, A.: Ang Chemie 7(2), 97–112 (1968)

24. Word, J.M., Lovell, S.C., Labean, T.H., Taylor, H.C., Zalis, M.E., Presley, B.K., Richardson, J.S., Richardson, D.C.: J. Mol. Bio. 285(4), 1711–1733 (1999)
25. Losonczi, J.A., Andrec, M., Fischer, M.W.F., Prestegard, J.H.: J. Magnet. Res. 138, 334–342 (1999)
26. Vijay-Kumar, S., Bugg, C.E., Cook, W.J.: J. Mol. Bio. 194, 531–544 (1987)
27. Ramirez, B.E., Voloshin, O.N., Camerini-Otero, R.D., Bax, A.: Protein Science 9, 2161–2169 (2000)
28. Zheng, D., Aramini, J.M., Montelione, G.T.: Protein Science 13, 549–554 (2004)
29. Ulmer, T., Ramirez, B., Delaglio, F., Bax, A.: J. Am. Chem. Soc. 125(13), 9179–9191 (2003)
30. Zeng, J., Boyles, J., Tripathy, C., Wang, L., Yan, A., Zhou, P., Donald, B.R.: J. Biomol. NMR 45(3), 265–281 (2009)

LQG-Based Planning, Sensing, and Control of Steerable Needles

Jur van den Berg, Sachin Patil, Ron Alterovitz, Pieter Abbeel, and Ken Goldberg

Abstract. This paper presents a technique for planning and controlling bevel-tip steerable needles towards a target location in 3-D anatomy under the guidance of partial, noisy sensor feedback. Our approach minimizes the probability that the needle intersects obstacles such as bones and sensitive organs by (1) explicitly taking into account motion uncertainty and sensor types, and (2) allowing for efficient optimization of sensor placement. We allow for needle trajectories of arbitrary curvature through duty-cycled spinning of the needle, which is conjectured to make a needle path small-time locally "trackable" [13]. This enables us to use LQG control to guide the needle along the path. For a given path and sensor placement, we show that a priori probability distributions of the needle state can be estimated in advance. Our approach then plans a set of candidate paths and sensor placements and selects the pair for which the estimated uncertainty is least likely to cause intersections with obstacles. We demonstrate the performance of our approach in a modeled prostate cancer treatment environment.

1 Introduction

We consider the problem of planning, sensing, and controlling a bevel-tip steerable needle towards a target in 3-D anatomy with obstacles, such as sensitive and impenetrable tissue. Needles are used in many forms of medical diagnosis and treatment, and accurately reaching a specific target is required in many procedures such as tissue biopsies and placement of radioactive seeds for cancer treatment. Bevel-tip steerable needles are asymmetric-tip, flexible needles that move along trajectories

Jur van den Berg · Sachin Patil · Ron Alterovitz
University of North Carolina at Chapel Hill
e-mail: {berg,sachin,ron}@cs.unc.edu

Pieter Abbeel · Ken Goldberg
University of California at Berkeley
e-mail: {pabbeel,goldberg}@berkeley.edu

of constant curvature κ_0 when pushed forward [27]. The direction of motion can be changed by reorienting the bevel tip through twisting of the needle at its base. This allows for steering the needle around anatomical obstacles towards previously inaccessible targets, and allows for significantly reducing patient trauma by avoiding the puncturing of sensitive tissues.

Planning and controlling the motion of a steerable needle is a challenging problem. A steerable needle is controlled from its base through only insertion and twisting, and we do not allow retractions and re-insertions as that results in excessive tissue damage. As such, a steerable needle is a highly underactuated non-holonomic system. In fact, the needle is not small-time locally controllable, and a natural needle path (with curvature κ_0) is not small-time locally "trackable" (i.e., the deviation with respect to the path is not small-time locally controllable) [13]. In addition, the motion of the needle is subject to uncertainty due to tissue inhomogeneity, tissue deformation, needle torsion, etc. [7, 23]. To address this issue, we follow the suggestion of Kallem [13] that a path with a smaller curvature $0 \leq \kappa < \kappa_0$ *is* small-time locally trackable and that this can be achieved using duty-cycled spinning of the needle during insertion. This would enable us to use feedback control to guide the needle along a pre-planned path. Our experiments suggest that this is indeed the case. The sensor feedback, however, may be noisy and partial as current medical imaging technology does not allow for measuring the full state of the needle tip (the imaging resolution is often too low to infer its orientation, for instance [12], and often only provides planar views in real-time feedback situations).

Our objective is to compute a sensor placement and a needle path to the target location, such that the path's execution using LQG control has a minimal probability of intersecting obstacles in the anatomy, given a stochastic model of the motion and sensing uncertainty. Our approach is as follows. First, we build on work of [16] to encapsulate the (high-frequency) duty cycled spinning of the needle in a higher-level kinematic model that allows direct control of the curvature of the needle motion. We then derive an LQG controller (consisting of a Kalman filter for state estimation and an LQR control policy) for the extended kinematic model to optimally guide the needle along a given path. Based on the sensor placement, we can compute *in advance* the a priori probability distributions of the state of the needle along the path [25]. From these distributions and a geometric model of the anatomical obstacles, we can quickly compute the probability that the needle will intersect obstacles. Our method then plans a set of candidate paths using a variant of the RRT algorithm [14] and samples a set of feasible sensor placements, and then selects the pair for which this probability of obstacle intersection is minimal.

The type and placement of sensor(s) can have a big influence on which path is optimal (see Fig. 1). For example, if a sensor obtains a 2-D projection along the x-axis of the 3-D position of the needle tip, there will be more uncertainty in the x-coordinate of the needle state than in the y- and z-coordinates. If we then have to steer the needle through a passage that is narrow in either the y- or the x-direction, our path planner will prefer the passage for which the probability of intersecting obstacles is less, which is the one that is narrower in the y-direction. Given a set of candidate paths and the space of possible sensor placements, our approach will

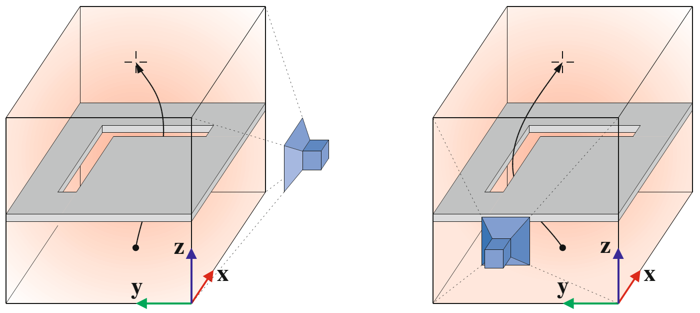


Fig. 1 Two examples of sensor placement, in which (left) only the x- and z-coordinate and (right) only the y- and z-coordinate of the needle-tip are measured by the imaging device (blue). Different paths will be optimal even as the obstacles (grey) and target location (cross) are the same.

choose a needle path and find an axis along which a projection is obtained that minimizes the probability of collisions for the chosen needle path. We demonstrate the performance of our approach in modeled prostate cancer treatment environments with simulated uncertainty for different examples of sensor models.

The remainder of this paper is organized as follows. We discuss previous work in Section 2, and review the kinematic model of a steerable needle in Section 3. In Section 4 we derive an LQG controller to optimally guide the needle along a given path. In Section 5, we show how to estimate the probability of obstacle intersection for a given path and sensor placement, and present our path and sensor planner. We present simulation results in Section 6 and conclude in Section 7.

2 Related Work

A significant body of previous work exists on planning and/or controlling bevel-tip steerable needles. A kinematic model for a steerable needle generalizing a unicycle model was introduced in [27], and has been used by most subsequent work on needle steering, including this paper. In [16], it was shown that in addition to the insertion and rotation speed, the curvature of the needle path can be controlled through duty-cycled spinning of the needle during insertion.

2-D planners that address motion uncertainty have been presented in [2, 3], which optimize a Markov decision process (MDP) over a discretized state space to provide feedback control assuming full state observation. The approach was extended in [22] and integrated with imaging feedback. In [12], a feedback controller is presented to stabilize the needle in a plane. Tissue deformation is taken into account in the planner of [1], which optimizes a path using 2-D FEM simulation of soft tissue.

Needle path planners for 3-D environments with obstacles have been proposed in [8, 9], based on optimization and inverse kinematics, respectively. Rapidly-exploring random trees (RRTs) have been used in [29, 30, 20] to explore the entire

3D space of feasible paths. These approaches do not address issues such as uncertainty in motion and sensing. A diffusion-based planner was introduced in [18], but it does not take into account obstacles or sensor types. A feedback controller for 3-D needle steering was presented in [11], and proved robust against motion deviation and sensing noise, even for a greedy control policy. The approach is not able to guide the needle along a prescribed path and does not address obstacle avoidance.

The approach we present in this paper extends these previous works and applies to 3-D environments with obstacles, takes into account motion and sensing uncertainty, and does not require discretization of the state-space. Our approach to needle steering is the first that specifically addresses the sensing capabilities and its effect on optimizing the path.

3 Needle Kinematics

We base our motion model of a bevel-tip steerable needle on the idealized kinematic model of [27], using the nomenclature of [8], in which the needle state is represented by a rigid body transformation. This model assumes that the motion of the needle is fully determined by the motion of the tip, which is beveled such that it follows a perfect arc of curvature κ_0 when pushed forward, independent of insertion speed and tissue properties. The model further assumes that the needle is flexurally flexible (it bends to follow the needle tip), but axially and torsionally stiff, such that the insertion and twisting of the needle at its base is directly transmitted to its tip.

The state of the needle tip can be described by a 3-D rigid body transformation relative to a world coordinate frame, which is compactly represented by a 4×4 transformation matrix $X \in SE(3)$;

$$X = \begin{bmatrix} R & \mathbf{p} \\ 0 & 1 \end{bmatrix}, \tag{1}$$

where $R \in SO(3)$ is a 3×3 rotation matrix describing the rigid body's orientation, and $\mathbf{p} \in \mathbb{R}^3$ a vector describing the rigid body's position.

The kinematics of a rigid body, i.e. the evolution of its state over time, can generally be described a follows. Let $\mathbf{v} \in \mathbb{R}^3$ and $\mathbf{w} \in \mathbb{R}^3$ be the vectors of instantaneous linear and angular velocities, respectively, expressed in the local coordinate frame attached to the rigid body. Then (using notation $'$ to refer to the time derivative):

$$X' = XU, \quad U = \begin{bmatrix} [\mathbf{w}] & \mathbf{v} \\ 0 & 0 \end{bmatrix}, \tag{2}$$

where 4×4 matrix $U \in se(3)$ is the twist of the rigid body. The notation $[\mathbf{a}]$ for a vector $\mathbf{a} = [a_x \;\; a_y \;\; a_z]^T \in \mathbb{R}^3$ refers to the following 3×3 skew-symmetric matrix:

$$[\mathbf{a}] = \begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix}. \tag{3}$$

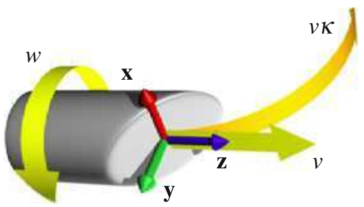


Fig. 2 Local coordinate frame attached to the needle tip. Its kinematics are shown by yellow arrows. Figure reproduced with permission from [9].

When the twist U is constant, the state of the rigid body at time t given the initial state at time 0 is computed explicitly by integrating Eq. (2), for which a closed form expression exists [8]:

$$X(t) = X(0)\exp(tU). \tag{4}$$

For the steerable needle, the local coordinate frame is rigidly attached to the tip of the needle such that the z-axis points along the forward direction of the needle, and the y-axis points along the bevel direction (see Fig. 2). The motion of the needle is determined by two control inputs: the linear forward speed (i.e. the speed with which the needle is inserted), denoted v, and the bevel orientation speed (i.e. the speed with which the needle is twisted at its base), denoted ω. Hence, the linear and angular velocities of the needle tip given the control inputs $v \geq 0$ and ω are $\mathbf{v} = \begin{bmatrix} 0 & 0 & v \end{bmatrix}^T$ and $\mathbf{w} = \begin{bmatrix} v\kappa_0 & 0 & \omega \end{bmatrix}^T$, respectively, where κ_0 is the (fixed) curvature of the arc the needle follows through the tissue.

This needle model is constrained by the fact that the curvature κ_0 of the needle paths is fixed. In recent work, however, Minhas et al. [16] show that by performing duty cycled spinning of the needle during insertion, any curvature κ of the needle motion between 0 and κ_0 can be approximated. The greatest degree of curvature ($\kappa = \kappa_0$) is achieved by no spin at all, while a straight trajectory ($\kappa = 0$) is created by constantly spinning the needle at a high (infinite) rate during insertion. Any trajectory in between these two extrema can be approximated by duty cycling the spinning in a spin-stop-spin-stop fashion. Longer stop intervals create steeper curvature of the needle, and shorter stop intervals create straighter trajectories. To be precise, the proportion $0 \leq \alpha \leq 1$ of the time spent in spin intervals to approximate a curvature of $0 \leq \kappa \leq \kappa_0$ is given by:

$$\alpha = 1 - \kappa/\kappa_0. \tag{5}$$

Let the needle perform a 2π rotation each spin interval, and let the spin intervals be of a small and constant duration δ. Then, the period of one spin-stop cycle is δ/α. In order to incorporate the duty cycling into the kinematic model of the needle, the control input $\omega(t)$ (parameterized by time t) is adjusted to:

$$\omega(t) = \begin{cases} 2\pi/\delta + w & \text{if } j \leq t/(\delta/\alpha) < j + \alpha \qquad \text{(spin)} \\ w & \text{if } j + \alpha \leq t/(\delta/\alpha) < j + 1, \qquad \text{(stop)} \end{cases} \tag{6}$$

for any $j \in \mathbb{Z}$, where w is the higher-level control of the speed with which the needle is rotated at its base (at the lower level, it is augmented with rapid 2π rotations).

By taking the limit $\delta \to 0$, the resulting high-level kinematic model is similar to the low-level kinematic model described above, with the difference that ω is replaced by w, and the curvature κ is added to the set of control inputs. The high-level twist U given high-level control inputs v, w and κ is then given by:

$$U = \begin{bmatrix} [\mathbf{w}] & \mathbf{v} \\ 0 & 1 \end{bmatrix}, \quad \mathbf{v} = \begin{bmatrix} 0 & 0 & v \end{bmatrix}^T, \quad \mathbf{w} = \begin{bmatrix} v\kappa & 0 & w \end{bmatrix}^T. \tag{7}$$

To account for the uncertainty the motion of the needle is subject to due to tissue inhomogeneity, tissue deformation, needle torsion, etc., we augment the model by assuming that the twist is corrupted by additive noise $\tilde{U}$ drawn from a zero-mean Gaussian distribution with variance M:

$$\tilde{U} = \begin{bmatrix} [\tilde{\mathbf{w}}] & \tilde{\mathbf{v}} \\ 0 & 1 \end{bmatrix}, \quad \begin{bmatrix} \tilde{\mathbf{v}} \\ \tilde{\mathbf{w}} \end{bmatrix} = \mathbf{m} \sim \mathcal{N}(\mathbf{0}, M), \tag{8}$$

The stochastic kinematics of the needle state X is then given by:

$$X' = X(U + \tilde{U}). \tag{9}$$

4 LQG Control for Steerable Needles

Let us be given a needle path Π consisting of states $\hat{X}$ and control input twists $\hat{U}$ formed from control inputs $\hat{v} > 0$, $\hat{w}$ and $0 \le \hat{\kappa} < \kappa_0$ (see Eq. (7)), such that

$$\hat{X}' = \hat{X}\hat{U}, \quad \hat{X} = \begin{bmatrix} \hat{R} & \hat{\mathbf{p}} \\ 0 & 1 \end{bmatrix}, \quad \hat{U} = \begin{bmatrix} [\hat{\mathbf{w}}] & \hat{\mathbf{v}} \\ 0 & 0 \end{bmatrix}. \tag{10}$$

That is, the path is consistent with the needle kinematics without noise, and as conjectured in [13], the path is small-time locally trackable since $0 \le \hat{\kappa} < \kappa_0$.

During control of the needle along the path Π, we can assume that we obtain potentially noisy and partial observations of the state as feedback from sensors, in order to compensate for unexpected needle motion. We assume that this feedback will be according to the following (general) sensor model:

$$\mathbf{z} = h(X, \mathbf{q}), \quad \mathbf{q} \sim \mathcal{N}(\mathbf{0}, Q), \tag{11}$$

where $\mathbf{z}$ is a vector of measurements that relates to the state X through function h, and $\mathbf{q}$ is the measurement noise drawn from a zero-mean Gaussian with variance Q.

To control the needle along the needle path Π, we use the LQG-controller, since it provides optimal control for linear Gaussian motion and sensor models with a quadratic cost function penalizing deviation from the path. The LQG controller uses a Kalman filter for state estimation in parallel with an LQR control policy. Since our motion and sensor model are non-linear, we approximate them with local linearizations around the path. This is reasonable as the needle is controlled to stay close to the path during execution.

For purposes of linearization, we will not directly control the state itself, but rather the *deviation* of the state with respect to the path. This is also convenient for dealing with the 3-D orientation in the needle state, which either has a redundant but internally constrained representation (e.g. a quaternion or rotation matrix) or a minimal but singularity-prone representation (e.g. Euler angles). Assuming that the deviation is small, the orientation deviation is "far away" from any singularities, and can hence safely be represented by three mutually unconstrained parameters. Whereas many other works use quaternions to represent rotations [10, 15, 24], we will linearize for the 4×4 matrix $X \in SE(3)$ (roughly following [4]), and derive a Kalman filter and LQR controller for the needle.

4.1 Model Linearization and Discretization

We define the state deviation $\bar{X}$ as the transformation between the path state $\hat{X}$ and the (unknown) true state X, and the twist deviation $\bar{U}$ as the difference between the true control input twist U and the control input twist $\hat{U}$ along the path:

$$\bar{X} = \begin{bmatrix} \bar{R} & \bar{\mathbf{p}} \\ 0 & 1 \end{bmatrix} = \hat{X}^{-1}X = \begin{bmatrix} \hat{R}^T R & \hat{R}^T(\mathbf{p}-\hat{\mathbf{p}}) \\ 0 & 1 \end{bmatrix}, \quad \bar{U} = \begin{bmatrix} [\bar{\mathbf{w}}] & \bar{\mathbf{v}} \\ 0 & 0 \end{bmatrix} = U - \hat{U}, \tag{12}$$

where the last equality follows from the fact that $\hat{X}^{-1} = \left[\begin{smallmatrix} \hat{R}^T & -\hat{R}^T\hat{\mathbf{p}} \\ 0 & 1 \end{smallmatrix}\right]$. The kinematics of the state deviation, i.e. its evolution over time, is given by

$$\begin{aligned} \bar{X}' &= \hat{X}^{-1}X' + (\hat{X}^{-1})'X = \hat{X}^{-1}X(U+\tilde{U}) - \hat{U}\hat{X}^{-1}X = \\ &= \bar{X}(U+\tilde{U}) - \hat{U}\bar{X} = \bar{X}(\hat{U}+\bar{U}+\tilde{U}) - \hat{U}\bar{X} = \\ &= \begin{bmatrix} \bar{R}[\hat{\mathbf{w}}+\bar{\mathbf{w}}+\tilde{\mathbf{w}}] - [\hat{\mathbf{w}}]\bar{R} & \bar{R}(\hat{\mathbf{v}}+\bar{\mathbf{v}}+\tilde{\mathbf{v}}) - [\hat{\mathbf{w}}]\bar{\mathbf{p}} - \hat{\mathbf{v}} \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} \bar{R}' & \bar{\mathbf{p}}' \\ 0 & 0 \end{bmatrix}, \end{aligned} \tag{13}$$

where the equalities follow from Eqs. (9) and (12), and the fact that $(\hat{X}^{-1})' = -\hat{U}\hat{X}^{-1}$ [21].

To get a non-redundant state vector, we represent the orientation deviation $\bar{R}$ as a rotation of angle $\|\bar{\mathbf{r}}\|$ about axis $\bar{\mathbf{r}} \in \mathbb{R}^3$. Assuming this deviation is sufficiently small, it is approximated well by the following first-order Taylor expansion:

$$\bar{R} = I + [\bar{\mathbf{r}}]. \tag{14}$$

By substituting Eq. (14) into Eq. (13), and ignoring all second-order error terms, we get to first order (using the identities $[\mathbf{a}]\mathbf{b} = \mathbf{a} \times \mathbf{b} = -[\mathbf{b}]\mathbf{a}$ and $[\mathbf{a}][\mathbf{b}] - [\mathbf{b}][\mathbf{a}] = [\mathbf{a} \times \mathbf{b}]$):

$$\bar{\mathbf{p}}' = [\bar{\mathbf{r}}]\hat{\mathbf{v}} + \bar{\mathbf{v}} + \tilde{\mathbf{v}} - [\hat{\mathbf{w}}]\bar{\mathbf{p}} = -[\hat{\mathbf{w}}]\bar{\mathbf{p}} - [\hat{\mathbf{v}}]\bar{\mathbf{r}} + \bar{\mathbf{v}} + \tilde{\mathbf{v}}, \tag{15}$$

$$[\bar{\mathbf{r}}]' = [\bar{\mathbf{r}}][\hat{\mathbf{w}}] + [\bar{\mathbf{w}} + \tilde{\mathbf{w}}] - [\hat{\mathbf{w}}][\bar{\mathbf{r}}] = [-[\hat{\mathbf{w}}]\bar{\mathbf{r}}] + [\bar{\mathbf{w}} + \tilde{\mathbf{w}}]. \tag{16}$$

Combining Eqs. (15) and (16), we get in matrix form:

$$\begin{bmatrix} \bar{\mathbf{p}}' \\ \bar{\mathbf{r}}' \end{bmatrix} = \begin{bmatrix} -[\hat{\mathbf{w}}] & -[\hat{\mathbf{v}}] \\ 0 & -[\hat{\mathbf{w}}] \end{bmatrix} \begin{bmatrix} \bar{\mathbf{p}} \\ \bar{\mathbf{r}} \end{bmatrix} + \begin{bmatrix} \bar{\mathbf{v}} \\ \bar{\mathbf{w}} \end{bmatrix} + \begin{bmatrix} \tilde{\mathbf{v}} \\ \tilde{\mathbf{w}} \end{bmatrix}, \tag{17}$$

which we can write as

$$\bar{\mathbf{x}}' = F\bar{\mathbf{x}} + G\bar{\mathbf{u}} + \mathbf{m}, \quad \mathbf{m} \sim \mathcal{N}(\mathbf{0}, M), \tag{18}$$

where $\bar{\mathbf{x}} = \left[\begin{smallmatrix} \bar{\mathbf{p}} \\ \bar{\mathbf{r}} \end{smallmatrix}\right]$, $F = \left[\begin{smallmatrix} -[\hat{\mathbf{w}}] & -[\hat{\mathbf{v}}] \\ 0 & -[\hat{\mathbf{w}}] \end{smallmatrix}\right]$, $\mathbf{m}$ is as defined in Eq. (8), and $\bar{\mathbf{u}}$ and G are defined as follows:

$$\bar{\mathbf{u}} = \begin{bmatrix} v - \hat{v} \\ w - \hat{w} \\ v\kappa - \hat{v}\hat{\kappa} \end{bmatrix}, \quad G = \begin{bmatrix} 0\,0\,1\,0\,0\,0 \\ 0\,0\,0\,0\,0\,1 \\ 0\,0\,0\,1\,0\,0 \end{bmatrix}^T. \tag{19}$$

Let us discretize time into stages of duration τ and assume that the control inputs v, w, κ and $\hat{v}$, $\hat{w}$, $\hat{\kappa}$ and variance M are constant for the duration of each stage k. The path Π then consists of a series of states and control input twists $(\hat{X}_0, \hat{U}_0, \ldots, \hat{X}_\ell, \hat{U}_\ell)$, where ℓ is the number of stages of the path, such that

$$\hat{X}_{k+1} = \hat{X}_k \exp(\tau \hat{U}_k). \tag{20}$$

We can then integrate Eq. (18) to get [15]:

$$\bar{\mathbf{x}}_{k+1} = A_k \bar{\mathbf{x}}_k + B_k \bar{\mathbf{u}}_k + \mathbf{n}_k, \quad \mathbf{n}_k \sim \mathcal{N}(\mathbf{0}, N_k), \tag{21}$$

where

$$A_k = e^{\tau F_k}, \quad B_k = \int_0^\tau e^{(\tau - t)F_k} G \, \mathrm{d}t, \quad N_k = \int_0^\tau e^{(\tau - t)F_k} M_k e^{(\tau - t)F_k^T} \, \mathrm{d}t. \tag{22}$$

Eq. (21) is the linearized and discretized motion model of the deviation of the needle state from the path.

The sensor model (see Eq. (11)) is discretized by assuming that in each stage k we obtain a measurement $\mathbf{z}_k$. To relate the state deviation vector $\bar{\mathbf{x}}$ (as opposed to the state matrix X as in Eq. (11)) to a measurement $\mathbf{z}$, we define (note that $\bar{\mathbf{p}}$ and $\bar{\mathbf{r}}$ are part of $\bar{\mathbf{x}}$):

$$\bar{h}_k(\bar{\mathbf{x}}, \mathbf{q}) = h\left(\hat{X}_k \begin{bmatrix} I + [\bar{\mathbf{r}}] & \bar{\mathbf{p}} \\ 0 & 1 \end{bmatrix}, \mathbf{q}\right), \tag{23}$$

where we use the fact that $X = \hat{X}\bar{X}$ (see Eq. (12)) to reconstruct the state matrix X from the state deviation vector $\bar{\mathbf{x}}$. Linearizing $\bar{h}_k$ around the path Π gives

$$\bar{\mathbf{z}}_k = H_k \bar{\mathbf{x}}_k + W_k \mathbf{q}_k, \quad \mathbf{q}_k \sim \mathcal{N}(\mathbf{0}, Q_k), \tag{24}$$

where

$$\bar{\mathbf{z}}_k = \mathbf{z}_k - \bar{h}_k(\mathbf{0},\mathbf{0}), \quad H_k = \frac{\partial \bar{h}_k}{\partial \bar{\mathbf{x}}}(\mathbf{0},\mathbf{0}), \quad W_k = \frac{\partial \bar{h}_k}{\partial \mathbf{q}}(\mathbf{0},\mathbf{0}). \tag{25}$$

Eq. (24) is the linearized and discretized sensor model of the deviation of the needle state from the path.

4.2 Kalman Filter and LQR Controller

Eqs. (21) and (24) form a standard linear Gaussian model, of which $\bar{\mathbf{x}}$ is the state, $\bar{\mathbf{u}}$ the control input and $\bar{\mathbf{z}}$ the measurement. The Kalman filter keeps track of the estimate $\hat{\mathbf{x}}$ and variance P of the true state $\bar{\mathbf{x}}$ during control. It continually performs two steps; a process update to propagate the applied control input $\bar{\mathbf{u}}$, and a measurement update to incorporate the obtained measurement $\bar{\mathbf{z}}$:

Process update step:

$$\hat{\mathbf{x}}_k^- = A_{k-1}\hat{\mathbf{x}}_{k-1}, \tag{26}$$

$$P_k^- = A_{k-1}P_{k-1}A_{k-1}^T + N_{k-1}. \tag{27}$$

Measurement update step:

$$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + W_k Q_k W_k^T)^{-1}, \tag{28}$$

$$\hat{\mathbf{x}}_k = K_k(\bar{\mathbf{z}}_k - H_k\hat{\mathbf{x}}_k^-), \tag{29}$$

$$P_k = (I - K_k H_k)P_k^-. \tag{30}$$

The Kalman-gain matrices K_k can be computed in advance (i.e. before execution) given the initial variance P_0, without knowledge of the actual control inputs $\bar{\mathbf{u}}$ and measurements $\bar{\mathbf{z}}$. We refer the reader to [28] for additional details.

The LQR controller provides optimal control for a motion model of the type given by Eq. (21) [5]. The optimal control inputs $\bar{\mathbf{u}}_k$ are found by minimizing the cost function

$$\min_{\bar{\mathbf{u}}}\Big(\sum_{k=0}^{\ell}(\bar{\mathbf{x}}_k^T C\bar{\mathbf{x}}_k + \bar{\mathbf{u}}_k^T D\bar{\mathbf{u}}_k)\Big), \tag{31}$$

which quadratically penalizes deviations from the path Π through given positive-definite weight matrices C and D. Matrix C specifies the cost for deviating from the planned path, while D specifies the cost for deviating from the planned control input. Penalizing the magnitude of $\bar{\mathbf{u}}$ is reasonable as the linearized motion model is only valid when $\bar{\mathbf{u}}$ is small.

Solving Eq. (31) gives the control policy $\bar{\mathbf{u}}_k = L_k\bar{\mathbf{x}}_k$, for feedback matrices L_k that are pre-computed using a standard recursive procedure (for $0 \le k < \ell$) [5]. As the true state $\bar{\mathbf{x}}_k$ is unknown, the estimate $\hat{\mathbf{x}}_k$ of the state which is obtained from the Kalman filter is used to determine the control input $\bar{\mathbf{u}}_k$ at each stage k during execution of the path. Hence, the control policy is $\bar{\mathbf{u}}_k = L_k\hat{\mathbf{x}}_k$. We refer the reader to [5, 25] for additional details.

The actual control inputs v_k, w_k and κ_k that are applied to the needle are found using Eq. (19), given $\bar{\mathbf{u}}_k$ and the control inputs $\hat{v}_k$, $\hat{w}_k$ and $\hat{\kappa}_k$ of path Π. After application of the control input, the Kalman filter produces the estimate of the next state from which in turn a new control input is determined. This cycle repeats until the execution of the path is complete.

5 Optimal Path and Sensor Planning

The first objective is to plan a needle path towards a target location $\mathbf{g}$ inside a workspace $\mathscr{W} \subset \mathbb{R}^3$. For simplicity, we assume that the workspace is the rectangular region $[0, x_{\max}) \times [0, y_{\max}) \times [0, z_{\max})$, and that the needle can enter the workspace at any point in the plane $z = 0$. Further, the workspace may contain obstacles defined by a region $O \subset \mathscr{W}$ that models impenetrable or sensitive tissue. The second objective is to select a placement of the sensor that will provide feedback during control.

The quality measure of a path Π and a sensor placement is the probability that the needle will intersect obstacles when the path is executed using LQG control. We will first discuss how to compute this probability for a given path Π and a given sensor model, and then discuss how we plan a set of candidate paths and sensor placements from which we select the pair that minimizes the probability of intersecting obstacles during control.

5.1 Obstacle Intersection Probability along a Path

Given the LQG controller for a path Π and sensor model h (see Section 4), we can analyze in advance how the true state $\bar{\mathbf{x}}_t$ and the estimated state $\hat{\mathbf{x}}_t$ will evolve during control as functions of each other. The evolution of the true state $\bar{\mathbf{x}}_t$ is dependent on the estimated state through the LQR control policy and the evolution of the estimated state $\hat{\mathbf{x}}_t$ is dependent on the true state through the measurement obtained in the Kalman filter. This gives the following equation in matrix form (see [25] for more details):

$$\begin{bmatrix} \bar{\mathbf{x}}_{k+1} \\ \hat{\mathbf{x}}_{k+1} \end{bmatrix} = \begin{bmatrix} A_k & B_k L_k \\ K_{k+1} H_{k+1} A_k & A_k + B_k L_k - K_{k+1} H_{k+1} A_k \end{bmatrix} \begin{bmatrix} \bar{\mathbf{x}}_k \\ \hat{\mathbf{x}}_k \end{bmatrix} + \tag{32}$$
$$\begin{bmatrix} I & 0 \\ K_{k+1} H_{k+1} & K_{k+1} W_{k+1} \end{bmatrix} \begin{bmatrix} \mathbf{n}_k \\ \mathbf{q}_{k+1} \end{bmatrix}, \quad \begin{bmatrix} \mathbf{n}_k \\ \mathbf{q}_{k+1} \end{bmatrix} \sim \mathscr{N}(\mathbf{0}, \begin{bmatrix} N_k & 0 \\ 0 & Q_{k+1} \end{bmatrix}),$$

which we write shorthand –for the appropriate definitions of $\mathbf{y}_k$, Y_k, V_k, $\mathbf{s}_k$ and Z_k– as (note that Y_k, V_k and Z_k can all be computed in advance for a given a path Π):

$$\mathbf{y}_{k+1} = Y_k \mathbf{y}_k + V_k \mathbf{s}_k, \quad \mathbf{s}_k \sim \mathscr{N}(\mathbf{0}, Z_k). \tag{33}$$

From this, we can compute the mean $\hat{\mathbf{y}}_k$ and the variance Σ_k of $\mathbf{y}_k = \left[\begin{smallmatrix} \bar{\mathbf{x}}_t \\ \hat{\mathbf{x}}_t \end{smallmatrix}\right]$ for any stage k of the execution of the path:

$$\hat{\mathbf{y}}_{k+1} = Y_k \hat{\mathbf{y}}_k, \qquad \hat{\mathbf{y}}_0 = \mathbf{0}, \tag{34}$$

$$\Sigma_{k+1} = Y_k \Sigma_k Y_k^T + V_k Z_k V_k^T, \qquad \Sigma_0 = \begin{bmatrix} P_0 & 0 \\ 0 & 0 \end{bmatrix}. \tag{35}$$

Note that the mean $\hat{\mathbf{y}}_k$ is zero for all stages t. Hence, $\left[\begin{smallmatrix} \bar{\mathbf{x}}_k \\ \hat{\mathbf{x}}_k \end{smallmatrix}\right] \sim \mathcal{N}(\mathbf{0}, \Sigma_k)$.

Given these a priori distributions of the state deviation, we can compute the probability that the needle will intersect an obstacle during the execution of path Π. Let $\Sigma_k^{\mathbf{p}}$ be the variance of the position deviation $\bar{\mathbf{p}}_k$, which is the upper-left 3×3-submatrix of the 12×12-matrix Σ_k (note that $\bar{\mathbf{x}} = \left[\begin{smallmatrix} \bar{\mathbf{p}} \\ \bar{\mathbf{r}} \end{smallmatrix}\right]$). As $\bar{\mathbf{p}} = \hat{R}^T(\mathbf{p} - \hat{\mathbf{p}})$ (see Eq. (12)), we have that $\mathbf{p} = \hat{\mathbf{p}} + \hat{R}\bar{\mathbf{p}}$, so the a priori distribution of the position $\mathbf{p}_k$ of the needle tip at stage k along path Π is given by $\mathcal{N}(\hat{\mathbf{p}}_k, \hat{R}_k \Sigma_k^{\mathbf{p}} \hat{R}_k^T)$. Hence, the probability p_k that the needle intersects the obstacle region at stage k along path Π is computed as:

$$p_k = \int_O \frac{\exp(-\frac{1}{2}(\mathbf{b} - \hat{\mathbf{p}}_k)^T (\hat{R}_k P_k^{\mathbf{p}} \hat{R}_k^T)^{-1} (\mathbf{b} - \hat{\mathbf{p}}_k))}{\sqrt{8\pi^3 \det(\hat{R}_k P_k^{\mathbf{p}} \hat{R}_k^T)}} d\mathbf{b}, \tag{36}$$

which is the integral over the obstacle region O of the probability density function of Gaussian $\mathcal{N}(\hat{\mathbf{p}}_k, \hat{R}_k P_k^{\mathbf{p}} \hat{R}_k^T)$.

Instead of computing the probabilities p_k exactly, we will use an approximation that can be computed efficiently. To this end, we look at the maximum factor by which the ellipsoid of one standard deviation of the a priori probability distribution can be scaled such that it does not intersect an obstacle. Let this factor be c_k for stage k along the path. For a multivariate Gaussian distribution of dimension n, the probability that a sample is outside c_k standard deviations is given by

$$\hat{p}_k = 1 - \Gamma(n/2, c_k^2/2), \tag{37}$$

where Γ is the regularized Gamma function [17]. It provides a (conservative) upper bound of the true probability of intersecting obstacles at stage k.

The value of c_k for stage k is efficiently computed by using a collision-checker capable of performing distance calculations and linear transformations on the obstacle geometry, such as SOLID [26]. Let E_k be a matrix such that $E_k E_k^T = \Sigma_k^{\mathbf{p}}$. The set of positions within one standard deviation is then an ellipsoid centered at the mean $\hat{\mathbf{p}}_k$ obtained by transforming a unit sphere by E_k. Transforming the environment by E_k^{-1} (such that the uncertainty ellipsoid becomes a unit sphere), and calculating the Euclidean distance between $\hat{\mathbf{p}}_k$ and the nearest obstacle in the transformed environment gives the value of c_k for stage k, from which the approximate probability $\hat{p}_k$ of intersecting obstacles at stage k can be computed using Eq. (37).

Assuming (somewhat opportunistically) that the probabilities $\hat{p}_k$ are independent, it follows that the probability $p(\Pi)$ that the needle intersects the obstacle region anywhere along path Π is given by $p(\Pi) = 1 - \prod_{k=0}^{\ell}(1 - \hat{p}_k)$.

5.2 Planning a Needle Path and Sensor Placement

To plan an optimal pair of a needle path and sensor placement, we (randomly) generate large sets of possible needle paths and sensor placements, and evaluate the probability of intersecting obstacles for each pair.

To generate a large set of (random) needle paths we use the RRT (random rapidly-exploring tree) algorithm [14], as it is known to create trees of paths that uniformly cover the space and handle kinematically constrained systems, such as steerable needles, well. As the target location is a specific point $\mathbf{g}$, and the entry location can be anywhere in a pre-defined entry zone, it is convenient to plan backwards from the target location. Backward kinematics are identical to forward kinematics, except that the (forward) control inputs $\hat{v}$, $\hat{w}$, and $\hat{\kappa}$ are integrated over a negative time-step. As the actual path traced out by the needle only depends on the ratio w/v and not on the values of the individual terms [8], we set $\hat{v} = 1$ cm/s and only vary $\hat{w} \in [-w_{\max}, w_{\max}]$ and $\hat{\kappa} \in [0, \kappa_0)$. We will not describe the RRT-planner in detail here, but refer to [20, 29, 30] for details on RRT implementations for steerable needles. All the paths in the resulting tree that reach the entry zone are valid.

Let $\mathscr{S}$ be the space of sensor placements, and let the sensor model $h(X, \mathbf{q})$ corresponding to a placement $s \in \mathscr{S}$ be denoted $h_s(X, \mathbf{q})$. We generate a large set of placements by random sampling from $\mathscr{S}$. We iterate over all valid paths contained in the RRT-tree and all placements sampled from $\mathscr{S}$, and select the pair for which the needle has minimal probability of intersecting obstacles as computed above.

6 Simulation Results

We experimentally evaluate our approach using simulations of the computed candidate paths using LQG control with simulated process and measurement noise. In our experiments, we use an anatomical model of the human male pelvic region to simulate needle insertion in tissue for delivering radioactive doses to targets within the prostate region for cancer treatments. We first show that the needle is controllable along a candidate path. We then show the effect the sensor can have on the optimal path, and how the sensor placement can be optimized for a given problem.

We implemented the system in C++ and tested it on a 3.33 Ghz 4-core Intel® i7™ PC. All experiments utilized only a single core for computation, but our approach could be parallelized over multiple cores to yield significant speedups. In our experiments, we model the needle motion and noise using the following parameters: $w_{\max} = 2\pi$ rad/s, $\tau = 0.1$ s, $\kappa = 0.2$ cm^{-1}, M is a diagonal matrix with 0.01 (cm/s)^2 for the position components and 0.05 (rad/s)^2 for the rotational components, and Q is a diagonal matrix with 0.05 cm^2 along the diagonal.

6.1 Needle Controllability

We first demonstrate the controllability of the needle along a candidate path using an LQG controller with artificially generated process and measurement noise. We

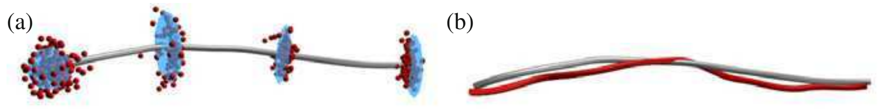


Fig. 3 (a) Given a candidate path (gray) from left to right, we illustrate samples (red spheres) from 100 simulated executions of the LQG controller and extended Kalman filter showing convergence to the path. The simulations included artificially generated process and measurement noise. The blue ellipsoids show the a priori distributions computed by the planner along the path. (b) A simulated example trajectory (shown in ref) using LQG control.

assume that the sensor can only measure the position **p** of the needle tip and not the orientation. Fig. 3a shows the samples obtained from 100 simulations of executions of the path using the LQG controller, demonstrating that the a priori probability distributions computed by the planner are close approximations of the true distribution of the states along the path and that the needle follows the candidate path closely. For an insertion length of 11.25 cm, the standard deviation of the distance to the target was found to be 0.17 cm.

To emulate uncertainties arising due to tissue heterogeneities, we applied spatially varying process noise sampled from Gaussian distributions with zero mean and variances up to three times the variance M. Following the same candidate path of length 11.25 cm, the standard deviation of the distance of the final needle-tip position to the target across 1000 simulation runs was 0.24 cm.

6.2 Effect of Sensor Placement on Optimal Path

We demonstrate the use of LQG to select plans that minimize the probability of failure and also show the effect of sensor placement on the optimal path. We performed these experiments using an anatomical model of the human pelvic region (see Fig. 4). As an example sensor model, we consider a 2-D image of the anatomy (for example, an x-ray or 2-D ultrasound image). The image is projected along the z-axis using an imaging device, from which we can only measure the x- and y-coordinate of the position $\mathbf{p} = [x, y, z]$ of the needle tip. This gives the following observation function h (note that $\mathbf{p}$ is part of X):

$$h(X, \mathbf{q}) = \begin{bmatrix} x \\ y \end{bmatrix} + \mathbf{q}, \quad \mathbf{q} \sim \mathcal{N}(\mathbf{0}, Q) \tag{38}$$

Fig. 4 (left) shows the optimal needle path within the RRT-tree for this sensor. The optimal path predominantly lies in the x-y plane to minimize the uncertainty along the viewing direction (z-axis). It took 41 seconds to generate a set of 1000 candidate paths and selecting an optimal path from this set required 4.6 seconds. Similarly, if we instead place the sensor such that it obtains 2-D images projected along the y-axis, the optimal path (shown in Fig. 4 (right)) is predominantly in the x-z plane.

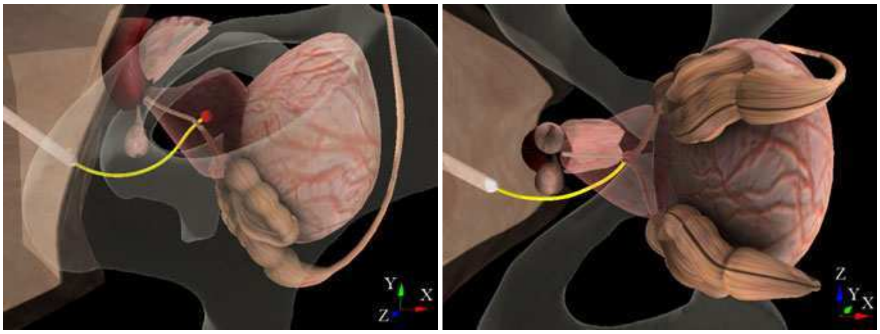


Fig. 4 Optimal needle paths in an anatomical environment modeling the human prostate and surrounding tissues. Two examples of a sensor placement, in which (left) only the x- and y-coordinate and (right) only the x- and z-coordinate of the needle-tip are measured by an imaging device by projecting the anatomy on the imaging plane (shown in the two views). The optimal path predominantly lies in the imaging plane to minimize uncertainty in the viewing direction.

To quantify the effect the sensor location has on the probability of success of a path, we compare the results of two paths assuming sensing along the z axis: the optimal path and a path that lies predominantly in the x-z plane. For each path, we estimated the "ground truth" probability of success by performing 1000 simulated executions using the LQG controller with artificially generated process and measurement noise. The path that is optimal according to our method, which lies predominantly in the x-y plane, has a probability of success of 89%. In contrast, the path that primarily lies in the x-z plane had a probability of success of only 44%, which is to be expected since there is greater uncertainty in needle pose along the viewing direction (z-axis).

6.3 Optimizing the Sensor Placement

In many clinical procedures involving bevel-tip steerable needles, the physician can select where to place an intra-operative sensing device for real-time feedback. We consider the case of an x-ray imaging sensor attached to a C-arm, a commonly used setup in operating rooms that allows the physician to rotate the sensor in a circle about the patient as shown in Fig. 5. The placement of the imaging device can be parameterized by angle θ relative to the horizontal axis of the patient. A 2-D image of the anatomy can be obtained by parallel projection along the viewing direction (along the radius of the arm). This gives the following observation function h:

$$h(X,\mathbf{q}) = \begin{bmatrix} x \\ y\cos\theta + z\sin\theta \end{bmatrix} + \mathbf{q}, \quad \mathbf{q} \sim \mathcal{N}(\mathbf{0}, Q) \tag{39}$$

To optimize sensor placement, we iterate over all paths computed by the RRT-based planner and all sampled sensor placements (obtained by varying the angle of rotation

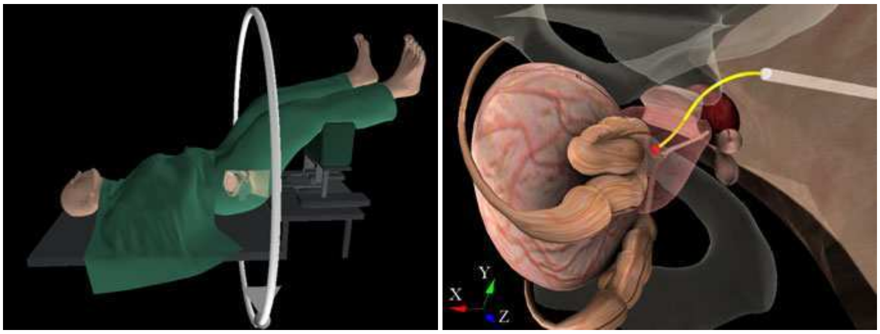


Fig. 5 Using an x-ray imager mounted on a rotating C-arm, it is possible to rotate the sensor about the horizontal axis along which the patient is positioned (left). The anatomy as viewed from the computed optimal sensor placement (right). The optimal path predominantly lies in the imaging plane to minimize uncertainty in the viewing direction.

about the horizontal axis θ in regular increments), and select the pair for which the needle has minimal probability of intersecting obstacles. Fig. 5 shows the optimal sensor placement for a given set of candidate paths and a 2-D view of the anatomy as visible from the imaging device. For a set of 1000 candidate paths and 36 possible sensor placements (obtained by discretizing over the interval $[0, \pi]$ in intervals of 5 degrees), our implementation took 185 seconds to compute the optimal pair over the set of candidate paths and sensor placements. It should be noted that with modern multi-core processors, this computation can be trivially parallelized to bring down the computation time within clinically acceptable limits.

7 Conclusion and Future Work

In this paper, we presented a technique for planning and controlling flexible steerable needles towards a target location in 3-D anatomy with obstacles such as bones and sensitive organs. Our approach minimizes the probability that the needle intersects obstacles by explicitly taking into account both needle motion uncertainty and the sensors used to obtain (noisy, partial) feedback on the needle state. We demonstrated how the sensor influences the optimal path and optimize over the set of candidate paths and feasible sensor placements to select the pair for which the estimated uncertainty is least likely to cause intersections with obstacles.

In our current implementation, the LQR controller does not bound the control inputs within permissible limits during feedback. This can be a problem when the control for needle curvature exceeds the attainable curvature of the needle. We plan to address this issue in future work. We also plan to incorporate uncertainty introduced by tissue deformation using a physically accurate FEM simulator [6].

We also plan to evaluate our new planning and control approach using a robotic device that steers a needle in phantom tissue as in prior 2-D experiments [22]. We will utilize the ability of this planner and controller to account for unexpected events that might occur due to tissue inhomogeneity, tissue deformation, and estimation errors in motion model parameters, as well as to optimize sensor placement to minimize the probability of intersecting an obstacle.

Acknowledgements. This research was supported in part by the National Science Foundation (NSF) under grants IIS-0905344 and IIS-0904672 and by the National Institute of Health (NIH) under grants R01-EB006435 and R21-EB011628. The authors also thank members of the needle steering community for their valuable input.

References

1. Alterovitz, R., Goldberg, K., Okamura, A.: Planning for steerable bevel-tip needle insertion through 2D soft tissue with obstacles. In: IEEE Int. Conf. on Robotics and Automation (2005)
2. Alterovitz, R., Branicky, M., Goldberg, K.: Motion planning under uncertainty for image-guided medical needle steering. Int. J. Robotics Research 27(11-12), 1361–1374 (2008)
3. Alterovitz, R., Siméon, T., Goldberg, K.: The stochastic motion roadmap: a sampling framework for planning with Markov motion uncertainty. In: Robotics: Science and Systems (2007)
4. Baldwin, G., Mahony, R., Trumph, J., Hamel, T., Cheviron, T.: Complementary filter design on the Special Euclidean group $SE(3)$. In: European Control Conference (2007)
5. Bertsekas, D.: Dynamic programming and optimal control. Athena Scientific (2001)
6. Chentanez, N., Alterovitz, R., Ritchie, D., Cho, L., Hauser, K., Goldberg, K., Shewchuk, J., O'Brien, J.F.: Interactive Simulation of Surgical Needle Insertion and Steering. ACM Trans. on Graphics 28(3), 88.1–88.10 (2009)
7. Cowan, N.J., Goldberg, K., Chirikjian, G.S., Fichtinger, G., Alterovitz, R., Reed, K.B., Kallem, V., Park, W., Misra, S., Okamura, A.M.: Robotic Needle Steering: Design, Modeling, Planning, and Image Guidance. In: Surgical Robotics - Systems, Applications, and Visions. Springer, Heidelberg (2010)
8. Duindam, V., Alterovitz, R., Sastry, S., Goldberg, K.: Screw-based motion planning for bevel-tip flexible needles in 3D environments with obstacles. In: IEEE Int. Conf. on Robotics and Automation (2008)
9. Duindam, V., Xu, J., Alterovitz, R., Sastry, S., Goldberg, K.: Three-dimensional motion planning algorithms for steerable needles using inverse kinematics. Int. J. Robotics Research 29(7), 789–800 (2010)
10. Hall, J., Knoebel, N., McLain, T.: Quaternion attitude estimation for miniature air vehicles using a multiplicative extended Kalman filter. In: IEEE/ION Position, Location and Navigation Symp. (2008)
11. Hauser, K., Alterovitz, R., Chentanez, N., Okamura, A., Goldberg, K.: Feedback control for steering needles through 3D deformable tissue using helical paths. In: Robotics: Science and Systems (2009)
12. Kallem, V., Cowan, N.: Image-guided control of flexible bevel-tip needles. In: IEEE Int. Conf. on Robotics and Automation (2007)
13. Kallem, V.: Vision-based control on lie groups with application to needle steering. PhD Thesis, Johns Hopkins University (2008)
14. LaValle, S., Kuffner, J.: Randomized kinodynamic planning. Int. J. Robotics Research 20(5), 378–400 (2001)

15. Lefferts, E., Markley, F., Shuster, M.: Kalman filtering for spacecraft attitude estimation. Journal of Guidance, Control and Dynamics 5(5), 417–429 (1982)
16. Minhas, D., Engh, J., Fenske, M., Riviere, C.: Modeling of needle steering via duty-cycled spinning. In: Int. Conf. IEEE Engineering in Medicine and Biology Society (2007)
17. Olver, F.W., Lozier, D.W., Boisvert, R.F., Clark, C.W.: NIST Handbook of Mathematical Functions. Cambridge University Press, Cambridge (2010)
18. Park, W., Kim, J., Zhou, Y., Cowan, N., Okamura, A., Chirikjian, G.: Diffusion-based motion planning for a nonholonomic flexible needle model. In: IEEE Int. Conf. on Robotics and Automation (2005)
19. Park, W., Wang, Y., Chirikjian, G.: Path planning for flexible needles using second order error propagation. In: Workshop on Algorithmic Foundations of Robotics (2008)
20. Patil, S., Alterovitz, R.: Interactive motion planning for steerable needles in 3D environments with obstacles. In: IEEE Int. Conf. on Biomedical Robotics and Biomechatronics (2010)
21. Petersen, K., Pedersen, M.: The Matrix Cookbook. Tech. Univ. of Denmark (2008)
22. Reed, K., Kallem, V., Alterovitz, R., Goldberg, K., Okamura, A., Cowan, N.: Integrated planning and image-guided control for planar needle steering. In: IEEE/RAS-EMBS Int. Conf. Biomedical Robotics and Biomechatronics (2008)
23. Reed, K., Okamura, A., Cowan, N.: Controlling robotically steered needle in the presence of torsional friction. In: IEEE Int. Conf. on Robotics and Automation (2009)
24. Trawny, N., Roumeliotis, S.: Indirect Kalman filter for 3D attitude estimation; a tutorial for quaternion algebra. Tech. report TR-2005-002, University of Minnesota (2005)
25. van den Berg, J., Abbeel, P., Goldberg, K.: LQG-MP: Optimized path planning for robots with motion uncertainty and imperfect state information. In: Robotics: Science and Systems (2010)
26. van den Bergen, G.: Collision detection in interactive 3D environments. Morgan Kaufmann Publishers, San Francisco (2004)
27. Webster, R., Kim, J., Cowan, N., Chirikjian, G., Okamura, A.: Nonholonomic modeling of needle steering. Int. J. Robotics Research 25(5-6), 509–525 (2006)
28. Welch, G., Bishop, G.: An introduction to the Kalman filter. Tech. Report TR 95-041, University of North Carolina at Chapel Hill (2006)
29. Xu, J., Duindam, V., Alterovitz, R., Goldberg, K.: Motion planning for steerable needles in 3D environments with obstacles using rapidly-exploring random trees and backchaining. In: IEEE Int. Conf. on Automation Science and Engineering (2008)
30. Xu, J., Duindam, V., Alterovitz, R., Pouliot, J., Cunha, A., Hsu, I., Goldberg, K.: Planning fireworks trajectories for steerable medical needles to reduce patient trauma. In: IEEE Int. Conf. on Intelligent Robots and Systems (2009)

Cyber Detectives: Determining When Robots or People Misbehave

Jingjin Yu and Steven M. LaValle

Abstract. This paper introduces a problem of validating the claimed behavior of an autonomous agent (human or robot) in an indoor environment containing one or more agents, against the observation history from a sparse network of simple, stationary sensors deployed in the same environment. Following principles of dynamic programming, we partition the decision problem into incremental search over a sequence of connectivity subgraphs induced by sensor recordings, which yields efficient algorithms for both single and multiple agent cases. In addition to immediate applicability towards security and forensics problems, the idea of behavior validation using external sensors complements design time model verification.

1 Introduction

One night, a crime was committed in an office building with complex interior structure. The next morning, a few suspects were identified but none of them would come forward. Instead, all of them provided seemingly convincing stories that excused them from being present at the crime scene. Unknown to the suspects, however, the building's security system, composed of a set of sensors with different capabilities, had made a sequence of recordings of passing people. Knowing that the criminal among the suspects was lying, can we use the sensor recordings to help solve the crime?

Similarly, in computer science, robotics, and control, a frequently encountered problem is verifying that an autonomous system, be it a program or a robot, is performing as designed. For example, a service robot may plan a path to clean office

Jingjin Yu
University of Illinois, Urbana-Champaign
e-mail: `jyu18@uiuc.edu`

Steven M. LaValle
University of Illinois, Urbana-Champaign
e-mail: `lavalle@uiuc.edu`

rooms one by one. Due to internal (sensor/actuator/computing units malfunctioning) or external factors (strong electromagnetic interference for example), the robot may mistake one room for another and fail to accomplish its task without knowing that it has failed. A robot or a system may also be compromised for malicious purposes, producing intentionally bogus records of its actual path to hide the fact. In such cases, it would be highly desirable if external monitoring could automatically determine that a robot has faltered.

In this paper, we introduce realistic abstractions of above problems and show that such formulations are computationally tractable. Specifically, one or more agents (robots or people) are assumed to move in an indoor environment, of which regions are monitored by external sensors (beam detectors and occupancy sensors). We assume that the agents are not aware of these sensors. From a story told by an agent, which is a sequences of places in the environment it has visited, and combined recordings of these sensors, we provide polynomial time algorithms (with respect to the complexity of the environment, the length of the story, as well as the length of the observation history) for the inference problem of whether the given story is consistent with the sensor recordings.

Our work takes inspirations from two active research topics in robotics and control. If one assumes that the behavior of a set of moving bodies is largely unknown, above problem becomes inferring various properties of these moving bodies with a network of simple sensors. Binary proximity sensors have been employed to estimate positions and velocities of a moving body using particle filters [3] and moving averages [17]. The performance limits of a binary proximity sensor network in tracking a single target are discussed and approached in [25], followed by an extension to the tracking of multiple targets [26]. The task of counting multiple targets is also studied under different assumptions [4, 15]. In these works, the sensor network's aggregate sensing range must cover the targets of interest at all times, which is much more difficult to implement than guarding critical regions of an environment. When only subsets of an environment are guarded, *word problems in groups* [12, 14] naturally arise. For the setup in which targets moving inside a 2D region are monitored with a set of detection beams, [28] characterizes possible target locations, target path reconstruction up to homotopy, and path winding numbers. In this domain, the surfacing of more interesting behaviors also induces an increase in complexity; few efficient algorithms exist. This prompts us to ponder: Can we do better if partial knowledge of a target's behavior is available? In viewing its resemblance to the questions asked in [3, 25, 28], our problem requires the design of a combinatorial filter, similar to those in [20, 29, 30]. These combinatorial filters are minimalist counterparts to widely known Bayesian filters [6, 9, 13, 21, 22, 27, 31].

On the other hand, if sensors external to moving bodies are ignored, one is left with the task of systematically verifying that the moving bodies do not have unexpected behaviors. Complex moving bodies such as robots are often modeled as hybrid systems. Existing verification techniques either address subclasses of hybrid systems or approximate reachable sets of such systems [2, 8, 10], because the problem of verifying a system with continuous state space and control input is generally undecidable [1]. In practice, this difficulty translates into the necessity of external measures to safeguard the unverified portion of a system. Alternatively, when high level task specifications can be coded as General Reactivity(1) formulas [23], the task of composing controllers into verifiably correct hybrid automata can be carried out automatically using linear temporal logic [11, 18]. Even for such provably correct designs, malfunction can still occur due to sensor/actuator/computer errors. Keeping these systems in check again requires monitoring with external sensors.

The main contributions of this paper are twofold. First, using a sparse network of simple sensors to validate the claimed behavior of an autonomous agent introduces a new methodology that complements traditional system verification techniques such as [2, 8, 10]. We believe this is a necessary approach given that most verification processes focus on high level abstractions of an autonomous system, which only models simplified, ideal behavior. Second, applying principles of dynamic programming [5], we show that polynomial time algorithms exist for the proposed decision problems, providing insights into the structure of these detective game like problems. Moreover, the practical algorithmic solution may readily find its way in real world applications, such as system design/monitoring/verification, security, and sensor-based forensics.

The rest of the paper is organized as follows. Section 2 defines the two detective games we study in this paper. For the case in which a single agent triggers all sensor recordings, Section 3 extracts a base graph structure that captures the connectivity of the environment, which is subsequently broken down into pieces for the incremental search introduced and analyzed in Section 4. Section 5 extends the graphs and search algorithm to account for additional agents that are present in the environment[1]. Section 6 discusses many open questions and concludes the paper.

2 Problem Formulation

2.1 Workspace, Agents and Stories

Let the *workspace* $W \subset \mathbb{R}^2$ be a bounded, path connected open set with a polygonal boundary, ∂W. Let one or more point agents move around in W, carrying out unknown tasks. Every agent has a map of W and may move arbitrarily fast along some continuous path $\tau : [t_0, t_f] \to W$. The initial time t_0 and the final time t_f are common to all agents. Assume that we are interested in a specific agent x with a *story* of its own, which may be truthful or fictional. Since an agent may not always have an accurate estimate of its state, a truthful story is not necessarily what has happened. For example, a human can usually recall a (partial, possibly inaccurate) sequence of events after she has performed a task in an environment. In this iteration we let the story have a very basic form: A sequence of places in W that agent x has visited in increasing chronological order,

[1] An interactive implementation of algorithms from Section 4 and 5 is available at `http://msl.cs.uiuc.edu/~jyu18/pe/cd.html`. A web browser with Java 1.5 support is required to access the page.

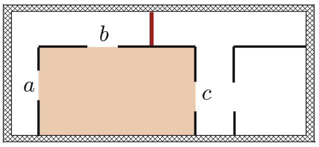


Fig. 1 A simple workspace with an occupancy sensor and a beam detector. The occupancy sensor guards the shaded area with three doorways a, b, and c. The beam detector guards the vertical line segment at the top.

$$\mathbf{p} = (p_1, p_2, \dots, p_n), \quad p_i \subset W,$$

such that the unique elements of **p** are each a simply connected region with a polygonal boundary and pairwise disjoint. The set of all unique elements of **p** is denoted $\mathscr{C}_p$. We assume that for every $p \in \mathscr{C}_p$, x has accounted for all its visits to p in **p**. As an example, agent x may simply report "I went from room A to room B, then came back to room A, and eventually arrived room C, at which point I stopped."

2.2 *Sensors and the Observation History*

Let a subset of the workspace W be guarded by a heterogeneous set of sensors. The placement of sensors in W is unknown to all agents. Among the commonly available sensors for surveillance, we focus on occupancy sensors and beam detectors. An occupancy sensor is assumed to detect the presence of an agent in a fixed, convex subset $s \subset W$. For example, a room may be monitored by such a sensor (the shaded area in Fig. 1). A data point recorded by an occupancy sensor o_i has two parts, an *activation*,

$$r_{oa} = (o_i, t_a),$$

and a *deactivation*,

$$r_{od} = (o_i, t_d),$$

in which t_a is the time when the first agent enters an empty s and t_d is the time when the last agent exits s. A beam sensor, on the other hand, guards a straight line segment, $\ell \subset W$, between two edges of ∂W (for example, the red line segment in Fig. 1). A data point of such a sensor, b_i, is recorded as an agent crosses ℓ, which can be represented by a 2-tuple:

$$r_b = (b_i, t).$$

A beam detector is deactivated right after activation. We further assume that when a beam detector is triggered by an agent, the agent must pass from one side of the beam to the other side. We denote the collection of all unique sensors in W as $\mathscr{C}_s$. With the introduction of occupancy sensors and beam detectors, we define the *observation history* simply as:

$$\mathbf{r} = (r_1, r_2, \dots, r_m),$$

in which each $r_i = r_{oa}, r_{od}$, or r_b, is indexed by the time when it occurs, incrementally.

Both occupancy sensors and beam detectors are weak sensors in the sense that they cannot tell an agent's passing direction. In the example given in Fig. 1, a sensor recording of the occupancy sensor could imply that the agent enters and exits from any of the doorways a, b, or c. Similarly, when the beam detector is triggered, an agent could be passing it from left to right or in the other direction. These sensors certainly cannot distinguish among different agents. We choose to work with these two typical but weak sensors so that the algorithms we present apply to a wider range of sensors, provided that they are at least as powerful (although the algorithms may not take full advantage of these stronger sensors). For example, a video camera is a stronger occupancy sensor, capable of providing both passing direction and identification of agents.

Observation History for a Single Agent. Without loss of generality, we assume that the sensors' detection regions (field of view) are pairwise disjoint: When two or more sensors have overlapping detection regions, we may create virtual sensors by repartitioning these sensors' detection regions so that the virtual sensors have disjoint detection regions [20]. This implies that when agent x is the only agent in W, the activation of any sensor must be followed by the deactivation of the same sensor, with no other sensor activities in between. In particular, in the observation history for a single agent, there can be no other sensor activations or deactivations between one activation/deactivation of an occupancy sensor.

Observation History for Multiple Agents. When there are multiple (an unknown number) agents in the workspace, it is no longer reasonable to assume that all sensor recordings have activation-deactivation intervals that are pairwise disjoint. For example, one agent may pass a beam detector while another one occupies a room monitored by an occupancy sensor. Different occupancy sensors can also have overlapping intervals of activation. When multiple agents are present in W, we assume that all sensor activation and deactivation times are distinct, since the likelihood of simultaneous sensor triggering is very low.

2.3 The Verification Problem

Given W, $\mathscr{C}_p, \mathscr{C}_s$, agent x's story $\mathbf{p}$ and the observation history $\mathbf{r}$, we are interested in determining whether $\mathbf{p}$ is consistent with $\mathbf{r}$. For the comparison to make sense, we require that both $\mathbf{p}$ and $\mathbf{r}$ span the same time interval, $[t_0, t_f]$. That is, we must determine whether there exists an agent path containing the locations given in $\mathbf{p}$ in the specified order that triggers the sensor recordings given by $\mathbf{r}$. For the purpose of introducing algorithms, we use the example workspace given in Fig. 2 and let agent x's story be:

$$\mathbf{p} = (A, C, B, A, C). \tag{1}$$

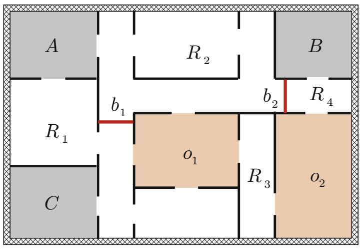


Fig. 2 A workspace with two beam detectors b_1, b_2, two occupancy sensors o_1, o_2, and three labeled rooms A, B and C. Thus, $\mathscr{C}_p = \{A, B, C\}$, $\mathscr{C}_s = \{b_1, b_2, o_1, o_2\}$. There are four connected components R_1 through R_4 when regions guarded by sensor range and rooms in agent x's story are treated as workspace obstacles.

In English, agent x told the story that it started in room A and went through room C, B, A, and C, in that order. When there is a single agent in the workspace, we let the observation history be:

$$\mathbf{r} = ((b_1, t_1), (o_1, t_2), (o_1, t_3), (b_2, t_4), (o_2, t_5), (o_2, t_6)). \tag{2}$$

For this simple example, it is not hard to see that $\mathbf{p}$ is not consistent with $\mathbf{r}$: B can only be visited after agent x passes b_2 from left to right; however, after visiting B, either b_2 or o_2, o_1 must be triggered for x to visit A once more. When there are multiple agents in the workspace, we let the observation history be slightly different (with which $\mathbf{p}$ is consistent):

$$\mathbf{r} = ((b_1, t_1), (o_1, t_2), (o_2, t_3), (b_2, t_4), (o_2, t_5), (o_1, t_6)). \tag{3}$$

In the example, we have implicitly made the assumption that elements of $\mathscr{C}_p$ and elements of $\mathscr{C}_s$ have coverage regions that are pairwise disjoint; overlapping cases will be handled after the main algorithms are introduced.

3 The Connectivity Graph and Sensing Induced Subgraphs

Both occupancy sensors and beam detectors, when not triggered, act as obstacles that change the workspace connectivity. When a sensor is triggered, the part of the workspace blocked by that sensor is temporarily connected. To explore the structure from this intuition, we first build a *connectivity graph* G that captures the topological features of W. As we are only interested in finding a path[2] through $\mathbf{p}$ that is compliant with $\mathbf{r}$, we only need G to capture how elements of $\mathscr{C}_p$ are connected and how they are connected to the sensors, $\mathscr{C}_s$. Therefore, we treat these elements as

[2] In graph theory, a path does not visit one vertex multiple times. Therefore, the image of a continuous path, when discretized, becomes a *walk* in graph theory terminologies, since it may visit a vertex multiple times. In this paper, we abuse the term *path* slightly to denote both a continuous function $\tau : [t_0, t_f] \to W$ and the corresponding walk in a discrete graph.

vertices of G. Since there are two possible directions that an agent may pass a beam detector, two vertices are needed for each beam detector. A single vertex is needed for each element of $\mathscr{C}_p$ and for each location guarded by an occupancy sensor. For the example from Fig. 2, the collection of vertices is

$$V = \{A, B, C, o_1, o_2, b_{1u}, b_{1d}, b_{2l}, b_{2r}\},$$

in which b_{1u}, b_{1d} are the upper and lower sides of b_1, respectively (these two sides are naturally obtained if a beam is represented as two oppositely oriented edges, as commonly used in computations involving polygons). Similarly, b_{2l}, b_{2r} are the left and right sides of b_2.

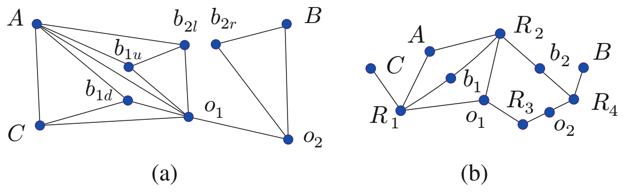


Fig. 3 a) The connectivity graph of the example given in Fig. 2. b) An alternative connectivity graph including connected components of W_{free} as vertices.

To connect the vertices, we need to obtain the connectivity of the workspace algorithmically, treating the regions occupied by elements of $\mathscr{C}_p$ and $\mathscr{C}_s$ as obstacles. Denote the workspace excluding these obstacles as W_{free}. Determining the connectivity of W_{free} is equivalent to finding the connected components of W_{free}. We call the subroutine that does this BUILDCONNECTIVITYGRAPH but omit the code since it is a fairly standard procedure [3]. Applying this procedure to our example yields the connectivity graph $G = (V, E)$ given in Fig. 3(a). We point out that there are other choices in constructing the connectivity graph. For example, following an (more natural) equivalence class approach, we may alternatively build the graph based on how regions R_1 through R_4 are connected (Fig. 3(b)). We may further treat sensors and rooms as directed edges. There are no fundamental differences between these choices for our purpose: Although the later two provide simpler graphs, slightly more sophisticated graph search routines would then be needed.

With G constructed, we can now explore the extra information provided by the observation history: The relative timing of sensor recordings. This information essentially partitions G into different pieces at different time instances. In this section we focus on the case of workspace with a single agent. In the observation history given in (2), b_1 is the first sensor that is set off. This means that at the time

[3] One efficient way of doing this is to apply a cell decomposition procedure (see [19], Chapter 6), such as vertical cell decomposition [7], to W_{free} and then combine the cells that share borders. For more details, please refer to the extended version of the paper at `http://msl.cs.uiuc.edu/~jyu18/wafr10/full.pdf`

right before t_1 when the sensor is activated, the agent must be at either b_{1u} or b_{1d}. During the time interval $[t_0, t_1)$, since b_2, o_1, and o_2 are inactive, they act as obstacles. The part of G that the agent may travel during $[t_0, t_1)$ is then given by G_1 in Fig. 4, in which A is the start vertex and b_{1u}, b_{1d} are the possible goal vertices. Vertex B does not appear in G_1 because it is not reachable. Similarly, we obtain the subgraphs of G during time intervals $(t_1, t_2), (t_3, t_4), (t_4, t_5), (t_6, t_f]$ as G_2 through G_5 in Fig. 4, respectively. Graph G_2 has two parts since there are two possible start vertices. Note that when the start and goal vertices in these subgraph correspond to sensor vertices, they can be visited only once as the start vertex or the goal vertex. The pseudocode is given in GETSUBGRAPH (Algorithm 1). The algorithm calls the subroutine GETREACHABLESUBGRAPH(G, s, V_C, V_G) (Algorithm 2), which returns the part of G reachable from s, passing only vertices in V_C. If V_G is not empty, then a path from s must also end at vertices of V_G. We separate this subroutine since it will be reused. In Algorithm 2, subroutine CONNECTEDCOMPONENT(G, s) returns the connected component of G containing s. We note that, although it is possible to work with G directly instead of working with these subgraphs, they will be helpful in understanding the algorithm and in complexity analysis. Moreover, it can be a good heuristic to build these subgraphs to restrict search in problems with large workspaces.

Algorithm 1. GETSUBGRAPH

```
Input: $G = (V, E)$, the start vertex $s$, $\mathscr{C}_p$, and goal vertices $V_G$
Output: $G' = (V', E')$, the part of $G$ that is reachable from $s$

1: $V_C \leftarrow \mathscr{C}_p \cup \{s\}$
2: return GETREACHABLESUBGRAPH$(G, s, V_C, V_G)$
```

Algorithm 2. GETREACHABLESUBGRAPH

```
Input: $G = (V, E)$, $s$, $V_C$, $V_G$
Output: $G' = (V', E')$

 1: for all edges $(v_i, v_j) \in E$ such that $v_i, v_j \in V_C$ do
 2:     add $(v_i, v_j)$ to $E'$        //  $V'$ is also updated.
 3: end for
 4: $G' \leftarrow$ CONNECTEDCOMPONENT$(G', s)$
 5: if $V_G$ is not empty then
 6:     for all $v_i, v_j$ such that $v_i \in V', v_j \in V_G$ do
 7:         if $(v_i, v_j) \in E$ then
 8:             add $(v_i, v_j)$ to $E'$
 9:         end if
10:     end for
11:     $V' \leftarrow V' \cup V_G$
12: end if
13: return $G'$
```

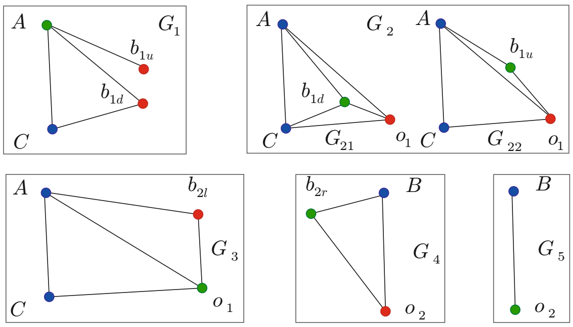


Fig. 4 The subgraphs of G induced by the sensor observation history. The green vertices are possible start positions and the red vertices are possible goal positions.

The correctness of Algorithm 1 through 2 is by construction, which is straightforward to verify. We now give an estimate of the worst case performance of these algorithms. Let W_{free} have an input size of n_w, BUILDCONNECTIVITYGRAPH has time complexity $O(n_w^2)$. In subroutine GETREACHABLESUBGRAPH, the subroutine for obtaining connected components takes time linear in n_w [16]. The complexity is then decided by the for loop at line 6 and the membership check at line 7, which takes no more than $O(|V_G| n_w \lg n_w)$ in total.

4 Validating a Single Agent's Story against an Observation History of a Single Agent

When there is a single agent in the workspace, every sensor recording is triggered by that agent. In this case, supposing that we have the subgraphs of G, the rest of the work becomes searching through these graphs, one by one, for a path that agrees with the agent's story. A straightforward approach is to connect one subgraph's goal vertices to the next subgraph's start vertices and perform an exhaustive search through paths to see whether there are matches. Such naive algorithms are not scalable, however, since every beam detector can require connecting the subgraphs in two ways (for example, G_{21}, G_{22} in Fig. 4). The number of search paths through the subgraphs is then exponential in the number of sensor recordings on average. In the worst case, breadth-first or depth-first search through all these graphs may take an exponential amount of time.

To organize the search more efficiently, we first connect the subgraphs to get a better understanding of the topology of the graph to be searched. To make the structure more explicit for search, we also make the subgraphs directed. Doing this to all sensing induced subgraphs of G yields the graph illustrated in Fig. 5. Denote this graph G_s. The problem of validating $\mathbf{p}$ against $\mathbf{r}$ becomes searching through G_s for a path $\mathbf{p}'$ such that, after deleting the vertices corresponding to sensors from $\mathscr{C}_s$, $\mathbf{p}'$ is exactly $\mathbf{p}$. We observe that, since G_s contains at most $2(m+1)$ copies of G, any

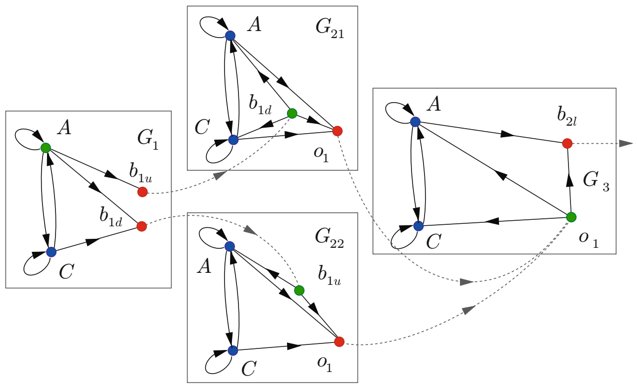


Fig. 5 Part of the composite graph G_s built from the connectivity graph G and sensor observation history $\mathbf{r}$.

element of $\mathscr{C}_p \cup \mathscr{C}_s$ cannot appear more than $O(m)$ times in G_s. This observation indicates it may be possible to apply the principles of dynamic programming to partition of the search problem into subproblems: Each subproblem is validating a tail $(p_i, \ldots, p_n)$ of $\mathbf{p}$, starting from a subset of vertices of G_s corresponding to p_{i-1}. The total number of subproblems per p_i is $O(m)$; if going from one subproblem to a smaller subproblem takes polynomial amount of time, then the total time spent on searching G_s is also polynomial.

This turns out to be the case for our problem. Before formally introducing the algorithm, we illustrate how it operates with the provided example. We write the agent's story compactly as $\mathbf{p} = ACBAC$. Since agent x starts in A, we are done with $p_1 = A$, leaving $CBAC$ to validate. For $p_2 = C$, it is possible to reach from A in G_1 to copies of C in G_1, G_{21} (passing b_{1u}, b_{1d}) , and G_3 (passing b_{1d}, b_{1u}, o_1). The copy of C in G_{22} is not directly reachable from A in G_1, passing only vertices from sensors. We may write the three subproblems as (For $P \in \mathscr{C}_p$, P_{G_i} denotes that the copy of P is from the subgraph G_i. For example, A_{G_1} denotes the copy of A from the subgraph G_1):

$$\begin{array}{r|l} A_{G_1}C_{G_1} & BAC, \\ A_{G_1}b_{1u}b_{1d}C_{G_{21}} & BAC, \\ A_{G_1}b_{1d}b_{1u}o_1C_{G_3} & BAC. \end{array}$$

Since there are multiple subproblems for $p_3 = B$, going through these subproblems individually may introduce a factor of $O(m)$ per problem; there can be $O(m)$ subproblems, which will contribute a factor of $O(m^2)$ to the overall running time. To avoid this, we again use the sequential nature of G_s. Instead of processing each subproblem individually, we process all of them together, staged at each G_j. For our example, the first subproblem starts with the copy of C in G_1: It is possible to go through b_{1d}, b_{1u} and get to o_1. We now pick up the second subproblem and see that it is possible to go from C in G_{21} to o_1 as well. At this point, the first two subproblems collapse into a single subproblem. Going into G_3, we pick up the third subproblem.

For the copy of C in G_3, since it must pass A to reach B, this subproblem dies; we are left with a single subproblem to reach B from b_{21} in G_3. Following above procedure, we obtain two subproblems after processing $p_3 = B$:

$$A_{G_1} b_{1u} b_{1d} C_{G_{21}} o_1 b_{21} b_{2r} B_{G_4} \Big| AC,$$
$$A_{G_1} b_{1u} b_{1d} C_{G_{21}} o_1 b_{21} b_{2r} o_2 B_{G_5} \Big| AC.$$

Note that we do not keep all valid paths in this search; doing so will require space exponential in m. After all of $\mathbf{p}$ is processed, if some subproblems survive, then $\mathbf{p}$ is consistent with $\mathbf{r}$; any surviving subproblem also provides a feasible path.

Algorithm 3. VALIDATEAGENTSTORY

Input: $G, \mathbf{p} = (p_1, \ldots, p_n), \mathbf{r} = (r_1, \ldots, r_m)$
Output: **true** if $\mathbf{p}$ is consistent with $\mathbf{r}$, **false** otherwise

```
 1: $V_I \leftarrow \{p_1\}$
 2: for $j = 1$ to $m+1$ do
 3:     initialize $V_G$ as an empty set
 4:     if IsDeactivation($r_j$) then
 5:         continue
 6:     end if
 7:     if ($j \neq m+1$) then
 8:         $V_G \leftarrow$ SensorVertices($r_i$)
 9:     else
10:         empty $V_G$
11:     end if
12:     $G_j \leftarrow$ GetSubGraph($G, V_I, \mathscr{C}_p, V_G$)
13:     $V_I \leftarrow V_G$
14: end for
15: $G_s \leftarrow$ Chain($G_1, \ldots, G_{m+1}$)
16: initialize $V_s, V_s'$ as empty sets of two tuples
17: $V_s \leftarrow \{(p_1, 1)\}$      //   A two tuple is a vertex of $G_s$
18: for $i = 2$ to $n$ do
19:     for $j = 1$ to $m+1$ do
20:         if $(p_i, j)$ adjacent to $(p_{i-1}, k) \in V_s$ for some $k \leq j$ then
21:             if $i == n \&\& j == m+1$ then
22:                 return true
23:             end if
24:             add $(p_i, j)$ to $V_s'$
25:         end if
26:     end for
27:     $V_s \leftarrow V_s'$; empty $V_s'$
28: end for
29: return false
```

The pseudocode is summarized in Algorithm 3. Subroutine ISDEACTIVATION(r) returns true only if r is the deactivation of an occupancy sensor. The subroutine SENSORVERTICES(r) returns the vertices of G induced by the sensor in a sensor recording r. The subroutine CHAIN$(\ldots)$ connects all input graphs sequentially based on sensor crossings, which is trivial to implement. In the code, we use (p_i, j) to denote the the copy of p_i in subgraph G_j. The correctness of VALIDATEAGENTSTORY follows from its construction based on dynamic programming, which we briefly corroborate. After each p_i is worked on, there are up to $O(m)$ subproblems since there are no more than $O(m)$ copies of p_i in G_s. Because the further observation that G_s is sequential, the suproblems for each p_i can be processed in a single pass of G_s. We make $O(m)$ calls to GETSUBGRAPH, which takes $O(mn_w \lg n_w)$ total time (Since $|V_G| \leq 2$ in calls to GETSUBGRAPH). Going through the for loops, it is straightforward to get that the rest of the algorithm has complexity no worse than $O(n \cdot m \lg n_w) = O(nm \lg n_w)$. The worst case running time is then upper bounded by $O(m(n+n_w)\lg n_w)$.

As mentioned in the problem formulation, we have assumed that $\mathscr{C}_p$ and $\mathscr{C}_s$ do not overlap in $\mathbb{R}^2$. These are not included in the above algorithm to avoid complicating the presentation. What if some $p \in \mathscr{C}_p$ and $s \in \mathscr{C}_s$ do overlap? There are several subcases. If the regions of p, s coincide (for example, there may be an occupancy sensor in room A), this essentially breaks the problem into several smaller problems, to which the above algorithm applies. If $s \subsetneq p$, agent x then must go through p to reach s, in which case we can build G to make s a vertex connecting to p only. The same applies if $p \subsetneq s$. In the last case s, p partially overlap but do not include each other; we can treat s, p as three regions: $s \backslash p$ as an sensor, $p \backslash s$ as a room, and $s \cap p$ as a fully overlapping sensor and room (this case only happens to occupancy sensors, not beam detectors). This will split the verification problem into several subproblems, which may induce exponential growth in running time. However, the last case is not likely to often happen since occupancy sensors are usually placed to guard an entire room. We can also minimize such a problem by carefully placing the sensors.

5 Validating a Single Agent's Story against an Observation History of Multiple Agents

When there are multiple agents (an unknown number) in the workspace, two complications arise. First, as mentioned in Section 2, when multiple agents are in the workspace, there can be many agents in the region monitored by an occupancy sensor during one of its activation-deactivation time interval. Effectively, this allows any agent to temporarily go through the region monitored by an activated occupancy sensor. This suggests that the recordings from occupancy sensors should only be treated as events that change the connectivity of the environment. That is, whether agent x is the agent that triggered the activation/deactivation of an occupancy sensor is not relevant. Second, agent x may not be responsible for all beam detector recordings. Instead, it may trigger any subsequence of sensor recordings. For an observation history sequence of length m, there are up to $O(2^m)$ possible subsequences; agent x may have triggered any one of these subsequences, but not the rest of $\mathbf{r}$.

To overcome these difficulties, we start by examining how the composite graph G_s changes. As analyzed above, only beam detector events need to be considered for agent x. For occupancy sensors, we maintain an active list as $\mathbf{r}$ is processed; additional processing is needed only when deactivation of an occupancy sensor happens. To illustrate the procedure for building the composite graph, we use $\mathbf{p}$ from (1) and $\mathbf{r}$ from (3). Starting with the first beam detector recording, (b_1, t_1), if agent x is responsible for it, then the reachable part before b_1 is crossed is the same as G_1 from Fig. 4. To emphasize that this graph is built from t_0 to t_1, we denote it as G_1^0. The next two recordings in $\mathbf{r}$ are activations of o_1 and o_2. Since there are only activations, which only cause more locations of the environment to become reachable, we store these in the active occupancy sensor list and continue.

For the next recording, (b_2, t_4), three subgraphs need to be built, one start from A, one start from b_{1u}, and one start from b_{1d}. Following the naming convention of G_1^0, these should have names G_4^0, G_4^{11}, and G_4^{12}, respectively. To build G_4^0, we need to keep vertices $\{A, b_{2l}, b_{2r}, o_1, o_2\}$. We also add $\{B, C\}$ since these are vertices of $\mathscr{C}_p$ that are reachable from $\{A, b_{2l}, b_{2r}, o_1, o_2\}$ without crossing additional sensors. This gives us the subgraph in Fig. 6(a). To facilitate searching, for each pair of neighbors of an active occupancy sensor, we add an edge between them and remove the occupancy sensor vertex, which yields the graph in Fig. 6(b). The pesudocode for building this subgraph, GETSUBGRAPHMULTI, is given in Algorithm 4. Assuming we have obtained G_4^{11} and G_4^{12} similarly, the next sensor recordings is (o_2, t_5),

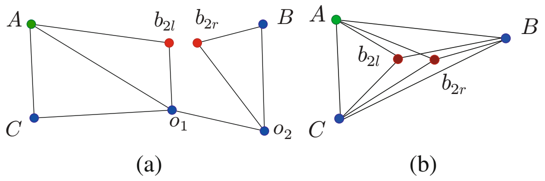


Fig. 6 a) One possible connectivity subgraph, G_4^0, during (t_4, t_5) when mulitple agents are in the workspace. b) Updated connectivity subgraph reflecting whether two vertices are reachable without triggering additional sensor recordings.

Algorithm 4. GETSUBGRAPHMULTI

Input: $G = (V, E)$, the start vertex s, $\mathscr{C}_p$, the active occupancy sensors O, and goal vertices V_G.

Output: $G' = (V', E')$, the part of G that is reachable from s.

```
1: $V_C \leftarrow \mathscr{C}_p \cup O \cup \{s\}$
2: $G' \leftarrow$ GETREACHABLESUBGRAPH$(G, s, V_C, V_G)$
3: for all $o \in (O \cap V')$ do
4:     add to $E'$ an edge between each pair of $o$'s neighbors
5:     remove $o$ from $G'$
6: end for
7: return $G'$
```

which corresponds to the deactivation of o_2. For this event, we need to create five subgraphs starting from $A, b_{1u}, b_{1d}, b_{2l}, b_{2r}$ with names $G_5^0, G_5^{11}, G_5^{12}, G_5^{41}, G_5^{42}$, respectively. Since during $[t_5, t_f]$, no new locations of G become reachable and no other beam sensor recordings happen, this part of $\mathbf{r}$ can be ignored. After connecting all these subgraphs based on sensor crossings, we obtain the composite graph G_s as illustrated in Fig. 7.

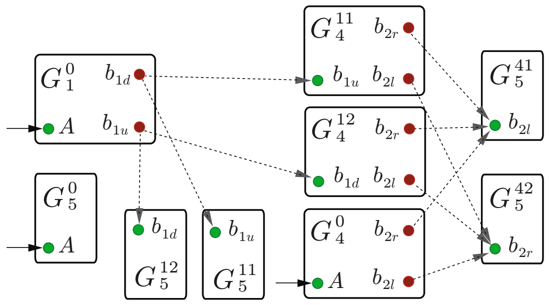


Fig. 7 Sketch of the composite graph G_s.

Before continuing with searching G_s, we make one observation from above graph building procedure: Since each sensor recording may cause up to $O(m)$ new subgraphs to be built, up to $O(m^2)$ subgraphs may be built altogether. This is the number of subgraphs in G_s since each subgraph appears once in G_s. To search through G_s for a path matching $\mathbf{p}$, the same strategy from VALIDATEAGENTSTORY can be applied. That is, a dynamic programming approach can be used in which a subproblem is a tail of $\mathbf{p}$ and a location in G_s. Since there are no more than $O(m^2)$ copies of p_i in G_s, there can be at most $O(m^2)$ subproblems after each p_i is processed. Similar to VALIDATEAGENTSTORY, during the processing of each p_i each subgraph only needs to be considered once. This limit the time complexity of searching G_s at $O(nm^2 \log m_w)$. To get the total time complexity, we need the time for building G_s, which is m^2 times the cost of the subroutine GETSUBGRAPHMULTI. The running time of GETSUBGRAPHMULTI is determined by the loop at lines 3 through 6, which takes $O(m_w^3)$ time. This yields the overall time complexity $O(m^2(n \log m_w + m_w^3))$. Since the algorithm operates much like VALIDATEAGENTSTORY, we omit the pesudocode.

6 Conclusion and Open Questions

We introduced a decision problem in which a story of an autonomous agent is validated against the observation history gathered by simple sensors placed in the same environment. We showed that sensor recordings act as temporary obstacles that constrain agents' movements in the environment, effectively creating a sequence of connectivity subgraphs of the environment. Based on this observation, we designed polynomial time algorithms that decide whether the agent's story is possible given the observation history and retrieves a possible path if one exists.

As a first attempt at a new problem, more questions are opened than answered. Some immediate ones are: What if the agent can only recall a partial story? In the multiple agent case, what if every agent's story is valid but the combined story is inconsistent? In letting the agents move arbitrarily fast, we only addressed one discrete aspect of a general problem in this paper. A natural next step is to work on a kinematic agent model with bounded control inputs, which seems to require more careful analysis of the continuous state space and much richer behaviors. Since an agent's speed is limited, some plausible paths the agent could have taken will now be ruled out. Continuous state space also makes it possible to study optimality: a network of sensors can potentially detect whether an agent is performing its task most efficiently, in terms of time or distance traveled. This becomes more relevant as system designs become more and more complex; sometimes it is challenging to just get a feasible plan [11, 18]. Another interesting direction is to study optimal sensor placements for the detective task, which was discussed to some extent in [24, 28]. Furthermore, the story in this paper only contains "when" and "where". It is an intriguing open problem to check stories involving "what", "how", and "why"? Logic seems essential in investigating these elements of a story.

Probabilistic formulations of the story validation problem may also be fruitful to explore. The agents in this paper are assumed to be nondeterministic in that the algorithms treat all possible paths equally. In a real environment, however, a typical agent usually does have preferences when multiple choices are present. Models considering the transition probability of an agent from room to room could then uncover the most likely path(s) taken by the agent. If none of the feasible paths taken by the agent are probable, the agent could still be considered as misbehaving. Probability can also be introduced into sensor observation history. Sensors, no matter how reliable they are, may have false positives and false negatives. For example, a beam sensor may misfire (triggered by a mouse for instance) with small probability. This implies that sensors, unlike walls, may be better treated as "soft" obstacles.

Acknowledgements. The authors thank Amir Nayyeri, Max Katsev, and the anonymous reviewers for helpful comments. This work was supported in part by NSF grants 0904501 (IIS Robotics) and 1035345 (Cyberphysical Systems), DARPA SToMP grant HR0011-05-1-0008, and MURI/ONR grant N00014-09-1-1052.

References

1. Alur, R., Henzinger, T.A., Lafferriere, G., Pappas, G.J.: Discrete abstractions of hybrid systems. Proceedings of the IEEE, 971–984 (2000)
2. Asarin, E., Bournez, O., Dang, T., Maler, O.: Approximate reachability analysis of piecewise-linear dynamical systems, pp. 21–31. Springer, Heidelberg (2000)
3. Aslam, J., Butler, Z., Constantin, F., Crespi, V., Cybenko, G., Rus, D.: Tracking a moving object with a binary sensor network, pp. 150–161. ACM Press, New York (2002)

4. Baryshnikov, Y., Ghrist, R.: Target enumeration via integration over planar sensor networks. In: Proceedings of Robotics: Science and Systems IV, Zurich, Switzerland (June 2008)
5. Bellman, R.E.: Dynamic Programming. Princeton University Press, Princeton (1957)
6. Castellanos, J., Montiel, J., Neira, J., Tardós, J.: The SPmap: A probabilistic framework for simultaneous localization and mapping. IEEE Transactions on Robotics & Automation 15(5), 948–953 (1999)
7. Chazelle, B.: Approximation and decomposition of shapes. In: Schwartz, J.T., Yap, C.K. (eds.) Algorithmic and Geometric Aspects of Robotics, pp. 145–185. Lawrence Erlbaum Associates, Hillsdale (1987)
8. Cheng, P., Kumar, V.: Sampling-based falsification and verification of controllers for continuous dynamic systems. In: Proceedings Workshop on Algorithmic Foundations of Robotics (2006)
9. Choset, H., Nagatani, K.: Topological simultaneous localization and mapping (T-SLAM). IEEE Transactions on Robotics & Automation 17(2), 125–137 (2001)
10. Chutinan, A., Krogh, B.: Computational techniques for hybrid system verification. IEEE Transactions on Automatic Control 48(1), 64–75 (2003)
11. Conner, D.C., Kress-gazit, H., Choset, H., Rizzi, A.A., Pappas, G.: Valet parking without a valet. In: IEEE/RSJ International Conference on Intelligent Robots & Systems (2007)
12. Dehn, M.: Papers on Group Theory and Topology. Springer, Berlin (1987)
13. Dissanayake, G., Newman, P., Clark, S., Durrant-Whyte, H.F., Csorba, M.: A solution to the simultaneous localisation and map building (SLAM) problem. IEEE Transactions on Robotics & Automation 17(3), 229–241 (2001)
14. Epstein, D.B.A., Paterson, M.S., Camon, G.W., Holt, D.F., Levy, S.V., Thurston, W.P.: Word Processing in Groups. A. K. Peters, Natick (1992)
15. Fang, Q., Zhao, F., Guibas, L.: Counting targets: Building and managing aggregates in wireless sensor networks. Technical Report P2002-10298, Palo Alto Research Center (June 2002)
16. Hopcroft, J., Tarjan, R.: Efficient algorithms for graph manipulation. Communications of the ACM 16(6), 372–378 (1973)
17. Kim, W., Mechitov, K., Choi, J., Ham, S.: On target tracking with binary proximity sensors. In: ACM/IEEE International Conference on Information Processing in Sensor Networks, pp. 301–308. IEEE Press, Los Alamitos (2005)
18. Kress-gazit, H., Fainekos, G.E., Pappas, G.: Where's waldo? sensor-based temporal logic motion planning. In: Proceedings IEEE International Conference on Robotics & Automation, pp. 3116–3121 (2007)
19. LaValle, S.M.: Planning Algorithms. Cambridge University Press, Cambridge (2006), `http://planning.cs.uiuc.edu/`
20. LaValle, S.M.: Filtering and planning in information spaces. Technical report, Department of Computer Science, University of Illinois (October 2009)
21. Montemerlo, M., Thrun, S., Koller, D., Wegbreit, B.: FastSLAM: A factored solution to the simultaneous localization and mapping problem. In: Proceedings AAAI National Conference on Artificial Intelligence (1999)
22. Parr, R., Eliazar, A.: DP-SLAM: Fast, robust simultaneous localization and mapping without predetermined landmarks. In: Proceedings International Joint Conference on Artificial Intelligence (2003)
23. Piterman, N., Pnueli, A., Sa'ar, Y.: Synthesis of reactive(1) designs. In: Proceedings Verification, Model Checking, and Abstract Interpretation, pp. 364–380. Springer, Heidelberg (2006)

24. Rao, B.S.Y., Durrant-Whyte, H.F., Sheen, J.S.: A fully decentralized multi-sensor system for tracking and surveillance. International Journal of Robotics Research 12(1), 20–44 (1993)
25. Shrivastava, N., Mudumbai, R., Madhow, U., Suri, S.: Target tracking with binary proximity sensors: fundamental limits, minimal descriptions, and algorithms. In: Proc. 4th Internat. Conf. on Embedded Networked Sensor Systems, pp. 251–264. ACM Press, New York (2006)
26. Singh, J., Kumar, R., Madhow, U., Suri, S., Cagley, R.: Tracking multiple targets using binary proximity sensors. In: Proc. Information Processing in Sensor Networks (2007)
27. Thrun, S., Burgard, W., Fox, D.: A probabilistic approach to concurrent mapping and localization for mobile robots. Machine Learning 31(5), 1–25 (1998)
28. Tovar, B., Cohen, F., LaValle, S.M.: Sensor beams, obstacles, and possible paths. In: Proceedings Workshop on Algorithmic Foundations of Robotics (2008)
29. Tovar, B., Murrieta, R., LaValle, S.M.: Distance-optimal navigation in an unknown environment without sensing distances. IEEE Transactions on Robotics 23(3), 506–518 (2007)
30. Yu, J., LaValle, S.M.: Tracking hidden agents through shadow information spaces. In: Proceedings IEEE International Conference on Robotics & Automation (2008)
31. Yu, J., LaValle, S.M.: Probabilistic shadow information spaces. In: Proceedings IEEE International Conference on Robotics & Automation (2010)

Gravity-Based Robotic Cloth Folding

Jur van den Berg, Stephen Miller, Ken Goldberg, and Pieter Abbeel

Abstract. We consider how the tedious chore of folding clothes can be performed by a robot. At the core of our approach is the definition of a cloth model that allows us to reason about the geometry rather than the physics of the cloth in significant parts of the state space. We present an algorithm that, given the geometry of the cloth, computes how many grippers are needed and what the motion of these grippers are to achieve a final configuration specified as a sequence of *g-folds*—folds that can be achieved while staying in the subset of the state space to which the geometric model applies. G-folds are easy to specify and are sufficiently rich to capture most common cloth folding procedures. We consider folds involving single and stacked layers of material and describe experiments folding towels, shirts, sweaters, and slacks with a Willow Garage PR2 robot. Experiments based on the planner had success rates varying between 5/9 and 9/9 for different clothing articles.

1 Introduction

An English patent for a clothes washing machine was issued in 1691. Since then, there have been many innovations in washing and drying, but folding of clothes remains a manual (and notoriously tedious) activity. In this paper, we present a geometric model and algorithms that are steps toward autonomous robot folding of clothes. Cloth is highly non-rigid, flexible, and deformable with an infinite-dimensional configuration space. We consider articles of clothing that can be described by a simple polygonal boundary when lying flat on a horizontal surface.

We introduce a deterministic geometric model of cloth motion based on gravity and assumptions about material properties. We constrain robot motion such that at all times, one part of the cloth is lying horizontally on the table and one part (possibly empty) hangs vertically from the grippers parallel to the gravity vector. This allows a configuration of the cloth to be fully determined by the line that separates

Jur van den Berg · Stephen Miller · Ken Goldberg · Pieter Abbeel
University of California, Berkeley
e-mail: {berg,sdavidmiller,goldberg,pabbeel}@berkeley.edu

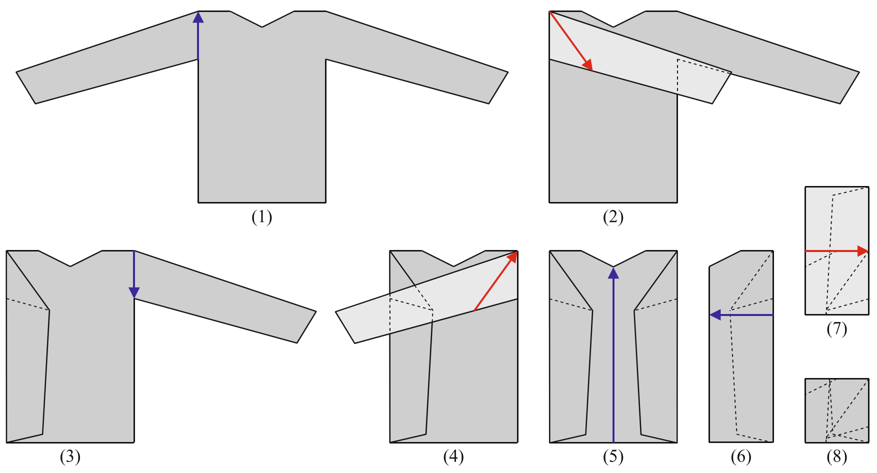


Fig. 1 Folding a long-sleeve into a square using a sequence of seven g-folds. Red g-folds apply to the geometry that was folded in the preceding g-fold. Blue g-folds apply to the entire geometry.

the horizontal and the vertical parts. We call this line the *baseline*, which defines a 2-D configuration space for the material.

Given polygonal geometry of the cloth, number of grippers, and desired fold sequence (see Fig. 1), we present an algorithm that computes a motion plan for the grippers that moves the cloth through the C-space to reach the desired final arrangement, or a report that no such motion plan exists. We implemented the algorithm on a Willow Garage PR-2 robot and report experiments folding towels, t-shirts, sweaters, and slacks.

The remainder of this paper is organized as follows. In Section 2 we discuss related work. In Section 3 we define the problem. In Section 4 we describe our algorithm to compute the manipulation motion of the robot to execute a given folding sequence using g-folds. In Section 5 we report experimental results folding towels, t-shirts, and slacks using a Willow Garage PR-2 robot. We conclude in Section 6.

2 Related Work

The work that is most related to ours is the work of Bell and Balkcom [3, 4]. In [4], the grasp points are computed to immobilize a polygonal non-stretchable piece of cloth. Gravity is used to reduce the number of grasp points to hold cloth in a predictable configuration, potentially with a single fold, using two grippers in [3]. We extend this work and include a folding surface. We assume that points that are lying on a table are fixed by friction and gravity, and need not be grasped. The work of [3] also shows how to fold a t-shirt using the *Japanese method*; the fold can be achieved by grasping the cloth at three points without regrasping.

In [7], the robot handling of cloth material is discussed and some specific folds are shown. The work of [14] also discusses a specific folding manipulation. The work of [12] deals specifically with folding towels. The work focuses on visual detection of the vertices of the towel, and use a scripted motion to achieve folds using a PR-2 robot. We build on the results of this work in our experiments.

There is also quite a large body of work on *cloth simulation*, which simulates the behavior of cloth under manipulation forces using the laws of physics [2, 5, 6]. In our work, we manipulate cloth such that it is always in a configuration that allows us to reason about the geometry of the cloth, rather than about its physics and dynamics.

Folding has been extensively studied in the context of *origami* [1, 9, 10]. Origami, or paper folding, is fundamentally different from cloth folding, since unfolded regions of the paper are considered to be rigid facets connected by "hinges" that model the creases in the paper. In contrast, cloth material is flexible everywhere. Yet, we draw from results in paper folding in our work. Applications of paper folding outside origami include box folding [13, 11] and metal bending [8], where the material model is essentially the same as that of paper.

3 Problem Description

We assume gravity acting in the downward vertical ($-z$) direction and a sufficiently large planar table in the horizontal (xy) plane. We assume the article of clothing can be fully described by a simple *polygon* (convex or non-convex) initially lying on the horizontal surface. We are given the initial n vertices of the polygonal cloth in counterclockwise order.

We make the following standard assumptions on the cloth material:

1. The cloth has *infinite flexibility*. There is no energy contribution from bending.
2. The cloth is *non-stretchable*. No geodesic path lengths can be increased.
3. The cloth has *infinite friction* with the surface on which it lies and with itself.
4. The cloth has *zero thickness*.
5. The cloth is *subject to gravity*.
6. The cloth has *no dynamics*.

At the core of our approach is the following additional assumption, which we call the *downward tendency assumption:*

7. If the cloth is held by a number of grippers, and one or more grippers release the cloth, no point of the cloth will move upwards as a result of gravity and internal forces within the cloth.

This assumption does not directly follow from physics, rather it is an approximation which seems to match the behavior of reasonably shaped cloth, such as everyday clothing articles, surprisingly well[1], and it allows us to reason purely about the *geometry* rather than the physics of the cloth.

[1] The assumption is not accurate for an exotic family of shapes called pinwheels, as shown in [3].

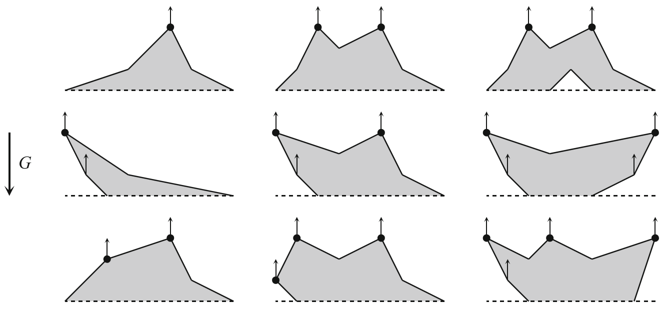


Fig. 2 Examples of vertical parts of cloths in various configurations. In order for the cloth not to be immobilized, all convex vertices not at the baseline at which the negative gravity vector (small arrows) does not point into the cloth must be grasped. These vertices are indicated by the dots.

The downward-tendency assumption allows the cloth to be held by the grippers such that one section lies horizontally on the surface and another section hangs vertically. The line that separates the horizontal and the vertical parts is called the *baseline*. To ensure deterministic behavior of the cloth, the grippers must be arranged so that the vertical section is immobilized. The points that are lying on the surface (including those on the baseline) are immobilized, as they cannot move in the plane due to friction and will not move upward per the downward-tendency assumption, so they need not be grasped. Fig. 2 shows an example, where points of the cloth are held by grippers. To make sure that the vertical part of the cloth is immobilized, it turns out that every *convex* vertex of the vertical part of the cloth at which the negative gravity vector does not point into the cloth polygon must either be held by a gripper or be part of the baseline. This follows from the following theorem:

Theorem 1. *In our material model, a vertically hanging cloth polygon is immobilized when every* convex *vertex of the cloth at which the negative gravity vector does not point into the cloth polygon is fixed (i.e. be held by a gripper or be part of the baseline).*

Proof: By [3], we know that a non-stretchable planar tree is fully immobilized if each node of the tree of which its incident edges do not positively span $\mathbb{R}^2$ is fixed. Now, let us define an *upper string* of a polygon as a maximal sequence of edges of which the extreme vertices are convex vertices of the polygon, and no part of the polygon lies above the edges (see Fig. 3(a)). A given polygon P can have multiple upper strings, but has at least one.

For each upper string holds that at its convex vertices the negative gravity vector points outside the polygon. As these convex vertices are fixed (by a gripper), the entire set of edges the string consists of is immobilized. This can be seen by adding virtual vertical edges fixed in gravity pointing downward from the non-convex vertices, which make sure that the non-convex vertices cannot move upward (per the downward-tendency assumption). The incident edges of the non-convex vertices now positively-span $\mathbb{R}^2$, hence the entire string is immobilized.

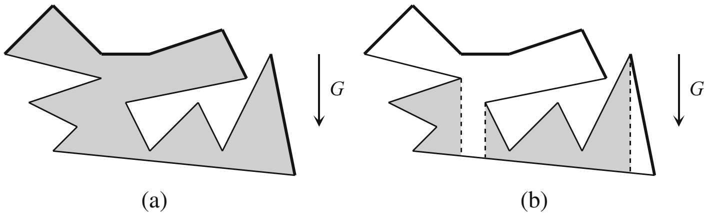


Fig. 3 (a) A polygon with two upper strings shown thick. **(b)** The white part of the polygon (including the vertical dashed edges) has proven immobilized. The grey part remains.

Now, every point of the polygon P that can be connected to an upper string by a vertical line segment that is fully contained within P is immobilized. This is because this point cannot move downward per the non-stretchability assumption (note that the upper string is immobilized), and it cannot move upward per the downward-tendency assumption. Hence, all such points can be "removed" from P – they have been proven immobilized. What remains is a smaller polygon P' (potentially consisting of multiple pieces) for which immobilization has not been proven (see Fig. 3(b)). The smaller polygon P' has vertical edges that did not belong to the original polygon P. The points on these vertical edges are immobilized, including both incident vertices (of which the upper one may be a non-convex vertex of P that is convex in P'), as they vertically connect to the upper string.

Then, the proof recurses on the new polygon P', of which the convex vertices of the upper string(s) need to be fixed. Note that P' may have convex vertices that were non-convex in P. These need not be fixed, as they were already proven immobilized since they are part of the vertical edge of P'.

This proves the theorem. Note that convex vertices where the negative gravity vector points into the polygon will never be part of an upper string at any phase of the proof, so they need not be fixed. Also, the recursion "terminates." This can be seen by considering the vertical trapezoidal decomposition of the original polygon P, which contains a finite number of trapezoids. In each recursion step, at least one trapezoid is removed from P, until the entire polygon has proven immobilized. □

A *g-fold* (g refers to gravity) is specified by a directed line segment in the plane whose endpoints lie on the boundary of the cloth polygon. The segment partitions the polygon into two parts, one to be folded over another (see Fig. 4). A g-fold is successfully achieved when the part of the polygon to the *left* of the directed line segment is folded across the line segment and placed horizontally on top of the other part, while maintaining the following property:

- At all times during a folding procedure, every part of the cloth is either horizontal or vertical, and the grippers hold points on the vertical part such that it is immobilized (see Fig. 5).

This ensures that the cloth is in a fully predictable configuration according to our material model at all times during the folding procedure. Not all folds can be achieved using a g-fold; in terms of [1], *valley folds* can be achieved using a g-fold, but *mountain folds* cannot.

A *g-fold sequence* is a sequence of g-folds as illustrated in Fig. 1. After the initial g-fold, the *stacked* geometry of cloth allows us to specify two types of g-fold: a “red” g-fold and a “blue” g-fold. A blue g-fold is specified by a line segment partitioning the polygon formed by the *silhouette* of the stacked geometry into two parts, and is successfully achieved by folding the (entire) geometry left of the line segment. A red g-fold is similarly specified, but only applies to the (potentially stacked) geometry that was folded in the previous g-fold (see Fig. 4).

We are given a robot with k point grippers that can grasp the cloth at any point on the *boundary* of the polygon formed by the silhouette of the stacked geometry. At each such point, the gripper will grasp all layers of the stack at that point (i.e., it is not capable of distinguishing between layers). Each of the grippers is able to move independently above the xy-plane and we assume that gripper motion is exact.

The problem we discuss in this paper is then defined as follows. Given a specification of a sequence of g-folds, determine whether each of the folds are feasible given the number of grippers available, and if so, compute the number of grippers needed and the manipulation motion for each of the grippers to achieve the g-folds.

4 Planning G-Folds

4.1 Single G-Folds on Unstacked Geometry

We first discuss the case of performing a single g-fold of the original (unstacked) polygon. During the manipulation, the cloth must be separated in a vertical part and a horizontal part at all times. The line separating the vertical part and the horizontal part is called the *baseline*.

Given a polygonal cloth and a specification of a g-fold by a directed line segment (e.g. the first g-fold of Fig. 4(a)), we plan the manipulation as follows. The manipulation consists of two phases: in the first phase, the cloth is manipulated such that the part that needs to be folded is brought vertical above the line segment

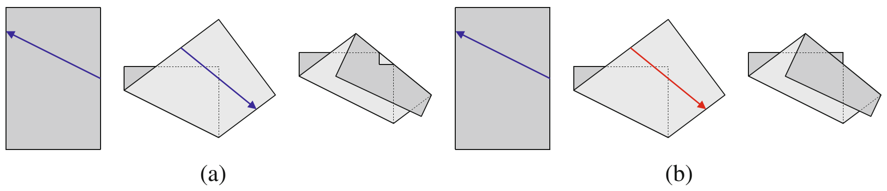


Fig. 4 (a) A g-fold is specified by a directed line segment partitioning the (stacked) geometry into two parts. The g-fold is successfully achieved when the part of the geometry left of the line segment is folded around the line segment. A sequence of two g-folds is shown here. **(b)** A g-fold sequence similar to (a), but the second g-fold (a red g-fold) is specified such that it only applies to the part of the cloth that was folded in the previous g-fold.

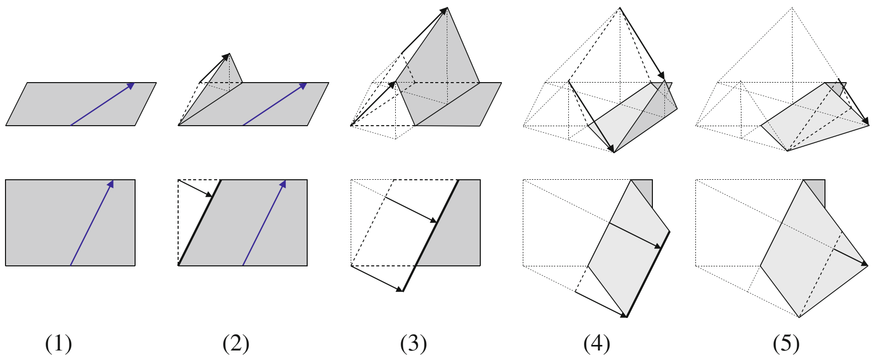


Fig. 5 The motion of two grippers (arrows) successfully performing the first g-fold specified in Fig. 4(a) shown both in 3-D view and top-view. At all times during the manipulation, all parts of the cloth are either vertical or horizontal and immobilized. The boundary between the vertical part and the horizontal part of the cloth is called the *baseline*.

specifying the g-fold (see Figs. 5-(1), 5-(2), and 5-(3)). In the second phase, the g-fold is completed by manipulating the cloth such that the vertical part is laid down on the surface with its original normal reversed (Figs. 5-(3), 5-(4), and 5-(5)).

The first phase is carried out as shown in Figs. 5-(1), 5-(2) and 5-(3), manipulating the cloth such that the baseline of the vertical part is parallel to the line segment at all times. Initially, the "baseline" is outside the cloth polygon (meaning that there is no vertical part) and is moved linearly towards the line segment specifying the g-fold. In the second phase, the g-fold is completed by laying down the vertical part of the cloth using a mirrored manipulation in which the baseline is again parallel to the line segment at all times. Initially the baseline is at the line segment specifying the g-fold and in moved linearly outward until the baseline is outside the folded part of the polygon (see Figs. 5-(3), 5-(4) and 5-(5)).

The corresponding motions of the grippers holding the vertices can be computed as follows. Let us assume without loss of generality that the line segment specifying the g-fold coincides with the x-axis and points in the positive x-direction. Hence, the part of the polygon above the x-axis needs to be folded. Each convex vertex of this part in which the positive y-vector points outside of the cloth in its initial configuration needs to be held by a gripper at some point during the manipulation. We denote this set of vertices by V and can be computed in $O(n)$ time. Let y^* be the maximum of the y-coordinates of the vertices in V. Now, we let the baseline, which is parallel to the x-axis at all times, move "down" with speed 1, starting at $y_b = y^*$, where y_b denotes the y-coordinate of the baseline. Let the initial planar coordinates of a vertex $v \in V$ be (x_v, y_v). As soon the baseline passes y_v, vertex v starts to be manipulated. When the baseline passes $-y_v$, vertex v stops to be manipulated. During the manipulation, the vertex is held precisely above the baseline. In general, the 3-D coordinate $(x(y_b), y(y_b), z(y_b))$ of the gripper holding vertex v as a function of the y-coordinate of the baseline is:

$$x(y_b) = x_v, \qquad y(y_b) = y_b, \qquad z(y_b) = y_v - |y_b|, \tag{1}$$

for $y_b \in [y_v, -y_v]$. Outside of this interval, the vertex is part of the horizontal part of the cloth and does not need to be grasped by a gripper. This reasoning applies to all vertices $v \in V$. When the baseline has reached $-y^*$, all vertices have been laid down and the g-fold is completed. As a result, we do not need to grasp any vertex outside of V at any point during the manipulation, and the motions of the vertices in V can be computed in $O(n)$ time.

4.2 Sequences of G-Folds and Stacked Geometry

We now discuss the case of folding already folded geometry. First, we discuss how to represent folded, stacked geometry. Let us look at the example of the longsleeve t-shirt of Fig. 1, and in particular at the geometry of the cloth after five g-folds. The creases of the folds have subdivided the original polygon into facets (see Fig. 6(a)). With each such facet, we maintain two values: an integer indicating the height of the facet in the stacked geometry (1 is the lowest) and a transformation matrix indicating how the facet is transformed from the original geometry to the folded geometry. Each transformation matrix is a product of a subset of the matrices F_i that each correspond to the mirroring in the line segment specifying the i'th g-fold. In Fig. 6(b), we show the lines of each of the g-folds with the associated matrix F_i.

Given the representation of the current stacked geometry and a line segment specifying a new g-fold, we show how we manipulate the cloth to successfully perform the g-fold or report that the g-fold is infeasible. We assume that the line segment specifying the g-fold partitions the silhouette of the stacked geometry into two parts (i.e., a blue g-fold). Let us look at the sixth specified g-fold in the longsleeve t-shirt example, which folds the geometry of Fig. 6.

Each facet of the geometry (in its folded configuration) is either fully to the left of the line segment, fully to the right, or intersected by the line segment specifying the g-fold. The facets intersected by the line segment are subdivided into two new

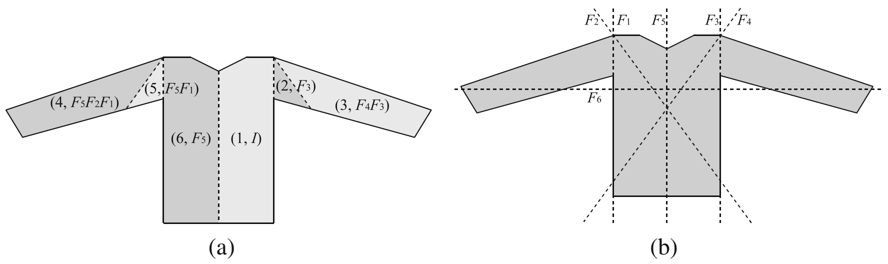
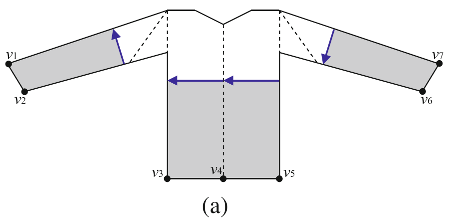


Fig. 6 **(a)** The representation of folded stacked geometry. The example shown here is the longsleeve t-shirt of Fig. 1 after five g-folds. With each facet, the stack height (integer) and a transformation matrix is stored. **(b)** Each transformation matrix F_i corresponds to mirroring the geometry in the line segment specifying the i'th g-fold.



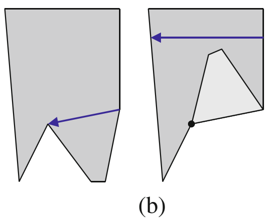


Fig. 7 (a) The geometry in the longsleeve t-shirt example after subdividing the facets by the line segment specifying the sixth g-fold. The gray facets need to be folded. The convex vertices for which the negative gravity vector points outside of the facet are shown using dots. **(b)** The vertex marked by the dot need not be held by a gripper to perform the fold, as it is non-convex in the dark gray facet (even though it is convex in the light gray facet).

facets, both initially borrowing the data (the stack height and the transformation matrix) of the original facet. Now, each facet will either be folded, or will not be folded. Fig. 7(a) shows the new geometry in the longsleeve t-shirt example after subdividing the facets by the line segment specifying the g-fold. The gray facets need to be folded.

As in the case of folding planar geometry, for each facet each convex vertex at which the gravity vector points outside of the facet at the time it is above the line segment specifying the g-fold should be held by a gripper, and each non-convex vertex and each convex vertex at which the negative gravity vector points inside the facet need not be held by a gripper. If multiple facets share a vertex, and according to at least one facet it needs not be held by a gripper, it does not need to be held by a gripper (see Fig. 7(b)).

For the t-shirt example, the vertices that need to be grasped are shown using dots in Fig. 7(a) and labeled $v_1, \ldots, v_7$. Applying the transformation matrices stored with the incident facet to each of the vertices shows that v_1, v_3, v_5, and v_7 will coincide in the plane. As a gripper will grasp all layers the geometry, only one gripper is necessary to hold these vertices. Vertex v_4 also needs to be held by gripper. Vertices v_2 and v_6 remain, but they need *not* be grasped. This is for the following reason. As can be seen in Fig. 1, these vertices are fully *covered*. That is, the vertex is "hidden" behind other facets of the cloth both below and above it in the stacked geometry. As we assume that the friction between two pieces of the cloth is infinite, this vertex will not be able to move as a result of gravity, and need not be grasped. Using the heights stored at each facet, we can compute for each vertex whether it is covered or not.

This defines fully what vertices need to be grasped to achieve a g-fold of stacked geometry. If any such vertex is not on the boundary of the silhouette of the stacked geometry, the g-fold is infeasible (for example, the second g-fold of Fig. 4 (a) is infeasible for this reason). The 3-D motion of the grippers can be computed in the same way as for planar geometry, as discussed in Section 4.1. The running time for computing the vertices that need to be grasped is in principle exponential in the number of g-folds that preceeded, as in the worst case i g-folds create 2^i facets. If we consider the number of g-folds a constant, the set of vertices that need to be grasped can be identified in $O(n)$ time.

After the g-fold is executed, we need to update the data fields of the facets that were folded in the geometry: each of their associated transformation matrices is *pre*-multiplied by the matrix F_i corresponding to a mirroring in the line segment specifying the g-fold (F_6 in Fig. 6(b) for the t-shirt example). The stack height of these facets is updated as follows: the order of the heights of all facets that are folded are *reversed* and put on top of the stack. In the example of Fig. 7(a), the facets that are folded have heights 4, 6, 1, and 3 before the g-fold, and heights 8, 7, 10, and 9 after the g-fold, respectively.

The above procedure can be executed in series for a sequence of g-folds. Initially, the geometry has one facet (the original polygon) with height 1 and transformation matrix I (the identity matrix). If a g-fold is specified to only apply to the folded part of the geometry of the last g-fold (a “red” g-fold), the procedure is the same, but only applies to those facets that were folded in the last g-fold. We allow these kinds of g-folds as a special primitive if they need the same set of vertices to be grasped as the previous g-fold. Even if the vertices that are grasped are not on the boundary of the silhouette of the geometry, the g-fold can be achieved by not releasing the vertices after the previous g-fold. This enriches the set of feasible fold primitives.

5 Experiments

5.1 Experimental Setup

We used a Willow Garage PR2 robotic platform [15], shown in Fig. 8. The PR2 has two articulated 7-axis arms with parallel jaw grippers. We used a soft working

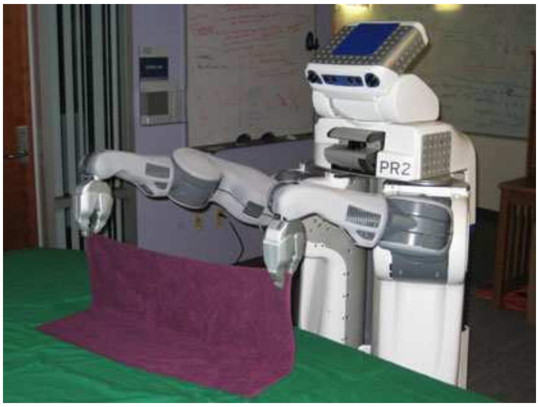


Fig. 8 The PR2 robotic platform (developed by Willow Garage) performing a g-fold on a towel.

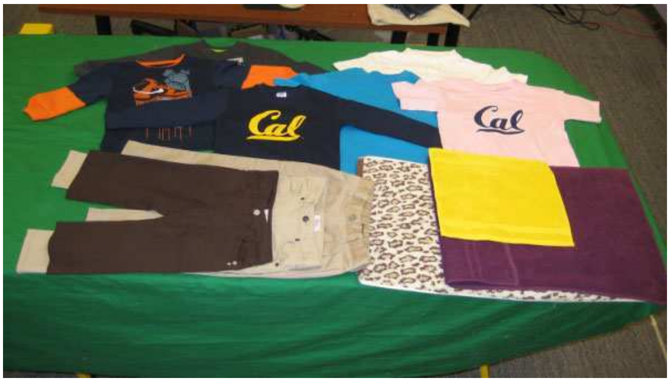


Fig. 9 Three of each clothing category were used in conducting our experiments.

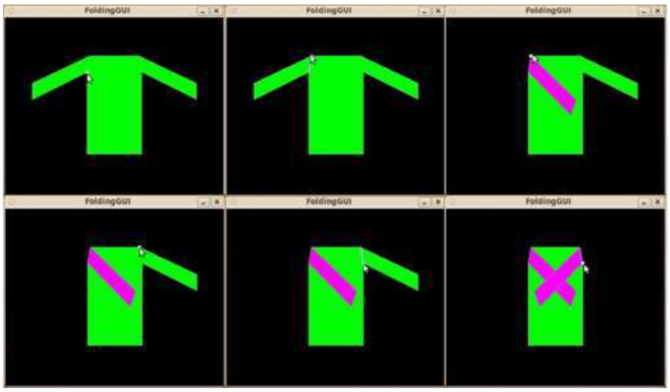


Fig. 10 An example sequence of user-specified folds. The user first clicks on the left arm-pit, then on the left shoulder to specify the first fold. The GUI then verifies that this is a valid g-fold for the chosen number of grippers. In this case it is, and it then shows the result after executing the g-fold (3rd image in the top row). Then the user specifies the next fold by two clicks, the program verifies whether it's a valid g-fold, and then shows the result after executing the g-fold.

surface, so the relatively thick grippers can easily get underneath the cloth. Our approach completely specifies end-effector position trajectories. It also specifies the orientation of the parallel jaw grippers' planes. We used a combination of native IK tools and a simple linear controller to plan the joint trajectories.

We experimented with the clothing articles shown in Fig. 9. Whenever presented with a new, spread-out clothing article, a human user clicks on the vertices of the article in an image. The user is then presented with a GUI that allows them to specify a sequence of folds achievable with the two grippers. Once a valid g-fold has been specified, the GUI executes the fold and the user can enter the next fold. Fig. 10 illustrates the fold sequence specification process through an example.

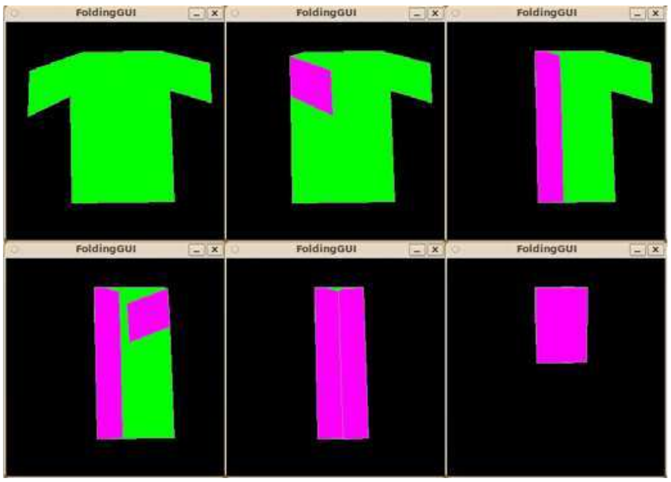


Fig. 11 An example folding primitive, automatically executed on a T-shirt polygon. Note the clean fold, despite the imperfect symmetry of the original polygon.

As many sophisticated folds are difficult, if not impossible, to perfectly define by hand, the GUI is also seeded with a set of folding primitives. When presented with a particular article of clothing, the user is given the option of calling one of these primitives. Once called, a sequence of folds is computed, parametrized on a number of features such as scaling, rotation, and side lengths. Fig. 11 shows an example primitive being executed on a user-defined polygon in the shape of a shirt. To ensure consistency across multiple trials, such primitives were used to execute the folds detailed in the Experimental Results section below.

While our approach assumes the cloth has zero resistance against bending, real cloth does indeed resist against bending. As a consequence, our approach outlined so far overestimates the number of grippers required to hold a piece of cloth in a predictable, spread-out configuration. Similarly, our robot grippers have non-zero size, also resulting in an overestimation of the number of grippers required. To account for both of these factors, our implementation offers the option to allocate a radius to each of our grippers, and we consider a point being gripped whenever it falls inside this radius. To compute the grip points, we first compute the grip points required for point grippers and infinitely flexible cloth. We then cluster these points using a simple greedy approach. We begin by attempting to position a circle of fixed radius in the coordinate frame such that it covers the maximum number of grip points, while subsequently minimizing the average distance from each covered point to its center. This process is iterated until no point remains uncovered. For the duration of the fold, our grippers now follow the trajectory of the center of each cluster, rather than individual grip points.

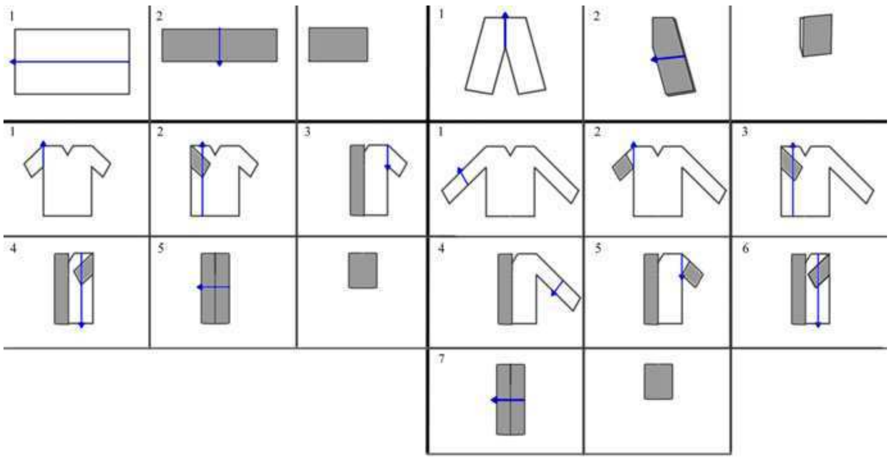


Fig. 12 The user-requested sequences of folds used in our experiments.

5.2 Experimental Results

We tested our approach on four categories: towels, pants, short-sleeved shirts, and sweaters. Fig. 12 shows the fold sequences used for each category. To verify robustness of our approach, we tested on three instances of each category of clothing. These instances varied in size, proportion, thickness, and texture. At the beginning of each experimental trial, we provided the PR2 with the silhouette of the polygon through clicking on the vertices in two stereo images.

Fig. 13 shows the robot going through a sequence of folds. Table 1 shows success rates and timing on all clothing articles. As illustrated by these success rates, our method demonstrates a consistent level of reliability on real cloth, even when the manipulated fabric notably strays from the assumptions of our model. For instance, the g-fold method worked reasonably well on pants, despite the material's clear violation of the assumption of non-zero thickness, and a three-dimensional shape which was not quite polygonal. It was also able to fold a collared shirt with perfect

Table 1 Experimental results of autonomous cloth folding.

Category	Success rate	Avg time (s)	Category	Success rate	Avg time (s)
Towels	**9/9**	**200.0**	**Short-Sleeved Shirts**	**7/9**	**337.6**
Purple	3/3	215.6	Pink T-Shirt	2/3	332.8
Leopard	3/3	210.9	Blue T-Shirt	2/3	343.2
Yellow	3/3	173.5	White Collared	3/3	337.6
Pants	**7/9**	**186.6**	**Long-Sleeved Tops**	**5/9**	**439.0**
Long Khaki	3/3	184.9	Long-Sleeved Shirt	2/3	400.7
Brown	1/3	185.9	Gray Sweater	1/3	458.4
Short Khaki	3/3	189.1	Blue Sweater	2/3	457.8

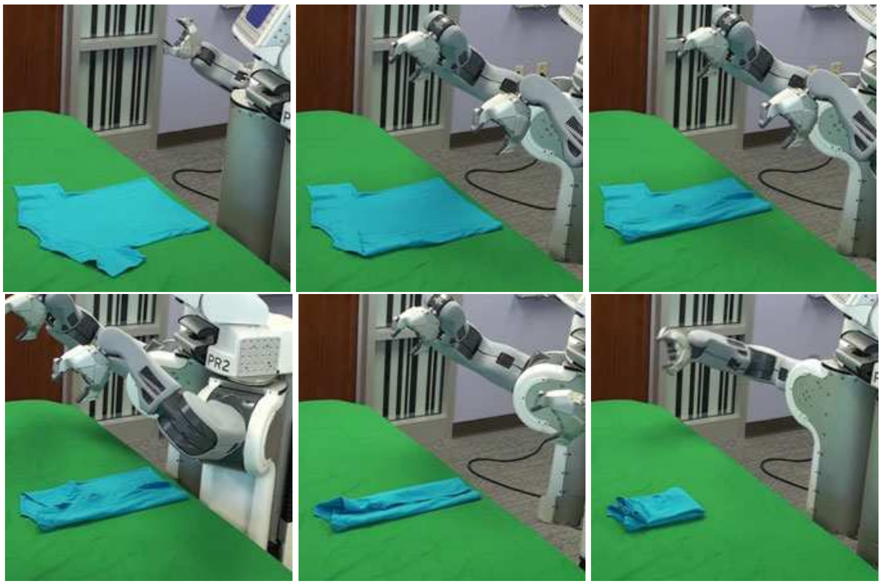


Fig. 13 The robot folding a t-shirt using our approach.

accuracy, despite a rigid collar and buttons, neither of which are expressible in the language of our model. This level of practical success is indicative of a certain robustness to our approach, which lends itself to a number of implications. The first is that despite the simplifications inherent to our model, real cloth behaves deterministically under roughly the same conditions as its ideal counterpart. While human manipulation of cloth exploits a number of features which our model neglects, these features generally arise in states which our model considers unreachable. That is, handling true fabric often requires less caution than our g-fold model predicts, but rarely does it require more. The second is that even when unpredicted effects do arise, the final result is not compromised. Although factors such as thickness may cause the cloth to deviate slightly from its predicted trajectory – most often in the form of "clumping" for thick fabrics – the resulting fold generally agrees with the model, particularly after smoothing.

Much of our success can be attributed to a number of assumptions which real cloth very closely met: namely, the infinite friction between the cloth and the table, and the infinite friction between the cloth and itself. The former allowed us to execute g-folds even when the modeled polygon did not perfectly match the silhouette of the cloth. As actual articles of clothing are not comprised solely of well-defined corners, this imprecision often resulted in a nonzero horizontal tension in the cloth during the folding procedure. However, as the friction between the cloth and the table far outweighs this tension, the cloth remained static. The latter allowed us to stabilize loose vertices by "sandwiching" them between two gripped portions of cloth. This technique, in combination with the robust gripping approach detailed above, allowed us to execute a number of folds (such as the shirt folds in Fig. 12) which more closely resembled their standard human counterpart. With the exception of long-sleeved shirts, all sequences could be perfectly executed by a pair of point grippers. However, some relied on the ability to create perfect 90 degree angles, or perfectly align two edges which (in actuality) were not entirely straight. Exact precision was impossible in both of these cases; but where there was danger of gravity influencing a slightly unsupported vertex, the friction of the cloth, in conjunction with its stiffness, often kept it in a stable configuration.

The trials were not, however, without error. Most often, failure was due to the limitations of our physical control, rather than a flaw in our model. For instance, 2/2 Short-Sleeved failures and 3/4 Long-Sleeved failure occurred at steps where the robot was required to grasp a portion of previously folded sleeve (Short-Sleeve Steps 2 and 4, Long-Sleeve Steps 3 and 6 in Fig. 12) In all cases, the failure could be easily predicted from the location of the initial grasp. Either the robot did not reach far enough and grasped nothing, or reached too far and heavily bunched the cloth. These failures suggest a clear issue with our implementation: namely, the reliance on open-loop control. While the initial position of each vertex is given, the location of a folded vertex must be derived geometrically. For this location to be correct, we must make two assumptions: that the cloth at hand is perfectly represented by the given polygon, and that the trajectory, once computed, can be exactly followed. Clearly, both are idealizations: the former disregards the multi-layered nature of all but towels (which saw a 100% success rate) and the latter is hindered by the inherent imprecision of any robotic mechanism. A closed-loop method, which would allow the robot to adjust the shape of the modeled polygon to allow for real-world discrepancies, would likely eliminate these issues.

In addition, the robot moved very slowly in our trials, leading to large running times. The reason for moving slowly, besides the fact that it helps the cloth behave in a more deterministic fashion (no dynamics effects), is that it led to higher accuracy of the motions of the arms and base of the robot, which was required for open-loop control. Also here, a closed-loop method would likely enable performance improvements.

Videos of our experimental results are available at http://rll.berkeley.edu/wafr10-gfolds/.

6 Conclusion and Future Work

We described a geometric approach to cloth folding—avoiding the difficulties with physical simulation of cloth. To do so, we restrict attention to a limited subset of cloth configuration space. Our experiments show that (i) this suffices to capture interesting folds, and (ii) real cloth behaves benignly, even when moderately violating our assumptions. Our approach enabled reliable folding of a wide variety of clothing articles.

In the experiments we performed, a human user had to click on the vertices of the clothing article, and would then select a fold sequence. We plan to develop computer vision algorithms that enable automatic recognition of clothing article categories, as well as specific geometric instances thereof. The robot could then look up the desired folding sequence for the presented article. We are also working on simple primitives that will enable taking clothing articles from crumpled, unknown configurations to the spread-out configuration.

Careful inspection of the gripper paths shows that a single very large parallel jaw gripper would suffice to execute a g-fold requiring an arbitrary number of point grippers. We plan to investigate a practical implementation of this idea for the PR2. A large such gripper would reduce the collision free workspace volume significantly.

Acknowledgements. This research was supported in part by the National Science Foundation (NSF) under grants IIS-0905344 and IIS-0904672, by the National Institute of Health (NIH) under grant R01-EB006435, and by Willow Garage by providing the PR-2 robot.

References

1. Balkcom, D., Mason, M.: Robotic origami folding. Int. J. on Robotics Research 27(5), 613–627 (2008)
2. Baraff, D., Witkin, A.: Large steps in cloth simulation. In: Proc. SIGGRAPH (1998)
3. Bell, M.: Flexible object manipulation. PhD Thesis, Dartmouth College (2010)
4. Bell, M., Balkcom, D.: Grasping non-stretchable cloth polygons. Int. Journal of Robotics Research 29, 775–784 (2010)
5. Bridson, R., Fedkiw, R., Anderson, J.: Robust treatment of collisions, contact, and friction for cloth animation. In: Proc. SIGGRAPH (2002)
6. Choi, K., Ko, H.: Stable but responsive cloth. In: Proc. SIGGRAPH (2002)
7. Fahantidis, N., Paraschidis, K., Petridis, V., Doulgeri, Z., Petrou, L., Hasapis, G.: Robot handling of flat textile materials. IEEE Robotics and Automation Magazine 4(1), 34–41 (1997)
8. Gupta, S., Bourne, D., Kim, K., Krishnan, S.: Automated process planning for robotic sheet metal bending operations. Journal of Manufacturing Systems 17, 338–360 (1998)
9. Kanade, T.: A theory of origami world. Artificial Intelligence 13(3), 279–311 (1980)
10. Lang, R.: A computational algorithm for origami design. In: Proc. Symp. on Computational Geometry (1996)
11. Liu, J., Dai, J.: An approach to carton-folding trajectory planning using dual robotic fingers. Robotics and Autonomous Systems 42, 47–63 (2003)
12. Maitin-Shepard, J., Cusumano-Towner, M., Lei, J., Abbeel, P.: Cloth grasp point detection based on multiple-view geometric cues with application to robotic towel folding. In: Proc. IEEE Int. Conf. on Robotics and Automation (2010)
13. Lu, L., Akella, S.: Folding cartons with fixtures: a motion planning approach. IEEE Trans. on Robotics and Automation 16(4), 346–356 (2000)
14. Osawa, F., Seki, H., Kamiya, Y.: Clothes folding task by tool-using robot. Journal of Robotics and Mechatronics (2006)
15. Wyrobek, K., Berger, E., Van der Loos, H.F.M., Salisbury, K.: Towards a Personal Robotics Development Platform: Rationale and Design of an Intrinsically Safe Personal Robot. In: Proc. IEEE Int. Conf. on Robotics and Automation (2008)

Author Index