



springer tracts in advanced robotics 86

**Emilio Frazzoli
Tomás Lozano-Pérez
Nicholas Roy
Daniela Rus (Eds.)**

Algorithmic Foundations of Robotics X

**Proceedings of the Tenth Workshop on the Algorithmic
Foundations of Robotics**

Editors

Prof. Bruno Siciliano
Dipartimento di Ingegneria Elettrica
e Tecnologie dell'Informazione
Università degli Studi di Napoli
Federico II
Via Claudio 21, 80125 Napoli
Italy
E-mail: siciliano@unina.it

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

Editorial Advisory Board

Oliver Brock, TU Berlin, Germany
Herman Bruyninckx, KU Leuven, Belgium
Raja Chatila, ISIR - UPMC & CNRS, France
Henrik Christensen, Georgia Tech, USA
Peter Corke, Queensland Univ. Technology, Australia
Paolo Dario, Scuola S. Anna Pisa, Italy
Rüdiger Dillmann, Univ. Karlsruhe, Germany
Ken Goldberg, UC Berkeley, USA
John Hollerbach, Univ. Utah, USA
Makoto Kaneko, Osaka Univ., Japan
Lydia Kavraki, Rice Univ., USA
Vijay Kumar, Univ. Pennsylvania, USA
Sukhan Lee, Sungkyunkwan Univ., Korea
Frank Park, Seoul National Univ., Korea
Tim Salcudean, Univ. British Columbia, Canada
Roland Siegwart, ETH Zurich, Switzerland
Gaurav Sukhatme, Univ. Southern California, USA
Sebastian Thrun, Stanford Univ., USA
Yangsheng Xu, Chinese Univ. Hong Kong, PRC
Shin'ichi Yuta, Tsukuba Univ., Japan

STAR (Springer Tracts in Advanced Robotics) has been promoted under the auspices of EURON (European Robotics Research Network)



Emilio Frazzoli, Tomás Lozano-Pérez, Nicholas Roy,
and Daniela Rus (Eds.)

Algorithmic Foundations of Robotics X

Proceedings of the Tenth Workshop
on the Algorithmic Foundations of Robotics



Editors

Prof. Emilio Frazzoli
Massachusetts Institute of Technology
Cambridge
USA

Prof. Tomás Lozano-Pérez
Massachusetts Institute of Technology
Cambridge
USA

Prof. Nicholas Roy
Massachusetts Institute of Technology
Cambridge
USA

Prof. Daniela Rus
Massachusetts Institute of Technology
Cambridge
USA

ISSN 1610-7438
ISBN 978-3-642-36278-1
DOI 10.1007/978-3-642-36279-8
Springer Heidelberg New York Dordrecht London

e-ISSN 1610-742X
e-ISBN 978-3-642-36279-8

Library of Congress Control Number: 2012956289

© Springer-Verlag Berlin Heidelberg 2013

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed. Exempted from this legal reservation are brief excerpts in connection with reviews or scholarly analysis or material supplied specifically for the purpose of being entered and executed on a computer system, for exclusive use by the purchaser of the work. Duplication of this publication or parts thereof is permitted only under the provisions of the Copyright Law of the Publisher's location, in its current version, and permission for use must always be obtained from Springer. Permissions for use may be obtained through RightsLink at the Copyright Clearance Center. Violations are liable to prosecution under the respective Copyright Law.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Foreword

Robotics is undergoing a major transformation in scope and dimension. From a largely dominant industrial focus, robotics is rapidly expanding into human environments and vigorously engaged in its new challenges. Interacting with, assisting, serving, and exploring with humans, the emerging robots will increasingly touch people and their lives.

Beyond its impact on physical robots, the body of knowledge robotics has produced is revealing a much wider range of applications reaching across diverse research areas and scientific disciplines, such as: biomechanics, haptics, neurosciences, 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 that 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 tenth edition and is edited by Emilio Frazzoli, Tomás Lozano-Pérez, Nicholas Roy and Daniela Rus. The book offers a collection ranging from foundational topics in robot motion planning, control and perception, through well-established topics in multi-agent systems, distributed robotics and planning under uncertainty, to emerging topics in minimalistic robotics, interaction with the environment, and behaviour of groups of humans or human-controlled vehicles.

The contents of the thirty-seven 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 control theory, computational geometry and topology, geometrical and physical modelling, reasoning under uncertainty, probabilistic algorithms, game theory, and theoretical computer science. 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!

December 2012
Naples, Italy

Bruno Siciliano
STAR Editor

Preface

Algorithms are a fundamental component of robotic systems. Robot algorithms process inputs from sensors that provide noisy and partial data, build geometric and physical models of the world, plan high-and low-level actions at different time horizons, and execute these actions on actuators with limited precision. The design and analysis of robot algorithms raise a unique combination of questions from many fields, including control theory, computational geometry and topology, geometrical and physical modeling, reasoning under uncertainty, probabilistic algorithms, game theory, and theoretical computer science.

The Workshop on Algorithmic Foundations of Robotics (WAFR) is a single-track meeting of leading researchers in the field of robot algorithms. Since its inception in 1994, WAFR has been held every other year, and has provided one of the premiere venues for the publication of some of the field's most important and lasting contributions, ensured the rapid circulation of new ideas, and served as a catalyst for productive collaborations.

The focus of WAFR is on the design and analysis of robot algorithms from both theoretical and practical angles. The topics of interest are very broad, as exemplified by the program of its tenth edition, in 2012. In addition to several papers on fundamental algorithmic issues in robot motion planning, control, and perception, and in well-established areas such as multi-agent systems and distributed robotics, planning under uncertainty, and sampling-based algorithms, several papers concentrate on emerging areas including minimalistic robotics, interaction with the environment, and the use of robotics-inspired algorithms to model and interpret the behavior of groups of humans or human-controlled vehicles.

The tenth edition of WAFR was held on June 13–15 2012 at the Massachusetts Institute of Technology, in Cambridge, Massachusetts. It had a strong program of 37 contributed papers, selected from 74 submissions from leading researchers worldwide. Each paper was rigorously reviewed by at least three program committee members, with additional input from external reviewers. The authors of selected papers were invited to submit expanded versions of their WAFR 2012 papers to special issues of the International Journal of Robotics Research and of the IEEE Transactions on Automation Science and Engineering. The workshop also featured

5 invited speakers: Bruce Donald (Duke University), Geoff Gordon (Carnegie-Mellon University), David Hsu (National University of Singapore), Andreas Krause (ETH Zürich), and Antonio Torralba (Massachusetts Institute of Technology).

Organizing this workshop was an exciting experience, allowing us to work in close contact with some of the most talented roboticists worldwide. We are grateful to the WAFR steering committee for their inspiring guidance, to the program committee for their dedicated efforts in ensuring the highest quality standards, and above all to the authors for submitting their cutting-edge work and to the participants for ensuring the success of the workshop.

Emilio Frazzoli
Tomás Lozano-Pérez
Nicholas Roy
Daniela Rus

Organization

Steering Committee

| | |
|-----------------------|--|
| Nancy Amato | Texas A&M University |
| Daniela Rus | Massachusetts Institute of Technology |
| Oliver Brock | Technische Universität Berlin |
| David Hsu | National University of Singapore |
| Ken Goldberg | University of California, Berkeley |
| Seth Hutchinson | University of Illinois, Urbana-Champaign |
| Lydia Kavraki | Rice University |
| Frank van der Stappen | Universiteit Utrecht |
| Jean-Claude Latombe | Stanford University |

Program Committee

| | |
|-------------------|---|
| Ron Alterovitz | University of North Carolina, Chapel Hill |
| Nancy Amato | Texas A&M University |
| J. Andrew Bagnell | Carnegie Mellon University |
| Devin Balkcom | Dartmouth College |
| Kostas Bekris | Rutgers University |
| Antonio Bicchi | University of Pisa |
| Emma Brunskill | Carnegie Mellon University |
| Howie Choset | Carnegie Mellon University |
| Nikolaus Correll | University of Colorado, Boulder |
| Jorge Cortes | University of California, San Diego |
| Carrick Detweiler | University of Nebraska-Lincoln |
| Michael Erdmann | Carnegie Mellon University |
| Ryan Eustice | University of Michigan |
| David Hsu | National University of Singapore |
| Volkan Isler | University of Minnesota |
| Vijay Kumar | University of Pennsylvania |

| | |
|----------------------|---|
| Jean Paul Laumond | LAAS-CNRS, Toulouse |
| Steven M. LaValle | University of Illinois, Urbana-Champaign |
| Dinesh Manocha | University of North Carolina, Chapel Hill |
| Sonia Martinez | University of California, San Diego |
| Nathan Michael | Carnegie Mellon University |
| Todd Murphrey | Northwestern University |
| Marco Pavone | Stanford University |
| Thierry Simeon | LAAS-CNRS, Toulouse |
| Surya Singh | University of Queensland |
| Stephen Smith | University of Waterloo |
| Siddhartha Srinivasa | Carnegie Mellon University |
| Cyrill Stachniss | University of Freiburg |
| Subhash Suri | University of California, Santa Barbara |
| Jur Vandenberg | University of Utah |

Contents

| | |
|---|-----|
| The Minimum Constraint Removal Problem with Three Robotics Applications | 1 |
| <i>Kris Hauser</i> | |
| Hierarchical Decision Theoretic Planning for Navigation Among Movable Obstacles | 19 |
| <i>Martin Levihn, Jonathan Scholz, Mike Stilman</i> | |
| Robust Complete Path Planning in the Plane | 37 |
| <i>Victor Milenkovic, Elisha Sacks, Steven Trac</i> | |
| Configurations and Path Planning of Convex Planar Polygonal Loops | 53 |
| <i>Li Han, Lee Rudolph, Michael Chou, Sean Corbett, Emily Eagle, Dylan Glotzer, Jake Kramer, Jonathan Moran, Christopher Pietras, Ammar Tareen, Matthew Valko</i> | |
| Equilibrium Configurations of a Kirchhoff Elastic Rod under Quasi-static Manipulation | 71 |
| <i>Timothy Bretl, Zoe McCarthy</i> | |
| From Discrete to Continuous Motion Planning | 89 |
| <i>Nicolas Perrin</i> | |
| Ray-Shooting Algorithms for Robotics | 105 |
| <i>Yu Zheng, Katsu Yamane</i> | |
| Optimal Gap Navigation for a Disc Robot | 123 |
| <i>Rigoberto Lopez-Padilla, Rafael Murrieta-Cid, Steven M. LaValle</i> | |
| Min-Max Latency Walks: Approximation Algorithms for Monitoring Vertex-Weighted Graphs | 139 |
| <i>Soroush Alamdari, Elaheh Fata, Stephen L. Smith</i> | |

| | |
|---|-----|
| Multi-agent Path Planning and Network Flow | 157 |
| <i>Jingjin Yu, Steven M. LaValle</i> | |
| Trajectory Planning and Assignment in Multirobot Systems | 175 |
| <i>Matthew Turpin, Nathan Michael, Vijay Kumar</i> | |
| <i>k</i>-Color Multi-robot Motion Planning | 191 |
| <i>Kiril Solovey, Dan Halperin</i> | |
| Distributed Construction of Truss Structures | 209 |
| <i>Quentin Lindsey, Vijay Kumar</i> | |
| Space-Time Group Motion Planning | 227 |
| <i>Ioannis Karamouzas, Roland Geraerts, A. Frank van der Stappen</i> | |
| Multi-robot Coverage and Exploration in Non-Euclidean Metric Spaces | 245 |
| <i>Subhrajit Bhattacharya, Robert Ghrist, Vijay Kumar</i> | |
| Lion and Man with Visibility in Monotone Polygons | 263 |
| <i>Narges Noori, Volkan Isler</i> | |
| Sparse Roadmap Spanners | 279 |
| <i>Andrew Dobson, Athanasios Krontiris, Kostas E. Bekris</i> | |
| Toggle PRM: A Coordinated Mapping of C-Free and C-Obstacle in Arbitrary Dimension | 297 |
| <i>Jory Denny, Nancy M. Amato</i> | |
| On the Power of Manifold Samples in Exploring Configuration Spaces and the Dimensionality of Narrow Passages | 313 |
| <i>Oren Salzman, Michael Hemmer, Dan Halperin</i> | |
| Sampling Extremal Trajectories for Planar Rigid Bodies | 331 |
| <i>Weifu Wang, Devin Balkcom</i> | |
| Convex Hull Asymptotic Shape Evolution | 349 |
| <i>Maxim Arnold, Yuliy Baryshnikov, Steven M. LaValle</i> | |
| Efficient Collision Checking in Sampling-Based Motion Planning | 365 |
| <i>Joshua Bialkowski, Sertac Karaman, Michael Otte, Emilio Frazzoli</i> | |
| Faster Sample-Based Motion Planning Using Instance-Based Learning | 381 |
| <i>Jia Pan, Sachin Chitta, Dinesh Manocha</i> | |

| | |
|--|-----|
| Scale-Free Coordinates for Multi-robot Systems with Bearing-Only Sensors | 397 |
| <i>Alejandro Cornejo, Andrew J. Lynch, Elizabeth Fudge, Siegfried Bilstein, Majid Khabbazian, James McLurkin</i> | |
| Mapping Polygons with Agents That Measure Angles | 415 |
| <i>Yann Disser, Matúš Mihalák, Peter Widmayer</i> | |
| Counting Moving Bodies Using Sparse Sensor Beams | 427 |
| <i>Lawrence H. Erickson, Jingjin Yu, Yaonan Huang, Steven M. LaValle</i> | |
| Convex Receding Horizon Control in Non-Gaussian Belief Space | 443 |
| <i>Robert Platt</i> | |
| Planar Uncertainty Propagation and a Probabilistic Algorithm for Interception | 459 |
| <i>Andrew W. Long, Kevin C. Wolfe, Gregory S. Chirikjian</i> | |
| Intention-Aware Motion Planning | 475 |
| <i>Tirthankar Bandyopadhyay, Kok Sung Won, Emilio Frazzoli, David Hsu, Wee Sun Lee, Daniela Rus</i> | |
| Point-Based Policy Transformation: Adapting Policy to Changing POMDP Models | 493 |
| <i>Hanna Kurniawati, Nicholas M. Patrikalakis</i> | |
| From Formal Methods to Algorithmic Implementation of Human Inspired Control on Bipedal Robots | 511 |
| <i>Shishir Nadubettu Yadukumar, Murali Pasupuleti, Aaron D. Ames</i> | |
| Direct Trajectory Optimization of Rigid Body Dynamical Systems through Contact | 527 |
| <i>Michael Posa, Russ Tedrake</i> | |
| Robust Online Motion Planning with Regions of Finite Time Invariance | 543 |
| <i>Anirudha Majumdar, Russ Tedrake</i> | |
| Towards Consistent Vision-Aided Inertial Navigation | 559 |
| <i>Joel A. Hesch, Dimitrios G. Kottas, Sean L. Bowman, Stergios I. Roumeliotis</i> | |
| Learning to Segment and Track in RGBD | 575 |
| <i>Alex Teichman, Sebastian Thrun</i> | |

| | |
|--|-----|
| The Path Inference Filter: Model-Based Low-Latency Map Matching of Probe Vehicle Data | 591 |
| <i>Timothy Hunter, Pieter Abbeel, Alexandre M. Bayen</i> | |
| Predicting Pedestrian Trajectories Using Velocity-Space Reasoning | 609 |
| <i>Sujeong Kim, Stephen J. Guy, Wenxi Liu, Rynson W.H. Lau, Ming C. Lin, Dinesh Manocha</i> | |
| Erratum | |
| Direct Trajectory Optimization of Rigid Body Dynamical Systems through Contact | E1 |
| <i>Michael Posa, Russ Tedrake</i> | |
| Author Index | 625 |

The Minimum Constraint Removal Problem with Three Robotics Applications

Kris Hauser

Abstract. This paper formulates a new Minimum Constraint Removal (MCR) motion planning problem in which the objective is to remove the fewest geometric constraints necessary to connect a start and goal state with a free path. I present a probabilistic roadmap motion planner for MCR in continuous configuration spaces. The planner operates by constructing increasingly refined roadmaps, and efficiently solves discrete MCR problems on these networks. A number of new theoretical results are given for discrete MCR, including a proof that it is NP-hard by reduction from SET-COVER, and I describe two search algorithms that perform well in practice. The motion planner is proven to produce the optimal MCR with probability approaching 1 as more time is spent, and its convergence rate is improved with various efficient sampling strategies. The planner is demonstrated on three example applications: generating human-interpretable excuses for failure, motion planning under uncertainty, and rearranging movable obstacles.

1 Introduction

Planners typically operate in binary fashion: they output a valid path if successful, but if they fail, then they provide no explanation for the failure. For several applications, it would be useful for planners to provide more informative explanations about their failures.

- In human-robot interaction, semantically meaningful explanations would help people diagnose and rectify problems.
- For a CAD/CAM designer of a compact assembly, a planner that explained why part cannot be accessed during assembly or repair would help the designer make appropriate modifications.

Kris Hauser
School of Informatics and Computing,
Indiana University,
Bloomington, IN
e-mail: hauserk@indiana.edu

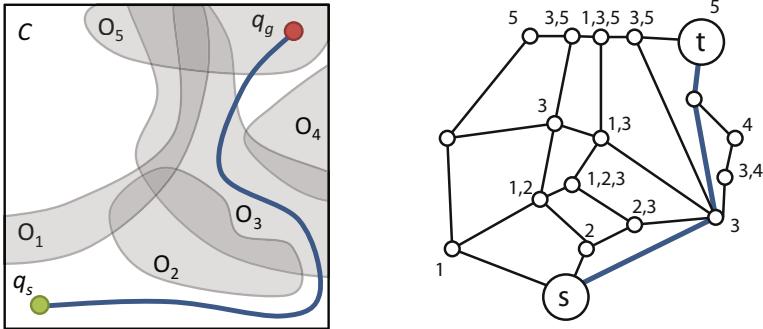


Fig. 1 MCR illustrated in a 2D configuration space. At least two obstacles (O_3 and O_5) need to be removed for a path to connect the start and goal. At right, a partition of the continuous space yields a discrete MCR problem on a graph. Each node is labeled with the subset of obstacles that “cover” the node. The MCR is a minimal subset of obstacles that covers all nodes in a path from the source s to the target t (highlighted). Because the obstacle partition is hard to compute exactly, our planner constructs progressively better approximations by growing a probabilistic roadmap.

- Explanations for planner failures would help a technician debug robot crashes.
- Robots that manipulate obstacles can use explanations to help it decide which obstacles must be moved in order to admit a feasible path.

Recent work has studied excuse-making in the symbolic planning setting by seeking small changes to the initial state that yield a feasible path [5]. A similar notion in the motion planning literature is the problem of finding *disconnection proofs* that certify that no path can be found [1, 2, 16, 11]. Proving infeasibility is computationally challenging and so prior techniques are effectively limited to low-dimensional [16, 11], geometrically simple [1], or algebraically simple [2] settings. Also, these approaches are not constructive in that they do not consider how to change a space in order to yield a feasible path.

I present a new motion planning formulation in which the planner generates explanations for failure by exploring counterfactual scenarios where subsets of constraints are removed from the state space (Figure 1). In particular we are interested in minimizing the number of constraints necessary to admit a solution, because explanations for failure should be parsimonious. This new class of *minimum constraint removal* (MCR) problems differs significantly from prior motion planning problems. It exhibits the curse of dimensionality in continuous spaces along with combinatorial complexity, because subsets of the constraint set (whose size can range into hundreds or thousands) must be enumerated in order to find optimal solutions. As a result, we seek efficient approximate solutions.

I present a sampling-based motion planner that approximately solves the MCR problem in high-dimensional configuration spaces. It incrementally grows a probabilistic roadmap (PRM) in the configuration space that approximates the

connectivity of the obstacle partition, and computes the path in the roadmap that violates the fewest constraints. I prove that it is asymptotically optimal, in that as more time is spent planning, it computes the fewest number of obstacles needed to admit a feasible path with probability approaching 1 under relatively weak conditions. Moreover it is an *any-time* algorithm that can be terminated at any time to yield the best result found so far. A key subroutine is a method to compute the minimum number of constraint violations to each node when restricted to paths on the roadmap. This requires solving the analogous *discrete MCR* problem. I demonstrate several new theoretical results for discrete MCR, including a proof that it is NP-hard by reduction from SET-COVER. I also present two search techniques that work well empirically.

The resulting planner is demonstrated to solve MCR problems in spaces up to 7D with hundreds of constraints. The capability to solve MCR problems appears to be valuable for several applications in robotics, and here we demonstrate its use on the problems of generating human-interpretable explanations of infeasibility, motion planning under environmental uncertainty, and backwards reasoning for manipulation planning.

2 The Minimum Constraint Removal Problem

For robotics applications we are interested in solving the following problem:

Continuous MCR

Input. connected d -dimensional configuration space $\mathcal{C} \subseteq \mathbb{R}^d$, n obstacle regions $O_1, \dots, O_n \subseteq \mathcal{C}$ which are taken to be open sets, and endpoints $q_s, q_g \in \mathcal{C}$.

Definition. The *cover* $C(q) : \mathcal{C} \rightarrow 2^{\{1, \dots, n\}}$ determines the set of obstacles violated at configuration q : $C(q) = \{i \mid q \in O_i\}$. Likewise define $C(A)$ for any subset $A \subseteq \mathcal{C}$ to be the union of $C(q)$ over all points $q \in A$, or equivalently $C(A) = \{i \mid A \cap O_i \neq \emptyset\}$.

Definition. For a subset $S \subseteq \{1, \dots, n\}$, we say that q' is *S-reachable* from q if there exists a continuous path $y(s) : [0, 1] \rightarrow \mathcal{C}$ starting at q and ending at q' that satisfies $C(y(s)) \subseteq S$ for all $s \in [0, 1]$.

Output. A *minimum constraint removal* (MCR) S^* is the cover of a path between q_s and q_g that attains minimum size.

Equivalently, the MCR is the (not necessarily unique) smallest set such that q_g is S^* -reachable from q_s . Often we are also interested in producing the connecting path y that achieves the cover $C(y) = S^*$, which is the *witness* to S^* . We also say that q' is *k-reachable* from q if q' is S -reachable for some subset S of size k .

MCR generalizes the classical Piano Mover's Problem [13], because $S^* = \emptyset$ iff there exists a feasible path. Otherwise, S^* "explains" the infeasibility optimally in the following way. If the constraints indexed by S^* are removed from the original problem, then q_s can be connected to q_g with a feasible path, and there is no way to connect the two without removing fewer than $|S^*|$ constraints.

If a planner computes a partition of \mathcal{C} , the connectivity among those regions forms a discrete graph. If each element in this partition is connected and has uniform cover, then Continuous MCR is successfully reduced to a discrete MCR problem on the graph. Such decompositions may be produced in theory, for example, by computing the partition of \mathcal{C} induced by the obstacle boundaries. Although there are methods from computational geometry to compute partitions for lines, planes, and spheres, such decompositions are intractable to compute in general for complex obstacles in high dimensional spaces. Nevertheless Discrete MCR is an essential subproblem for our sample-based approach. It is specified as follows:

Discrete MCR

Input. graph $G = (V, E)$, cover function $C[v]$, start and terminal vertices $s, t \in V$. $C[v]$ marks each vertex v with a subset of $\{1, \dots, n\}$ denoting which constraints are violated at v .

Output. A subset S^* of $\{1, \dots, n\}$ of minimum size such that there exists a path P in G from s to t for which $C[v] \subseteq S^*$ for all vertices $v \in P$.

The definitions provided above for Continuous MCR extend directly to the analogous concepts for Discrete MCR.

3 Analysis and Algorithms for Discrete MCR

Before proceeding to an algorithm for the continuous case, we will first prove some theoretical results about the MCR problem on graphs. Two practical search algorithms will be presented: one greedy and one optimal. Both of these will be used later in our planner to solve MCR on roadmap discretizations of the continuous space.

3.1 Discrete MCR Is NP-Complete

This section proves the following theorem.

Theorem 1. *The decision version of MCR (i.e., “is there a set S of k constraints such that t is S -reachable from s ?”) is NP complete.*

The proof that MCR is in NP is simple. Given a certificate in the form of an explanation S , S -reachability can be computed in polynomial time by selecting the subgraph of vertices v with $C[v] \subseteq S$ and computing the shortest path from s to t . To prove NP-hardness we perform a polynomial-time reduction from SET-COVER.

The input to SET-COVER is a universe $U = \{1, \dots, m\}$ and a family $F = \{S_1, \dots, S_n\}$ of subsets of U . A cover is a subset of F whose union covers U . SET-COVER asks whether there exists a cover that contains at most k elements of F . An instance of SET-COVER can be reduced to an instance of MCR on a graph constructed as follows (Figure 2). Let $V' = \{(e, S) \text{ for all } e \in U \text{ and } S \in F \mid e \in S\}$ be vertices indicating element-subset pairs for which the subset covers the element. Let $E' = \{(e, S) \rightarrow (e', S') \text{ for all } ((e, S), (e', S')) \in V' \times V' \mid e' = e + 1\}$ connect all

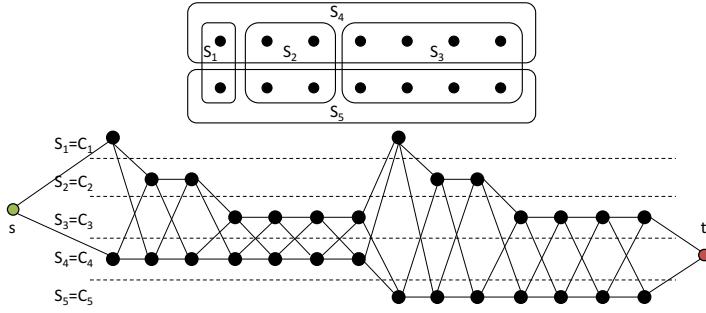


Fig. 2 The reduction from SET-COVER to discrete MCR for a 14-element, 5-set example problem. A graph is constructed so that the optimal set cover $\{S_4, S_5\}$ corresponds precisely to the MCR $\{C_4, C_5\}$.

vertices with numerically subsequent elements. Finally, construct $G = (V, E)$ by augmenting (V', E') with starting node s and terminal node t , where s is connected to all vertices with $e = 1$, and t is connected to all vertices with $e = m$. Clearly the size of this graph is polynomial in m and n . Now construct obstacles $C[(e, S_i)] = \{i\}$ such that each vertex in V' is covered by exactly the index of its subset, and let $C[s] = C[t] = \{\}$. Note that any path in G from s to t passes exactly once through all elements of U , and the union of $C[v]$ for all v along this path is a cover of U . Hence if $MCR(G, C, s, t, k)$ is true, then there exists an explanation S of size k that corresponds to a cover of size k . QED

A consequence of this reduction is that the optimization version of $MCR(G, C, s, t)$ cannot be approximated in polynomial time within a factor of $1/4 \log_2 n$ unless NP is in $DTIME(n^{\text{poly log} n})$ (Lund and Yannakakis, 1994 [10]). The reduction does not apply in the converse, because solutions to SET-COVER cannot be easily converted into solutions for MCR. Although SET-COVER has a greedy polynomial time approximation with $O(\log n)$ error [15], we suspect that MCR is actually harder to approximate. In Section 3.4 we will show that the natural greedy MCR algorithm has worst case $O(n)$ error.

3.2 Optimal Search Algorithm

Best-first search can be employed to solve Discrete MCR exactly. This formulation explores a state space in which states are vertex/subset pairs (v, S_v) in which S_v is the cover of some path P leading from s to v . Note that each vertex can be reached via many paths, each of which could be explained by a distinct subset, and hence a given vertex may potentially appear in 2^n search states. The search proceeds in best-first fashion in order of increasing $|S_v|$ from the initial state $(s, C[s])$, and generates children by enumerating edges $(v, S_v) \rightarrow (w, S_w)$ with $w \in \text{Adj}(v)$ and $S_w = C[w] \cup S_v$.

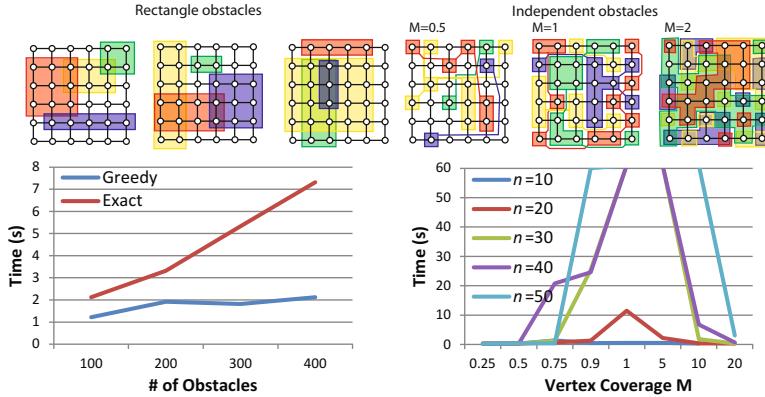


Fig. 3 Left: running times for exact and greedy search with the on a 100×100 grid with n random rectangular obstacles. Right: running times for exact search on a $10 \times 10 \times 10$ grid with n total obstacles and each vertex is covered by M obstacles chosen independently at random. (Greedy search solves each problem in less than 0.3 s).

It is complete and optimal because it maintains the following invariant: when a state (v, S_v) is expanded at node v , S_v is the minimum size cover over all paths from s to v .

Revisited and suboptimal states are pruned according to the following definition:

Definition. An *irreducible constraint removal* (ICR) is a set S such that t is S -reachable from s but not S' -reachable for any strict subset $S' \subset S$.

The search can safely prune non-ICR sets at a given vertex, i.e., a state (v, S_v) can be pruned if we have previously visited a state (v, S'_v) with $S'_v \subseteq S_v$. These pruned states do not affect the optimal explanation because that path to each descendant in the search tree under (v, S_v) is covered by a set no smaller than the cover of the path under (v, S'_v) that visits the same vertices.

In the worst-case the search generates $O(|G|2^{|S^*|})$ states. However, in practice the pruning step eliminates a large number of states and hence the algorithm is practical for many MCR instances.

3.3 Empirical Results on Random Graphs

We evaluated the optimal search on 2D and 3D grids of varying resolution and with varying numbers of random obstacles. It appears that the spatial coherence of obstacles is more relevant to performance than their number. We consider two random obstacle models (Figure 3, top):

1. Rectangles: rectangular obstacles are sampled randomly from the grid.
2. Independent Vertex: coverage sets $C[v]$ are drawn independently at random *per vertex*.

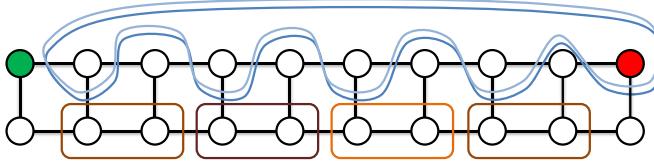


Fig. 4 An example for which the greedy discrete MCR algorithm achieves $O(n)$ error. The case where $n = 6$ is shown. To connect the start (left) and the goal (right), two overlapping obstacles block the top path, while $n - 2$ disjoint obstacles block the bottom. Greedy search produces the suboptimal bottom path even though the top path is optimal. The example generalizes to larger n by repeating the pattern on a wider graph.

For the Rectangles model, running times are roughly linear in the number of obstacles (Figure 3, left). The Independent Vertex model gives very different behavior. Running times are somewhat dependent on n , but curiously they rather rise dramatically for intermediate levels of vertex coverage (Figure 3, right). These obstacles realize the potential for worst-case exponential growth, as instances around $n = 30$ and vertex coverage 3 exhaust our test computer’s 2Gb of RAM. Note that for these tests we implemented MCR in the interpreted Python language, which is orders of magnitude slower than the implementation used in our motion planner. Interestingly, as coverage grows higher, running time becomes more manageable. This type of transition from easy-to-hard-to-easy problem instances has been observed in many NP-complete problems [12].

The per-vertex random obstacle model is a pathological case that is atypical of problems encountered in practice. Yet because discrete MCR searches are used often in planning it may be prudent to avoid rare case of exponential growth at the cost of optimality. This motivates our development and analysis of a greedy MCR algorithm.

3.4 Greedy Search

Greedy search is quite similar to the exact search but it prunes more aggressively. At each vertex it keeps only the subset S_v with minimum size and prunes any states (v, S'_v) with $|S'_v| \geq |S_v|$. Because it is guaranteed to only expand each node once, greedy search runs in $O(|G|n)$ time. A drawback is that the approximation factor can be arbitrarily bad. A counterexample, shown in Figure 4, demonstrates the possibility of achieving $O(n)$ error.

In experiments, greedy search is very fast and can solve much larger problems than the exact search. Problems with $|S^*|$, n , and $|V|$ on the order of thousands, and even the pathological cases of Figure 3 are solved in fractions of a second. Perhaps surprisingly, it also produces high-quality paths in practice. In our tests on Rectangles problems it seems to have 0 approximation error, and in Independent

Vertex problems its error was at most 2. This suggests that greedy search is not likely to introduce severe approximation errors in practice, particularly on the spatially coherent obstacles typical of continuous MCR problems.

The next theorem helps explain why the greedy algorithm performs well in practice. First consider the following lemma.

Lemma 1. *Let $P = (v_0, \dots, v_t)$ be any path in G starting at $v_0 = s$. Then the size of the subset at v_t computed by the greedy algorithm is no more than*

$$|C[v_0]| + \sum_{i=1}^t |C[v_i] \setminus C[v_{i-1}]|. \quad (1)$$

Proof. Let S_0, \dots, S_t denote the subsets obtained at v_0, \dots, v_t by greedy search. Define the partial sums $k_0 = |C[v_0]|$, $k_1 = k_0 + |C[v_1] \setminus C[v_0]|$, \dots , $k_t = k_{t-1} + |C[v_t] \setminus C[v_{t-1}]|$.

We will prove that $|S_i| \leq k_i$ by induction on i . The base case with $i = 0$ is true by definition. Now consider the subset S_{i-1} reached at v_{i-1} and make the inductive assumption $|S_{i-1}| \leq k_{i-1}$. Because S_{i-1} also contains $C[v_{i-1}]$, the search would add precisely $|C[v_i] \setminus C[v_{i-1}]|$ elements to S_{i-1} if it took the candidate edge from (v_{i-1}, S_{i-1}) to (v_i, S_i) . The greedy search has the option of traversing the edge $v_{i-1} \rightarrow v_i$, and will not choose an edge into v_i that produces a larger subset. So, we have that $|S_i| \leq |S_{i-1}| + |C[v_i] \setminus C[v_{i-1}]| \leq k_{i-1} + |C[v_i] \setminus C[v_{i-1}]| = k_i$ as desired. By induction this inequality holds for all i . QED

Theorem 2. *The solution computed by the greedy algorithm is optimal if there exists an s - t path P with cover S^* such that each obstacle in S^* enters into P at most once. Here, “entering” means that for any $i \in S^*$, the set of vertices along P for which i lies in their cover form a connected subsequence.*

Proof. Number the vertices of such a path $P = (v_0, \dots, v_t)$. The size of the covers of each prefix of P form a nondecreasing sequence (k_0, \dots, k_t) for which $k_0 = |C[s]|$ and $k_t = |S^*|$. The single entry assumption shows that $k_{i+1} - k_i = |C[v_{i+1}] \setminus C[v_i]|$. Now consider the S_t obtained at v_t by greedy search. By the lemma, $|S_t| \leq |C[v_0]| + \sum_{i=1}^t |C[v_i] \setminus C[v_{i-1}]| = k_t = |S^*|$. Since $|S^*|$ is optimal, $|S_t| \geq |S^*|$, and hence $|S_t| = |S^*|$ as desired. QED.

A similar line of reasoning leads to the conclusion that the approximation error of the greedy algorithm is bounded by the minimal number of times that obstacles must be reentered. Given the difficulty involved in constructing problem instances in which this number is high, it would stand to reason that the greedy algorithm almost always has low error in nonadversarial settings.

4 Continuous MCR Motion Planner

Let us now return to the continuous setting. Our motion planner grows a probabilistic roadmap (PRM) that progressively approximates the true connectivity of the

obstacle partition. As the approximation improves, a discrete MCR query restricted to the roadmap will approach the true MCR. The resulting algorithm is any-time, in that it can be queried at any time to produce an increasingly accurate series of MCR estimates. We also introduce exploration strategies that help improve the planner’s convergence rate.

4.1 Summary

The algorithm maintains a probabilistic roadmap G , which is a network of configurations (known as *milestones*) in the configuration space that are connected by straight-line paths. The planner is specialized to a problem instance by providing by two problem-specific subroutines: $\text{Cover}(q)$, which computes the set of constraints violated at configuration q ; and $\text{EdgeCover}(q, q')$, which computes the set of constraints violated along the straight-line path between q and q' .

We are primarily concerned with how well reachability on the roadmap approximates the true reachability in the continuous space, so we define some new terminology. Given G and an exploration limit k , we say that a node q in G is (G, k) -reachable if there exists a path in G from q_s to q with a cover of size no more than k . A good exploration strategy constructs G so that k -reachable nodes are likely to be (G, k) -reachable.

(G, k) -reachability is calculated using Discrete MCR search with the cover function taken to be $C[q] \equiv \text{Cover}(q)$. This works directly when each edge $q \rightarrow q'$ satisfies the condition $\text{EdgeCover}(q, q') = \text{Cover}(q) \cup \text{Cover}(q')$. For edges that violate this condition, we treat the edge as a “virtual node” to propagate edge-wise constraint violations appropriately.

Our planner grows G by expanding from (G, k) -reachable nodes for some *exploration limit* k . In a manner reminiscent of the RRG planner [8], an RRT-like [9] strategy encourages fast exploration of the (G, k) reachable space, while a PRM-like strategy connects new nodes to their nearby neighbors. These connections improve the connectivity of the roadmap and the quality of the (G, k) approximation. The choice of the limit k is another important facet of the exploration strategy. We vary k during planning using a strategy governed by two principles. First, the cover of any given path to the goal is an upper bound on the true $|S^*|$. So, k should never be raised above $|S_{\min}| - 1$, where S_{\min} is the cover of the best path found so far. Second, it is more important to begin exploring at low k because undersampling regions that are reachable with low k will handicap the planners ability to identify regions that are reachable with high k . So, we use an *incremental raising* strategy.

4.2 Algorithm

The planner takes as input a problem description and two parameters: N_{raise} , which dictates how many expansions are performed before raising k ; and δ , an RRT-like

step-size that governs the maximum length of edges in the roadmap. The algorithm is as follows:

Continuous-MCR:

1. $S_{min} \leftarrow \text{EdgeCover}(q_s, q_g)$
2. $k \leftarrow |\text{Cover}(q_s) \cup \text{Cover}(q_g)|$
3. $G \equiv (V, E) \leftarrow (\{q_s, q_g\}, \{q_s \rightarrow q_g\})$
4. For $N = 1, 2, \dots$ repeat:
 5. Expand-Roadmap(G, k)
 6. Compute the minimum explanations $S_G(q)$ for all $q \in V$
 7. $S_{min} \leftarrow \arg \min_{S \in S_G(q_g)} |S|$
 8. Every N_{raise} steps, raise k .
 9. If $k \geq |S_{min}|$, set $k \leftarrow |S_{min}| - 1$.

Lines 1-3 initialize the roadmap G to a single edge from q_s to q_g , and sets the exploration limit k to a known lower bound, which is the union of constraints violated at the start and goal. Line 5 expands the roadmap G using random sampling, and respects the exploration limit. Line 6 computes for each milestone q the minimum explanation sets $S_G(q)$ from q_s to q , restricted to edges of the roadmap G . Here we have the option of using the exact Discrete-MCR search, in which $S_G(q)$ is set of one or more irreducible explanation sets; or using the greedy search, in which $S_G(q)$ will just consist of a single explanation set that may not be minimal. Line 7 updates the best explanation, and lines 8-9 update the exploration limit. The operation of Continuous MCR is illustrated in Figure 5.

The Expand-Roadmap procedure grows the roadmap while limiting the domain of exploration such that each added node is guaranteed to be (G, k) -reachable. It operates very much like RRG in that it expands the roadmap toward a configuration drawn at random from \mathcal{C} , but then it also adds connections to neighbors as well.

Expand-Roadmap(G, k)

1. $q_d \leftarrow \text{Sample}()$
2. Let $q_n \leftarrow \text{Closest}(G, k, q_d)$
3. $q \leftarrow \text{Extend-Toward}(q_n, q_d, \delta, k)$
4. Let $\{q_1, \dots, q_m\} \leftarrow \text{Neighbors}(G, q)$
5. For $i = 1, \dots, m$, do:
 6. If $d(q_i, q) < \delta$, then add $q_i \rightarrow q$ to E

It operates by generating a random sample q_d (Line 1) and extends an edge from the closest (G, k) -reachable milestone toward q_d (Lines 2–3). The resulting leaf node q is then connected to nearby milestones in the roadmap (Lines 4–6). Each new extension and connection is limited to the maximum step δ .

Expand-Roadmap uses several subroutines, which are implemented as follows.

Closest. $\text{Closest}(G, k, q)$ finds the closest (G, k) -reachable node to q according to the distance metric $d(q, q')$.

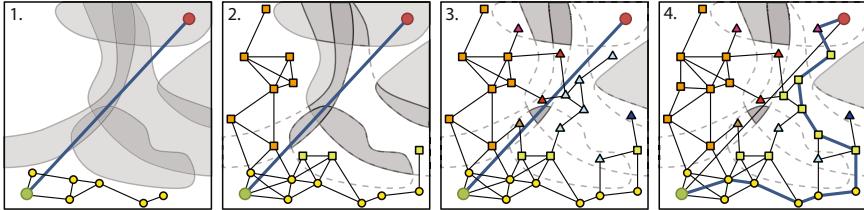


Fig. 5 Illustrating the planner on the example of Figure 1. Circles, squares, and triangles indicate $(G, 0)$ -, $(G, 1)$ -, and $(G, 2)$ -reachable nodes, respectively. 1) The best path is initialized to a straight line, setting $|S_{min}| = 4$. Exploration begins in the feasible subset of the C-space. 2) Exploration after one constraint violation $k = 1$ is allowed. 3) k is eventually raised to 2. A connection between the upper right milestone and the goal is not permitted because such a path would accumulate 3 violations. 4) After further roadmap refinement at $k = 2$, an alternate cover-1 path to the upper right milestone is found, which can then be connected to the goal via a cover-2 path.

Neighbors. $\text{Neighbors}(G, q)$ returns a set of milestones q_1, \dots, q_m that are close to q . As usual in sample based planners, the notion of “close” is defined either by selecting all roadmap nodes within a ball of radius r centered at q , or by picking the m nearest neighbors in the roadmap. Our experiments use the latter approach with $m = 10$.

Extend-Toward. $\text{Extend-Toward}(q_i, q, \delta, k)$ extends the roadmap with a new edge from q_i to a configuration q' in the direction of q . Like RRT, the step is limited to some maximum value δ by taking $q' = q_i + \min(\frac{\delta}{d(q_i, q)}, 1)(q - q_i)$. We also ensure that for some explanation set $S \in S_G(q_i)$, the resulting path to q' has no more than k violations; that is, $|S \cup \text{EdgeCover}(q_i, q')| \leq k$. If this is not satisfied, then we set q' to the midpoint of $q \rightarrow q'$ and repeat the test. This continues until an acceptable point is found or some fixed number of bisections is reached (4 in our implementation).

Reducing Discrete MCR Overhead. We reduce the number of nodes in the Discrete-MCR graph by clustering connected milestones in each region in the obstacle partition using a union-find data structure. The results of Discrete-MCR on the clustered roadmap are equivalent to the full roadmap but the graph is usually much smaller. Also, we observe that Extend-Toward operations simply add new leaf nodes to G and induce no changes to S_G at existing milestones, so, $S_G(q')$ can be computed quickly. Moreover, many new edges constructed in Line 6 do not affect S_G because they do not induce a better path to the neighbors of q' . So, we test whether edges out of q' improve S_G at each of its neighbors. If none are improved, then we avoid recalculating S_G . Finally, we reduce overhead even further by recalculating only local modifications to S_G in the manner of dynamic shortest paths algorithms [4]. In our experiments these reduce the overhead of Discrete-MCR updates to be less than 5% of overall running time amongst our example problems.

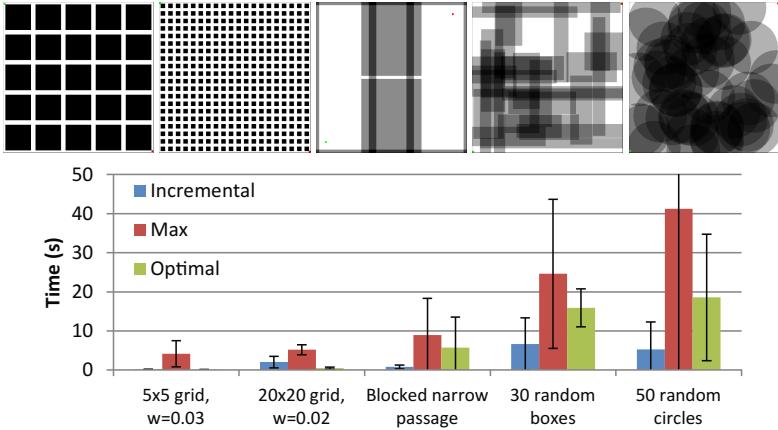


Fig. 6 Comparing the performance of strategies for choosing the exploration limit on five example problems (top row). Average time and standard deviation to find an optimal MCR over 10 runs is plotted for an incremental raising strategy with $N_{raise} = 1000$ (Incremental), a fixed limit equal at $|S_{min}| - 1$ (Max), and a fixed limit equal to the size of the optimal MCR $|S^*|$ (Optimal). MCRs for each problem have size 0, 0, 2, 9, and 12, respectively.

4.3 Rate of Convergence to the Minimum

The MCR planner will produce a true MCR S^* as long as its roadmap contains a path that passes through $\mathcal{C} \setminus \bigcup_{i \in S^*} O_i$. This section applies a theoretical result about traditional PRMs to demonstrate that the MCR planner does indeed converge, and it compares strategies for improving the convergence rate.

Traditional sample-based planners operate in the free space \mathcal{F} — the complement of the obstacle region. Given certain *visibility* characteristics of \mathcal{F} , the probability of failing to connect two points that lie in the same connected component of free space is bounded by an exponentially decreasing curve [6, 9]. Let us denote the domain $\mathcal{C} \setminus \bigcup_{i \notin S^*} O_i$ as \mathcal{F}^* . Observe that once $k \geq |S^*|$, our planner constructs a graph with at least as many milestones and edges as an RRT restricted to \mathcal{F}^* . Hence the probabilistic completeness of RRT [9] extends to our planner as follows:

Theorem 3. *If there exists a path in $\mathcal{F}^* = \mathcal{C} \setminus \bigcup_{i \notin S^*} O_i$ between q_s and q_g with clearance at least $\delta > 0$, then MCR is probabilistically complete.*

But observe that our planner generates many milestones that do not lie in \mathcal{F}^* , and do not contribute to the ultimate solution. So, a good expansion strategy should limit the sampling density to regions that are likely to be candidates for an optimal \mathcal{F}^* . Of course, the planner does not know the shape of \mathcal{F}^* because $|S^*|$ is unknown, and furthermore cannot even test whether a point lies within it. This motivates our choice for raising the expansion limit k incrementally. If k is set too low, then parts of \mathcal{F}^* will remain completely unexplored, but if it is too high, then \mathcal{F}^* will be a small fraction of the explored space.

We tested the effects of the exploration limit on several benchmark problems. Figure 6 shows results. Setting k to the size of the optimal MCR $|S^*|$ (Optimal) is certainly better than simply keeping k one below the size of the current best cover (Max). But it is surprising that incrementally raising k performs many times better than the Optimal strategy on the infeasible problems 3–5. This suggests that while certainly the volume of \mathcal{F}^* is an important factor, at least two other factors are at play:

- Better roadmaps in regions reachable under low k reduce the errors of (G, k) -reachability estimates, which subsequently leads to better roadmaps in regions reachable with high k .
- The overhead involved in calculating S_G is proportional to k .

The parameter N_{raise} must be tuned to achieve a good schedule of exploration limit raises. If there is prior knowledge that S^* is small, then N_{raise} should be large to prevent overexploration. For example, on the feasible benchmark problems #1 and #2, the Incremental strategy performs better as N_{raise} grows larger. If on the other hand S^* is large, then N_{raise} should be smaller to explore more quickly. However, it should not be too small because the benefits of exploring low k regions will be missed. For all of our examples $N_{raise} = 1000$ seemed to be an acceptable compromise.

MCR appears to have a roughly constant factor overhead above traditional sample-based motion planners in feasible problems. In the feasible problems #1 and #2, the MCR planner with the Optimal strategy was approximately 3x and 5x slower than a standard RRT, while the incremental strategy was approximately 4x and 15x slower. Similar overhead factors were observed across several problem variations. We did observe that in a small fraction of runs the planner spends a long time updating S_G when exact discrete MCR search is used. For this reason we typically use greedy search to reduce extreme variations in running time. Over hundreds of planner runs over dozens of benchmark problem variants we have only observed one instance in which the greedy search converged to a suboptimal MCR, and it only overestimated the MCR by 1.

5 Applications

Parsimonious Excuse-Making. If a robot were to succeed at a task *but for* some subset of its geometric, dynamic, and operational constraints, then the existence of those constraints can be interpreted as an explanation for failure. Clearly, small, parsimonious explanations of failure are more easily interpretable by a human. MCR, as we have formulated it, solves this problem for kinematic planning problems under point-wise constraints. Collision avoidance, joint limits, static balance, and static actuator limits fall in this category. Extending MCR to handle kinodynamic planning, differential constraints, and resource constraints would be interesting topics for future work.

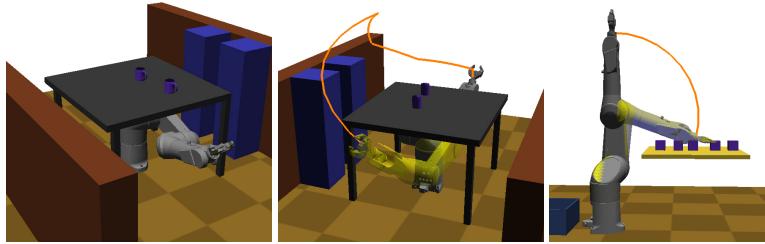


Fig. 7 Left: a robot arm in a highly cluttered environment is asked to reach a goal configuration (Middle, foreground). The MCR states that the goal could be reached but for collision with the table. Right: for simple failures at the query endpoints, MCR terminates with the exact solution almost immediately. End effector trajectories for witness paths are drawn in orange.

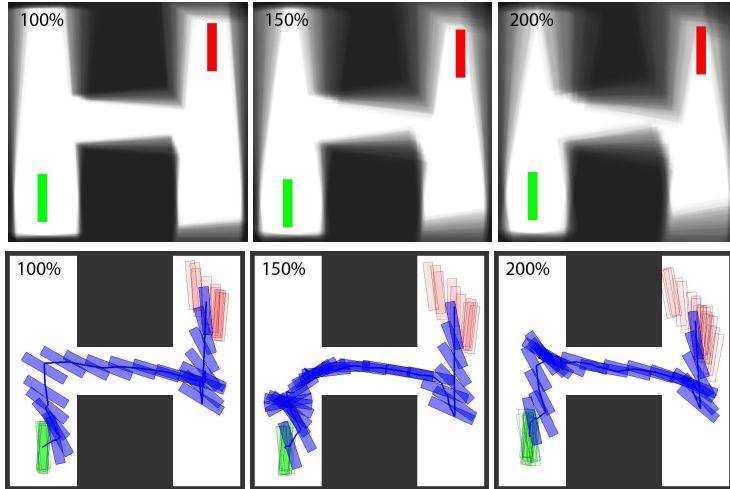


Fig. 8 A bar must move through the corridor from the lower left to upper right of the environment, without sensing feedback. 100 hypothetical samples indicate uncertainty in the world's position and orientation relative to the robot. At the original level of uncertainty (100%), there exists a feasible open-loop motion that brings the bar from the start to the goal. With increased uncertainty (150%) the planner finds a path that collides in 4/100 samples. At 200% uncertainty the optimized path collides in 36/100 samples.

Figure 7 illustrates an application to excuse-making for a 6DOF robot arm with a 1DOF gripper in a cluttered environment. 99 collision, 55 self-collision, and 7 joint limit constraints are modeled as configuration space obstacles. An MCR of size 1 was computed in 11 s which states that at a minimum the arm must collide with the tabletop for a feasible path to exist. Such an explanation may be reported as “the table is in the way”. One common cause of failure is when the start or

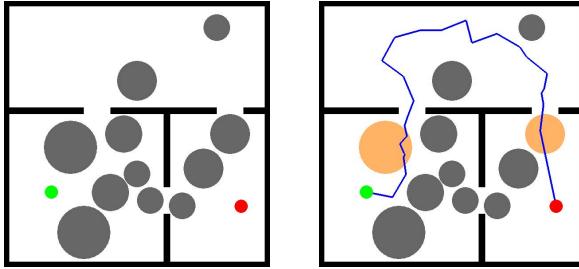


Fig. 9 A circular robot must move some obstacles (circles) out of the way in order to move from the lower right room to the lower left room. A naive direct plan would need to move three obstacles. The MCR planner identifies a path through the third room that requires moving only two obstacles. This path was computed in 2.4 s.

goal configuration is infeasible. MCR performs very well here; it terminates almost immediately because the upper and lower bounds of the exploration limit are found to be equal.

Planning under Environmental Uncertainty. Environmental uncertainty is commonly represented with a set of sampled *particles* that represent hypothetical placements of objects and obstacles in the world. Localization uncertainty can be represented in a similar manner: by fixing the reference frame relative to the robot, the uncertainty is captured in the relative transformation of the world with respect to the robot. Each of these particles can be considered as an instantiation of a C-space obstacle. Applied to this setting, MCR finds the path that *minimizes probability of collision*. Huang and Gupta (2009) considered a related problem of chance-constrained optimal planning, and they present an approximate planner for computing the minimum length path on a given roadmap that exceeds a collision probability threshold [7].

We implemented the minimum probability of collision approach for a simple planar robot that can translate and rotate, and is subject to localization uncertainty (Figure 8). The uncertain start location is represented by 100 particles sampled from a uniform distribution over a box in the (x, y, θ) space. The goal of the robot is to move to the upper right corner of the environment along an open loop path while minimizing the probability of collision. We then ran MCR for 10,000 iterations to generate the optimized paths in Figure 8. On the original feasible problem (100%) MCR converged in 7 s. With higher uncertainty (150% and 200%), MCR spent 68 s and 104 s respectively before progress stalled (as judged when 2,000 iterations passed without finding a better path).

Backward Reasoning for Manipulation Planning. Finally, we consider an application to navigation amongst movable obstacles and manipulation planning. Several authors, including Stilman and Kuffner (2005) and Dogar and Srinivasa (2010) consider *backwards reasoning* techniques for planning sequences of manipulation actions in the presence of cluttered movable obstacles [3, 14]. The common

strategy is to compute a path to the goal in the absence of movable obstacles and then use the robot’s swept volume along this path to determine a set of objects to move. This strategy, however, is only a heuristic and may result in unnecessarily large sets. MCR may lead to more efficient plans that disturb fewer objects (Figure 9).

6 Future Work

Ongoing work is investigating a variety of extensions to the basic MCR problem, including non-unit obstacle costs, goal regions instead of goal configurations, and optimizing both path costs and constraint removal costs. These seem to require only modest implementation change; for example, the latter extension might use techniques from the recent PRM*/ RRG* motion planners [8]. On a more theoretical note, upper bounds on an MCR are easy to obtain but lower bounds are not. It may be possible to use a disconnection prover [1] to provide such bounds and provide a “warm start” to the solver. Finally, for manipulation tasks it may be useful to study a *minimum constraint displacement* problem in which the planner moves constraints as little as possible in order to yield a feasible path.

References

1. Basch, J., Guibas, L., Hsu, D., Nguyen, A.T.: Disconnection proofs for motion planning. In: IEEE Int. Conf. Rob. Aut., Seoul, Korea, pp. 1765–1772 (2001)
2. Bretl, T., Lall, S., Latombe, J.-C., Rock, S.: Multi-step motion planning for free-climbing robots. In: Workshop on the Algorithmic Foundations of Robotics, Zeist, Netherlands (2004)
3. Dogar, M.R., Srinivasa, S.S.: A framework for push-grasping in clutter. In: Robotics: Science and Systems (2011)
4. Frigioni, D., Marchetti-Spaccamela, A., Nanni, U.: Fully dynamic algorithms for maintaining shortest paths trees. Journal of Algorithms 34(2), 251–281 (2000)
5. Göbelbecker, M., Keller, T., Eyerich, P., Brenner, M., Nebel, B.: Coming up with good excuses: What to do when no plan can be found. In: Int. Conf. on Automated Planning and Scheduling (2010)
6. Hsu, D., Latombe, J.-C., Motwani, R.: Path planning in expansive configuration spaces. In: IEEE Int. Conf. Rob. Aut., pp. 2219–2226 (1997)
7. Huang, Y., Gupta, K.: Collision-probability constrained prm for a manipulator with base pose uncertainty. In: IEEE/RSJ Int. Conf. Intel. Rob. Sys., pp. 1426–1432 (2009)
8. Karaman, S., Frazzoli, E.: Incremental sampling-based algorithms for optimal motion planning. In: Robotics: Science and Systems (RSS), Zaragoza, Spain (2010)
9. LaValle, S.M., Kuffner Jr., J.J.: Randomized kinodynamic planning. Int. J. Rob. Res. 20(5), 379–400 (2001)
10. Lund, C., Yannakakis, M.: On the hardness of approximating minimization problems. J. of the ACM 41(5), 960–981 (1994)
11. McCarthy, Z., Bretl, T., Hutchinson, S.: Proving path non-existence using sampling and alpha shapes. In: IEEE Int. Conf. Rob. Aut. (2012)
12. Prosser, P.: An empirical study of phase transitions in binary constraint satisfaction problems. Artificial Intelligence 81(1-2), 81–109 (1996)

13. Reif, J.H.: Complexity of the mover's problem and generalizations. In: 20th Annual IEEE Symposium on Foundations of Computer Science, San Juan, Puerto Rico, pp. 421–427 (1979)
14. Stilman, M., Kuffner, J.: Navigation among movable obstacles: Real-time reasoning in complex environments. International Journal of Humanoid Robotics 2(4) (December 2005)
15. Vazirani, V.V.: Approximation Algorithms. Springer (2001)
16. Zhang, L., Kim, Y., Manocha, D.: A simple path non-existence algorithm using c-obstacle query. In: Akella, S., Amato, N., Huang, W., Mishra, B. (eds.) Algorithmic Foundation of Robotics VII. STAR, vol. 47, pp. 269–284. Springer, Heidelberg (2008)

Hierarchical Decision Theoretic Planning for Navigation Among Movable Obstacles

Martin Levihn, Jonathan Scholz, and Mike Stilman

Abstract. In this paper we present the first decision theoretic planner for the problem of Navigation Among Movable Obstacles (NAMO). While efficient planners for NAMO exist, they are challenging to implement in practice due to the inherent uncertainty in both perception and control of real robots. Generalizing existing NAMO planners to nondeterministic domains is particularly difficult due to the sensitivity of MDP methods to task dimensionality. Our work addresses this challenge by combining ideas from Hierarchical Reinforcement Learning with Monte Carlo Tree Search, and results in an algorithm that can be used for fast online planning in uncertain environments. We evaluate our algorithm in simulation, and provide a theoretical argument for our results which suggest linear time complexity in the number of obstacles for typical environments.

1 Introduction

There is great interest in robots that can safely navigate in common environments such as homes and offices. However, the presence of obstacles poses a serious challenge. Interacting with a single piece of clutter found in typical environments is difficult in itself, and the robot may need to manipulate many pieces of clutter to clear a goal safely. Even given a map of the room, how does the robot decide which path to take, or which object to move? This problem is referred to as Navigation Among Movable Obstacles (NAMO) [17, 18, 21]. NAMO is an important research area that is on the critical path to robot interaction in human environments.

There are two primary challenges in developing a practical algorithm for the NAMO domain: the exponential size of the search space and the inevitable inaccuracies in perception as well as actuation that occur on physical systems.

Martin Levihn · Jonathan Scholz · Mike Stilman
Center for Robotics and Intelligent Machines,
Georgia Institute of Technology,
Atlanta, GA
e-mail: {levihn, jkscholz, mstilman}@gatech.edu

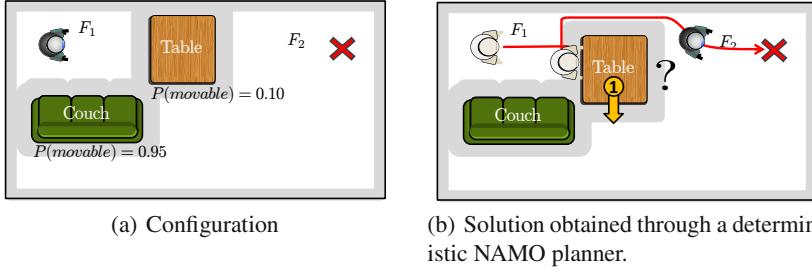


Fig. 1 The table wheels are likely to be locked, making it impossible for the robot to move the table. In contrast to deterministic planners, our proposed framework accounts for this probability.

To understand the NAMO search space, consider navigating in a room with movable obstacles such as the ones depicted in Figure 1 and 2(a). If $\mathcal{C}_R = (x, y)$ represents the configuration of the robot base with resolution n in each direction of motion, then the number of possible robot configurations is $|\mathcal{C}_R| = O(n^2)$. This is true for each object as well, $|\mathcal{O}_i| = O(n^2)$. The full space of possible environment configurations is the product of these subspaces, $\mathcal{C}_R \times \mathcal{O}_1 \times \mathcal{O}_2 \times \dots \times \mathcal{O}_N$, and therefore has $O(n^{2(N+1)})$ world states. In other words, the NAMO state-space is exponential in the number of objects it contains, with a base quadratic in map resolution.

Prior work on NAMO focused on handling the problem of *dimensionality*. It has not yet addressed the underlying issue of *uncertainty*. In reality, robots have noisy actuators and sensors, incomplete action models, and limited perceptual abilities.

To better understand why this might be a problem in an actual NAMO task, consider the example in Figure 1. Perhaps the robot knows that the shortest path to the goal involves moving the table, but it cannot see whether all the table wheels are unlocked. How might it weigh the costs of moving the table versus the couch? How would this answer be affected if it were given only a crude action model for manipulating the couch? These sorts of reasoning patterns are not expressible within the framework of deterministic search, without resorting to *ad hoc* heuristics.

Leveraging ideas from decision theory, we have achieved a novel representation that formally addresses uncertainty in NAMO - it is useful, efficient and theoretically well-defined. By casting the NAMO problem as a hierarchical Markov Decision Process (MDP), we describe the first NAMO planner which can bias its decisions at *plan time* in order to compute policies that are *likely to succeed*.

This work is organized as follows: Section 2 gives an overview of the challenges of stochastic planning in NAMO, and our approach to addressing them. Following related work in Section 3, Section 4 reviews the relevant ideas in stochastic planning, which we combine to describe our algorithm in Section 5. Section 6 provides a theoretical and empirical evaluation of our results, and Section 7 concludes with final remarks.

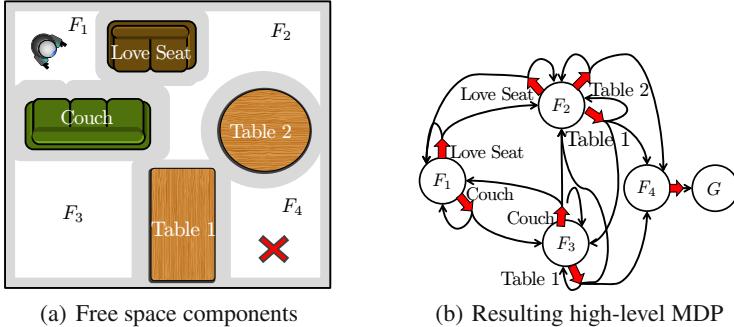


Fig. 2 Concept visualization. The proposed framework defines a hierarchy of MDPs. The high level MDP operates with abstract states representing free spaces and abstract actions reflecting the notion of “creating an opening” by manipulating a specific obstacle.

2 Overview

Our general strategy to handling uncertainty in NAMO is to construct an “approximate” MDP which closely resembles the real problem, but can be solved in real-time. To achieve this, we draw on two techniques from Reinforcement Learning literature. First, we hierarchically decompose the overall task to *create shorter subtasks*, and second we apply an online Monte-Carlo technique to *solve only the relevant portion of the subtask*.

The NAMO problem has two properties that make it a good candidate for hierarchical planning. First, there is a natural abstraction from the low-level state space into a small set of *abstract states*, which we call “free-space regions” to indicate that the robot can move freely without collision. Second, there are a small number of implied *abstract actions* for maneuvering in this abstract state space. Since each free space region is circumscribed by obstacles, we can define the abstract action “create an opening to neighboring free-space” for each obstacle. These two properties are the necessary components of an *abstract MDP* (Section 4), which is the basis for our algorithm. Fig. 2(b) shows this representation for the environment visualized in Fig. 2(a).

A hierarchical framework by itself, however, is insufficient for creating a tractable NAMO MDP: even the individual manipulation tasks are too large to solve completely. For this reason we incorporate a complementary approach called Monte-Carlo Tree Search (Section 4.3). MCTS is a technique for generating a policy for one state at a time, with runtime that depends on the length of the plan rather than the number of states. MCTS is well-suited for the individual manipulation tasks for two reasons. First of all, the abstract MDP only requires a manipulation policy from a few starting locations of the obstacle, which are known at plan-time. Second, since the abstract MDP divided the NAMO problem into smaller manipulation tasks,

the overall policy length for any subtask is quite small. These two properties allow us to substitute MCTS in place of value-iteration for our obstacle manipulation planner, without compromising the hierarchical optimality for the overall algorithm.

3 Related Work

The proposed framework lies at the intersection of search-based algorithms for the NAMO domain and stochastic planning algorithms for general domains with uncertainty. This section provides an overview of these two topics and their relationship to the proposed work.

3.1 NAMO Planning

Navigation and manipulation planning poses a significant computational challenge even with *complete environment information*. Wilfong [23] first proved that deterministic NAMO with an un-constrained number of obstacles is NP-hard. Demaine [3] further showed that even the simplified version of this problem, in which only unit square obstacles are considered, is also NP-hard.

In [18], Stilman presented a planner that solved a subclass of NAMO problems termed LP_1 where disconnected components of free space could be connected independently by moving a single obstacle. The planner was able to solve the hard problems presented in [2] and was successfully implemented on the humanoid robot HRP-2 [17]. Our state space decomposition is mainly based on the concept of “free-space regions” introduced in [18]. However, the free-space concept in [18] was simply a rhetorical device, and has never actually been used in a NAMO algorithm. The work presented here takes direct advantage of the free-space representation. Subsequent work presented a probabilistically complete algorithm for NAMO domains [21]. However, all these methods solved NAMO assuming perfect knowledge of the environment and deterministic action outcomes.

Wu [7] and Kakiuchi [9] introduced the first extensions to NAMO in *Unknown Environments*. In [7] a planner was presented that could solve NAMO problems given only partial obstacle information. However, that planner is not capable of handling uncertainty, and instead assumes that everything within sensor range is ground truth. Further, actions are again assumed to be deterministic. [9] presented a system that executes NAMO in unknown environments on the humanoid robot HRP-2. However, the authors took a reactive behavior-based approach, rather than attempting full decision theoretic planning.

3.2 Decision Theoretic Planning

As discussed in the previous section, there are two basic forms of uncertainty that may affect a NAMO planner: uncertainty in the *action outcome*, and uncertainty in the world *state*. In the decision theory planning literature, these types of uncertainty

are typically modeled in different ways. The first is captured by a *probabilistic action model*, and is the basis for the Markov Decision Process (MDP). The second requires augmenting the MDP with a *probabilistic observation model*, into the so-called Partially Observable MDPs (POMDP).

POMDPs have been applied separately to *navigation* planning by Koenig and Pineau [12] [14] and *grasping manipulation* by Hsiao [6]. While both domains are related to NAMO, they focus on the configuration space of a single robot or a single object. In contrast, the NAMO domain requires the robot to reason about the full set of objects in its workspace. Existing robot planners that use POMDPs are generally restricted to low-dimensional configuration spaces. This constraint holds even when applying the most recent approximate POMDP solvers such as point-based value iteration [14] and belief compression [15].

Although explicit belief-space planning with the POMDP model offers some theoretical advantages, it is not strictly required in order to handle perceptual uncertainty. In POMDPs, entropy from both the action and observation models ultimately manifests as stochasticity in the feedback the agent gets from the environment. Thus, it is always possible to collapse any POMDP into the simpler MDP rolling the observation uncertainty into the transition model. We take this approach in this paper for the sake of computational efficiency.

Several techniques have been developed for extending the MDP model to hierarchical tasks. Among the most well known include the options framework [19], hierarchies of abstract machines (HAM)[13], and the MAX-Q framework [4]. All three of these approaches rely on a generalization of the MDP to incorporate non-atomic, or “semi-markov” actions. The resulting model is referred to as a semi-markov decision process, or SMDP [5].

The primary difference between these approaches is whether they involve *simplifying* or *augmenting* the original MDP. The goal of the options framework is to introduce abstract actions without compromising the finer-grained planning of the original MDP. Therefore options-planning does not offer any *representational* abstraction: all planning is done in the state space of the original MDP. In HAM, the system designer specifies a hierarchical collection of state machines for solving sub-tasks. This state machine presents an abstract interface for the high-level task, which can again be solved as an SMDP. The drawback of these approaches which makes them inappropriate for NAMO is that the resulting SMDPs are either too low-level (options), or too high-level (HAM).

The third approach, MAX-Q, strikes a balance between Options and HAM. It is well-suited for NAMO because it does not require pre-defined subtask policies, but still utilizes an abstract planning representation. However, MAX-Q still assumes that each subtask is solvable using standard *dynamic programming* methods [4]. These algorithms are based on the theoretical results for the SMDP, which scales at best linearly in the size of the (non-abstract) state space [8]. This prohibits a direct application of the MAX-Q framework to the NAMO domain, and is the main technical challenge of this paper.

This challenge is addressed using an alternative to dynamic programming for solving MDPs, referred to as “sparse sampling” or “monte-carlo tree search”

(MCTS). The MCTS literature describes a family of algorithms which were introduced to provide MDP solvers which scale *independently of the size of state space* [10]. However, MCTS remains exponential in the depth of its search tree, which for NAMO can often require tens or hundreds of primitive actions. Several heuristics have been developed for MCTS, including UCT [11] and FSSS [22], which are significantly more sample efficient than vanilla MCTS. While these algorithms are good candidates for a subtask planner, they can not solve the overall NAMO problem due to the large depth, branching factor, and sparsity of rewards in the core MDP.

4 Preliminaries

This section highlights the concepts underlying our approach. Following the definition of an MDP, both MAX-Q and MCTS will be explained. While substantially different, MAX-Q and MCTS are techniques developed for handling large state spaces in an MDP. To our knowledge, these techniques have never been combined. Consequently this section will treat them separately while the following sections will demonstrate how they can be efficiently combined to achieve a practical planner.

4.1 Markov Decision Processes

The Markov Decision Process (MDP) is a model for stochastic planning, and the foundation of our algorithm. We define an MDP $M = (S, A, T_{ss'}^a, R_s^a, \gamma)$ for a finite set of states S , a finite set of actions A , a transition model $T_{ss'}^a = P(s'|s, a)$ specifying the probability of reaching state s' by taking action a in state s , a reward model $R_s^a = r(s, a)$ specifying the immediate reward received when taking action a in state s , and the discount factor $0 \leq \gamma < 1$.

The standard technique for solving MDPs is *Value Iteration* [16], which is used to find the optimal policy $\pi^* : f(s) \rightarrow a$ which maps states to actions in order to maximize the expected long-term reward, or value $V(s)$ for all states $s \in S$. Generic value iteration (VI) has a polynomial runtime in the number of states (with fixed error ε).

4.2 MAX-Q Value Function Decomposition

The MAX-Q framework is a technique within the reinforcement learning literature which describes how a *value function* for an MDP may be decomposed according to a *task hierarchy*. To understand this idea, consider a two-level hierarchy composed of a high-level policy π_0 , defined over a set of subtask policies $\pi_i, i \in \{1, n\}$, which are in turn defined over primitive actions. That is, $\pi_0 : f(s) \rightarrow \pi_i$, and $\pi_i : f(s) \rightarrow a$.

The key insight behind MAX-Q is that the value functions for the subtask policies $V_{\pi_i}(s)$ contain all the information needed to represent the value function of the parent policy $V_{\pi_0}(s)$. This is true because, according to the Bellman equation for SMDPs,

the value of executing subtask π_i in state s is simply the sum of the (discounted) reward accumulated by π_i itself, and the future expected reward of whichever state π_i terminates in:

$$Q(s, \pi_i) = R(s, \pi_i) + \sum_{s', \tau} P(s', \tau | s, \pi_i) \gamma^\tau V(s') \quad (1)$$

where γ^τ ensures that the value of s' is appropriately discounted according to the time τ that π_i took to terminate [1].

The first term in this expression, $R(s, \pi_i)$, is precisely the information encoded by $V_{\pi_i}(s)$:

$$R(s, \pi_i) = V_{\pi_i}(s) = \mathbb{E}_{\pi_i} \left[\sum_{t=1}^{\tau} \gamma^t R(s, a) \right] \quad (2)$$

Therefore, planning in the high-level task simply involves using the values of the subtasks as immediate rewards for their execution in the high-level policy.

In addition to an analysis of compositional value functions, the full MAX-Q framework also describes several model-free learning algorithms. In this paper, however, we are instead interested in model-based planning, since we assume the robot has knowledge of the effects of its actions. Our algorithm therefore differs in some important details, but shares the primary data structure (a hierarchical Q-function) and general format of “MAX-QQ” [4].

4.2.1 Types of Optimality

So far, we have described a *bottom-up* representation of value functions, in which values of subtasks are “projected up” to parent tasks. This approach provides space efficiency, as well as the opportunity to divide and conquer: each subtask can be learned separately and combined to learn the parent task. Dietterich [4] refers to the class of policies which can be represented by such a model as *recursively optimal*, meaning all policies are optimal given optimal solutions to their subtasks.

The drawback to this approach, which makes it inappropriate for the NAMO domain, is that subtasks are learned without knowing their context in the overall plan. This is a problem, for example, if moving two obstacles provides the same reward, but only one of them opens a path to the goal. In general, tasks with sparse rewards tend to require either additional *shaping rewards* for the subtasks, or a solution strategy that includes *top-down* information [1].

We take the top-down approach by using the value of the target free-space region as a reward for the manipulation policy that clears the appropriate obstacle. This is equivalent to Dietterich’s solution for the model-free case, in which the completion functions (the right-most term of Eq. 1) are treated as subtask terminal state rewards [4]. Policies that are learned in this fashion are referred to as *hierarchically optimal*, meaning that the overall policy is optimal given the constraints of the imposed hierarchy [4].

4.3 Monte-Carlo Tree Search

MCTS was invented as a way of allowing near-optimal planning for MDPs with large or infinite state spaces. The main idea is to relax the goal of computing a full policy, and instead focus on computing the optimal policy for a single state – the state the agent is in. In their original work on *sparse sampling*, Kearns et al. showed that it was possible to obtain ε -optimal Q-value estimates for the current state from a set of sampled transitions, and that the number of samples C per state was independent of $|S|$ [10].

MCTS works by building a search tree from the current state, selecting actions according to some search policy π_s , and sampling transitions from the transition model $T_{ss'}^a$ for each action. This tree generates a set of sampled rewards, which can be backed up to the root node to obtain a Q estimate according to:

$$Q_{SS'}^d(s, a) = R(s, a) + \gamma \sum_{s'} P(s'|s, a) \max_{a'} Q_{SS'}^{d-1}(s', a') \quad (3)$$

$Q_{SS'}^d(s, a)$ refers to the expected value of taking action a in state s , and following the optimal policy for $d - 1$ subsequent steps (so $Q_{SS'}^1(s, a) = R(s, a)$). Equation 3 defines a simple recursion for using forward simulation to select actions, despite uncertainty in the action model.

4.3.1 The Effective Horizon

There is one additional result from the MCTS literature, regarding search depth, which we exploit in our hierarchical NAMO algorithm. Kearns et al. proved in [10] that MCTS could achieve ε -optimality with an $O((|A||C|)^H)$ running time, where H is the *effective horizon* of the problem. Based on the fact that rewards in the distant future have little effect on the Q-values of the current state, this proof bounds H according to ε and R_{max} (the maximum possible reward):

$$H = \lceil \log_\gamma \left(\frac{\varepsilon(1-\gamma)^2}{4V_{max}} \right) \rceil, V_{max} = R_{max}/(1-\gamma) \quad (4)$$

Equation 4 states that the effective horizon *increases* with the value of the maximum possible reward that can be achieved. As we show in Section 5, this relation can be exploited in a hierarchical setting to have the MCTS manipulation planners spend time in proportion to the value of their target free-space region.

Neither MAX-Q nor MCTS alone are appropriate for the NAMO domain. The applicability of MAX-Q is hindered by requiring an exact solution for each sub-task. MCTS in turn is limited by the branching factor and sparsity of rewards in the NAMO domain. In the next section we present a new algorithm that solves these shortcomings by combining the two approaches.

5 Algorithm

In this section we outline how to construct an MDP for the NAMO problem, and how to solve it by combining the tools from MAX-Q and MCTS described above. Recall that a standard MDP has an action model associated with each state. For the NAMO domain, this implies a *displacement model* for each object, which depends on the action a being executed, and the target obstacle o :

$$\delta x, \delta y \sim P(\delta x, \delta y | a, o) \quad (5)$$

In our case, however, we represent action models for a discrete set C of *object categories*. This allows us to incorporate observation uncertainty. Now instead of knowing the action model for an obstacle with certainty, the robot has a belief distribution $P(c|o)$ over categories of action models. The actual displacement model is consequently the marginal over the categories:

$$\delta x, \delta y \sim \sum_c P(\delta x, \delta y | a, c) P(c|o) \quad (6)$$

$P(c|o)$ can be used to encode any uncertainty the robot might have about the category of object it is manipulating. For example, $P(c_{ut}|o)$ could be the posterior probability of obstacle o being an unlocked table c_{ut} given some sensor data D : $P(c_{ut}|D) \propto P(D|c_{ut})P(c_{ut})$.

Our algorithm (Algorithm 1), therefore takes the following input:

1. The set O of obstacles present in the workspace
2. Distributions $P(c_i|o_j)$ representing the probability of obstacle o_j belonging to category c_i
3. Motion models indicating 2D object displacements,
 $P(\delta x, \delta y | a_{ll}^1, c_1) \dots P(\delta x, \delta y | a_{ll}^k, c_m)$, indexed by action and object category
4. \mathcal{C}_{goal} The robot's goal configuration in the workspace

It outputs:

1. A high-level policy π_0 indicating which obstacle to move for each free space
2. A set of low-level partial policies Π_1 indicating the manipulation actions to execute for each obstacle

5.1 The NAMO MDP

The proposed NAMO MDP has a two-level hierarchy, with navigation between regions as the high-level task, and object manipulation as the low-level task. Here we define both MDPs, and their hierarchical semantics. Recall that an MDP is defined as $M = (S, A, T_{ss'}^a, R_s^a, \gamma)$, which leaves four properties to define at each level of the hierarchy (γ can be viewed as a parameter of the algorithm).

States and Actions: The fundamental state space in the NAMO problem is the set of possible configurations \mathcal{C}_W of the robot and environment. We define the low-level MDP M_{ll} in terms of these states S_{ll} , and a set of primitive actions A_{ll} for manipulating obstacles on the map. For simplicity, we used axis-aligned manipulations as

action primitives in our simulation, but in practice these would typically be replaced with a more appropriate choice for physical systems, such as those introduced in [18].

For the high-level MDP M_{hl} , we define the state space S_{hl} to be the set of free-space regions implied by the starting obstacle configuration. In our implementation we identify these regions by performing seeded wavefront expansions on \mathcal{C}_W to determine disjoint regions. During this operation we also save the list of obstacles bounding each free space region into an “adjacency list” \mathcal{L}_r , since these represent the possible manipulation targets for that state. The action space A_{hl} is the union of all possible manipulation sub-tasks for each region, where sub-task $a_{hl}^{ijk} = \pi_{ijk}$ means “open a path from state i to state j by manipulating obstacle k ”. The set of possible obstacles k for each starting state i are obtained from the adjacency list.

Transitions and Rewards: Following the discussion of MAXQ in Section 4.2.1, the rewards in M_{ll} and M_{hl} are defined to reflect their hierarchical relationship. Values for R_{hl} represent expectations over the reward accumulated by subtasks in A_{hl} (Eq. 2), and individual subtasks a_i receive the value of their final state in M_{hl} as a terminal reward (Section 4.2.1). This should be intuitive, since the high-level policy needs to know the actual outcome of executing each possible subtask, and each subtask needs to know its *context* in the high-level policy in order to know how important different obstacles are. Initially, only the state $s \in S_{hl}$ containing the goal configuration has a reward, set according to the robot’s utility function.

The transition probabilities in M_{hl} directly represent the expected outcome of executing a subtask π_{ijk} in some state s_{hl}^i . For example, the manipulation sub-task π_{ijk} terminates in state j if it successfully opens a path from i to j , and terminates in state i otherwise. Therefore, the transition probability $P(s' = j | s = i, \pi_{ijk})$ is 1 if and only if π_{ijk} terminates in j . In addition, this suggests that transition model T_{hl} is sparse: the probabilities are automatically zero for all states that do not share an obstacle in their adjacency lists. Finally, the transition model for the low-level MDP T_{ll} is sampled directly from the displacement model (Eq. 6), encoding the domain uncertainty. This completes the construction of the NAMO MDP.

Note that this construction is an instance of what Dietterich refers to as a funnel abstraction [4]: the value all possible robot configurations (positions) within the target free space get mapped to a single value: the value of that region. This is the basic abstraction from which the hierarchical MDP obtains its savings.

5.2 Solution

Section 5.1 described a two-level hierarchy of MDPs defined in terms of each other: the rewards for M_{ll} were obtained from the values of states in M_{hl} , and the values of states in M_{hl} were defined based on outcomes of subtasks in M_{ll} . With this formulation, values in both M_{ll} and M_{hl} reflect the true transition probabilities and rewards for manipulation actions.

Input: O : Obstacles, $P(c_1|o_1) \dots P(c_m|o_n)$: Distributions over categories,
 $P(\delta x, \delta y | a_{hl}^1, c_1) \dots P(\delta x, \delta y | a_{hl}^p, c_m)$: motion models estimates for action primitives
and categories, \mathcal{C}_{goal} : Goal Configuration

Output: π_h : High Level Policy, Π_l : Set of Low Level Policies

```

1   $(F, \mathcal{L}_r) \leftarrow \text{get\_freespaces}(O)$ ;
2   $M_{hl}(S_{hl}, A_{hl}, T_{hl}, R_{hl}) \leftarrow (F, \{\}, [\mathbf{0}], [\mathbf{0}])$ ;
3   $\pi_0 \leftarrow \emptyset; \Pi_l \leftarrow \emptyset$ ;
   // determine high level MDP definition:
4  foreach  $f_i \in F$  do
5    if  $f_i$  contains  $\mathcal{C}_{goal}$  then
6       $\forall a R[f_i, a] \Leftarrow$  utility of reaching the goal;
7    end
8    foreach  $o_k \in \mathcal{L}_r$  adjacent to  $f_i$  do
9      foreach  $f_j \in F$  do
10     if  $o_k$  adjacent to  $f_j$  then
11        $A_{hl} \Leftarrow A_{hl} \cup \{a_{hl}^{ijk}\}$  ;
12        $T[s_{hl}^i, s_{hl}^j, a_{hl}^{ijk}] \Leftarrow 0.5$ ; // connectivity; actual uncertainty
           encoded at lower level, adds to 1 due to self
           transition for failure case
13     end
14   end
15 end
16 end
   // run value iteration:
17 while error not within  $\varepsilon$  do
18   foreach  $s_{hl}^i \in S_{hl}$  do
19      $v \Leftarrow 0$ ;
20     foreach  $a_{hl}^{ijk} \in A_{hl}$  do
21        $h \Leftarrow \lceil \log_{\gamma}(\frac{(\varepsilon(1-\gamma)^2)/4}{s_{hl}^i \cdot v}) \rceil$ ; // dynamic horizon
22        $(q_k, \pi_{ijk}) \Leftarrow \text{MCTS}(a_{hl}^{ijk}, h, \sum_c P(\delta x, \delta y | a_{hl}^1, c)P(c|o_k), \dots, \sum_c P(\delta x, \delta y | a_{hl}^p, c)P(c|o_k))$ ;
           //  $a_{hl}^{ijk}$  provides necessary information about obstacle
           and free-spaces to connect
23       if  $q_k > v$  then
24          $v \Leftarrow q_k$ ;
25          $\pi_0(s_{hl}^i) \Leftarrow a_{hl}^{ijk}; \Pi_l(o_k) \Leftarrow \pi_{ijk}$ ;
26       end
27     end
28      $s_{hl}^i \cdot v \Leftarrow v$ ;
29   end
30 end
31 return  $\pi_h, \Pi_l$ ;

```

Algorithm 1. Proposed framework

This suggests an iterative scheme in which we alternate between updates to the high-level policy π_0 given the current values for subtasks $V_{\pi_{ijk}}$, and updates to the individual subtasks π_{ijk} given the values in V_{π_0} . However, computing these values requires actually solving the associated MDPs, which was shown to be intractable for M_{ll} in Section 1, since $S_{ll} = \mathcal{C}_W$.

Fortunately, sparse-sampling planners provide a way of obtaining approximate solutions to M_{ll} , and make assumptions which are compatible with our hierarchical approach (Section 4.3). Therefore, our actual algorithm performs the same alternating policy-iteration scheme described above, but with the substitution of an MCTS planner in place of value-iteration for the manipulation MDPs. These MCTS planners compute Q-values in $\mathcal{C}_W \times A_{ll}$, and sample transitions from the displacement model defined in Eq. 6. Sampling terminates when an opening is achieved or the maximum search depth, defined by the horizon H (Eq. 4), is reached. For a general MCTS planner, H is defined as a function of R_{max} . In combination with a hierarchical policy iteration, however, it can be used to explicitly force the computation effort for each subtask planner to scale in proportion to its estimated utility. This is achieved using a *dynamic horizon*. Dynamic horizon recomputes the H-value, Eq. 4, of each MCTS planner *separately* based on the value of the target state in S_{hl} . The overall effect of this operation is that π_i does not waste time sampling actions past the point where rewards can significantly affect the Q-value of the subtask.

The following section will argue the usefulness of this complete framework.

6 Evaluation

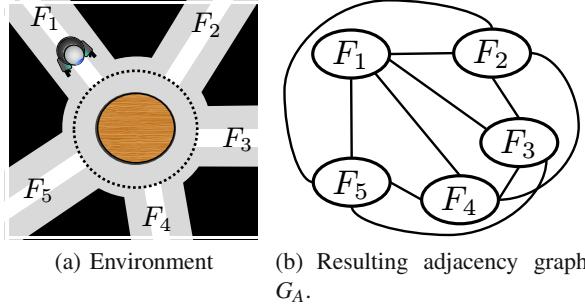
This section provides theoretical and empirical evidence that our algorithm has a run-time which is linear in the number of obstacles, for non-degenerate environments.

6.1 Theoretical Analysis

The following analysis is based on relating the sparsity of the free-space transition matrix T_{hl} to the *adjacency graph* G_A over free space regions. The sparsity of T_{hl} directly controls the complexity of value iteration.

Definition 1. The adjacency graph $G_A = (V, E)$ is defined to have vertices $v_i \in V$ which uniquely represent free space regions F_i , and edges $e(i, j) \in E$ connecting adjacent free space regions. That is, $e(i, j) \in E \Leftrightarrow \text{adjacent}(F_i, F_j)$, with $\text{adjacent}(F_i, F_j) = \text{true}$ iff F_i and F_j are disconnected through a single obstacle.

Figures 3 and 5 show examples of workspaces and their associated adjacency graphs. A graph G is planar if it can be drawn on the plane with no edges crossing except at common vertices. Kurarowski's theorem states that a graph is planar if and only if it does not contain a subgraph that is a subdivision of K_5 or $K_{3,3}$, where K_5 denotes a complete graph with five vertices, and $K_{3,3}$ denotes a complete bipartite graph on six vertices [20]. Note that G_A in typical environments will fulfill this definition. The contrary would require that either:

**Fig. 3** Example of G_A not being planar

- a) A set of five free spaces are all adjacent to each other, separated by only a single obstacle.
- b) A set of three free spaces are all adjacent to a disjoint set of three free spaces, but not adjacent to each other

Figure 3 shows one of these degenerate cases.

Further, recall that by definition of T_{hl} , the only next-states with non-zero transition probability from state s_{hl} are states representing free spaces adjacent to s_{hl} or s_{hl} itself (failure case). This gives rise to the following lemma:

Lemma 1. *T_{hl} has on average a constant number of next-states with non-zero probability if G_A is planar.*

Proof. First, it is trivially true that on average, rows of T_{hl} contain a constant number of self-transitions: $\frac{|S_{hl}|}{|S_{hl}|} = 1$. The remaining non-zero entries in T_{hl} are a direct mapping of $G_A = (V, E)$. The number of next-states with non-zero probability for s_{hl} is equal to the degree d of the associated vertex in G_A . This suggests that the average number of next-states with non-zero probability in T_{hl} is equal to the average degree of G_A .

We now bound the average degree of G_A to show that the number of non-self transitions in T_{hl} is at most 6. First, recall Euler's formula, which places a constraint on the number of possible edges e , vertices v , and faces f in a planar graph:

$$v - e + f = 2 \quad (7)$$

Now consider the set of all possible “edge-face pairs” $p \in P$, (for $v > 2$) where an edge is added to P for each face in which it appears. Observe that since an edge can contribute to at most 2 faces, we have $|P| \leq 2e$. In addition, each face must have at least 3 edges, implying $|P| \geq 3f$. Plugging this into Eq. 7, we have $e \leq 3v - 6$. We can now compute the average degree d_{avg} of G_A :

$$d_{avg} = \frac{d_{total}}{\#vertices} = \frac{\sum_{i=1}^v d_i}{v} = \frac{2e}{v} \leq \frac{2(3v - 6)}{v} = 6 - \frac{12}{v} < 6 \quad (8)$$

Consequently, T_{hl} has on average at most 6 next-states with non-zero probability. \square

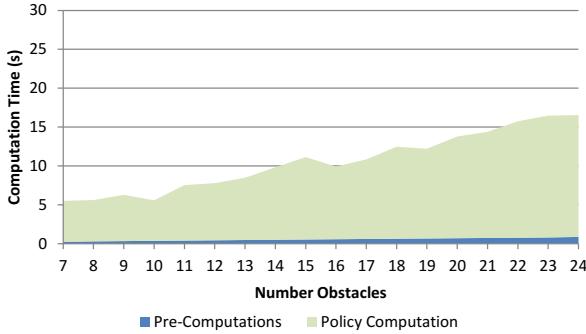


Fig. 4 Obtained average computation times as a function of the number of obstacles. The graph suggest linear complexity in the number of obstacles.

Lemma 2. *The run-time of the proposed framework is linear in the number of obstacles $|O|$ if the adjacency graph G_A is planar.*

Proof. First, consider M_{hl} . The number of states $|S_{hl}|$ is linear in $|O|$ (worst case $2|O|$ for infinite obstacles intersecting at a star pattern). Value iteration is performed over M_{hl} . Since the error ε is a free parameter, the complexity of value iteration is typically separated into the cost-per-iteration and the expected number of iterations. The number of iterations required by VI to achieve an ε -error bound is $N = \lceil \log(2R_{\max}/\varepsilon(1-\gamma))/\log(1-\gamma) \rceil$ [16], which is independent of S_{hl} . The inner loop of value iteration is quadratic in $|S|$ the worst case, due to the need to compute an expectation over the entire state-space in each Bellman update [8]. However, since T_{hl} has on average a constant number of next-states with non-zero probability for planar G_A (Lemma 1), the expected per-iteration cost of value iteration reduces to $6 \times |S_{hl}|$, yielding an overall complexity for M_{hl} of $O(N|S_{hl}|)$.

Finally, each action evaluation in M_{hl} requires MCTS sampling. As discussed in Section 4, the complexity of MCTS is independent of the state space size [10]. It is consequently constant over $|O|$, which implies that the overall algorithm is linear in $|O|$. \square

6.2 Empirical Analysis

To evaluate the proposed framework we have analyzed it using the implementation discussed in Section 5. All the experiments were run on a Intel Core i7 (2.1GHz) PC.

6.2.1 Runtime

We have evaluated the framework on 1000 randomly generated NAMO environments. The size of the map was sampled uniformly to be between 250x250 and 400x400 grid cells. The number of obstacles for each map was uniformly sampled

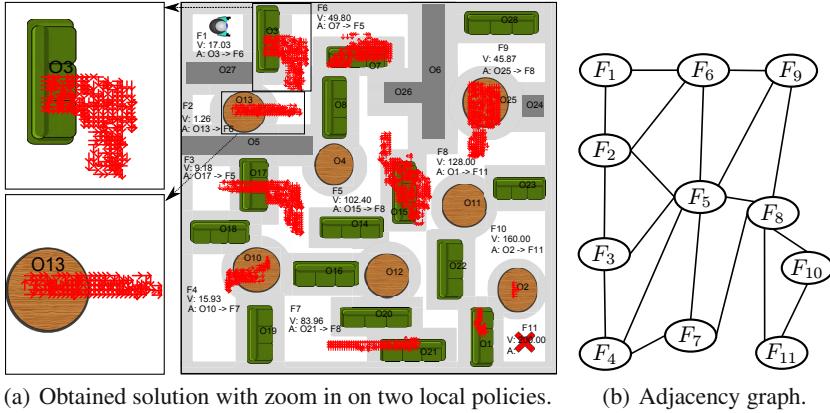


Fig. 5 Example environment. V is the states value and A denotes the action to execute. E.g. the free space containing the robot has a value of 17.03 and the best action to execute is to manipulate obstacle O_3 to gain access to free space F_6 . Low level policies visualized as a vector field. Static obstacles visualized in gray.

to be between 7 and 24, each obstacle in turn having random position, shape (rectangular or ellipsoid) and size (minimum 15x15 cells, maximum 65x65 cells occupied). Motion models $P(\delta x, \delta y | a_{ll}, c)$ were represented using Gaussians with their mean and standard deviation randomly varying for different categories. The category estimates $P(c|o)$ for each object were set to a randomly parameterized categorical distribution. The generated maps had an average of more than 70% cells occupied.

Figure 4 summarizes the computation time as a function of the number of obstacles. Pre-computations include the generation of the configuration space representation and determination of free space regions. While we show computation time as a function of number of obstacles, note that there are many other contributing factors, such as the particular configuration and number of free spaces, the complexity of planning to create openings etc. Figure 4 averages over these factors, and consequently resulted in a high standard deviation of up to 11.1s.

The empirical results are consistent with the theoretical analysis of our algorithm, and support applicability of the approach to practical environments.

6.2.2 Example

Fig. 5 shows an example environment and the solution obtained by our proposed framework. The high level policy is indicated in text, and the low level policy is visualized as a vector field for each obstacle. Only low level policies corresponding to obstacles that are chosen by the high level policy are visualized to preserve a clear view.

6.3 Online Execution

If the robot successfully executes a low-level policy in its workspace, it, in general, has *merged* two free spaces rather than transitioned between them. This alters the state-space of M_{hl} . While future work will investigate the applicability of policy iteration [16] to take advantage of the locality of the change, we address this by re-executing the proposed algorithm after each object manipulation. Given the linear runtime of the algorithm, this has not presented a significant disadvantage. However, as the algorithm does not consider all possible future free space configurations, the Q-values for a specific M_{hl} do not represent the true long term expected reward of those regions, but rather an approximation based on the current world configuration. In general, this approximation is sound as long as actions are reversible. Future work will examine the expected loss and convergence properties under this assumption.

7 Conclusion

We have presented the first decision theoretic planner for the problem of Navigation Among Movable Obstacles. This planner is able to overcome the exponential complexity of the NAMO domain, while incorporating uncertainty, by employing a novel combination of ideas from sparse-sampling and hierarchical reinforcement learning.

Our algorithm can be viewed from the top-down as a value-iteration scheme in the free-space representation of the NAMO problem. From this perspective, the primary difference with classical value iteration is that the Bellman updates issue calls to a low-level MCTS planner for evaluating the action rewards, rather than querying an atomic reward function. In this fashion, values spread through the free-space MDP as they typically would in value iteration. From a bottom-up perspective, we can also view the role of the free-space MDP as a data structure for learning a shaping reward for manipulation subtasks. Thus, the high-level MDP over free space serves to constrain the set of possible actions (and hence the branching factor), as well as the depth of manipulation plans. We expect that this new perspective on the NAMO problem will generate additional insight into robot planning in human environments.

Acknowledgments. This research was supported by NSF grant IIS-1017076. The authors thank Kaushik Subramanian and Jacob Steinhart for many insightful discussions.

References

- [1] Barto, A.G., Mahadevan, S.: Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems* 13(4), 341–379 (2003)
- [2] Chen, P., Hwang, Y.: Practical path planning among movable obstacles. In: Proceedings of the IEEE International Conference on Robotics and Automation, pp. 444–449 (1991)
- [3] Demaine, E., O'Rourke, J., Demaine, M.L.: Pushpush and push-1 are np-hard in 2d. In: Proceedings of the 12th Canadian Conference on Computational Geometry, pp. 211–219 (2000)

- [4] Dietterich, T.G.: An Overview of MAXQ Hierarchical Reinforcement Learning. In: Choueiry, B.Y., Walsh, T. (eds.) SARA 2000. LNCS (LNAI), vol. 1864, pp. 26–44. Springer, Heidelberg (2000)
- [5] Howard, R.A.: Dynamic probabilistic systems, vol. 317. John Wiley & Sons, New York (1971)
- [6] Hsiao, K., Kaelbling, L.P., Lozano-pérez, T.: Grasping pomdps. In: Proc. IEEE Int. Conf. on Robotics and Automation (ICRA), pp. 4685–4692 (2007)
- [7] Wu, H., Levihn, M., Stilman, M.: Navigation among movable obstacles in unknown environments. In: IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, IROS 2010 (October 2010)
- [8] Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement learning: A survey. Arxiv preprint cs/9605103 (1996)
- [9] Kakiuchi, Y., Ueda, R., Kobayashi, K., Okada, K., Inaba, M.: Working with movable obstacles using on-line environment perception reconstruction using active sensing and color range sensor. In: IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), pp. 1696–1701 (2010)
- [10] Kearns, M., Mansour, Y., Ng, A.Y.: A sparse sampling algorithm for near-optimal planning in large markov decision processes. Machine Learning 49, 193–208 (2002)
- [11] Kocsis, L., Szepesvári, C.: Bandit Based Monte-Carlo Planning. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) ECML 2006. LNCS (LNAI), vol. 4212, pp. 282–293. Springer, Heidelberg (2006)
- [12] Koenig, S., Simmons, R.G.: Xavier: A robot navigation architecture based on partially observable markov decision process models. In: Artificial Intelligence Based Mobile Robotics: Case Studies of Successful Robot Systems, pp. 91–122. MIT Press (1998)
- [13] Parr, R., Russell, S.: Reinforcement learning with hierarchies of machines. Advances in Neural Information Processing Systems, 1043–1049 (1998)
- [14] Pineau, J., Gordon, G., Thrun, S.: Point-based value iteration: An anytime algorithm for pomdps (2003)
- [15] Roy, N., Gordon, G., Thrun, S.: Finding approximate pomdp solutions through belief compression. Technical report (2003)
- [16] Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach, 3rd edn. Prentice Hall Press, Upper Saddle River (2009)
- [17] Stilman, M., Nishiwaki, K., Kagami, S., Kuffner, J.: Planning and executing navigation among movable obstacles. In: IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS 2006), pp. 820–826 (October 2006)
- [18] Stilman, M., Kuffner, J.J.: Navigation among movable obstacles: Real-time reasoning in complex environments. Journal of Humanoid Robotics, 322–341 (2004)
- [19] Sutton, R.S., Precup, D., Singh, S.: Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. Artificial Intelligence 112(1), 181–211 (1999)
- [20] Chiba, N., Nishizeki, T.: Planar graphs: theory and algorithms. Elsevier Science Ltd. (1988)
- [21] van den Berg, J., Stilman, M., Kuffner, J., Lin, M., Manocha, D.: Path Planning among Movable Obstacles: A Probabilistically Complete Approach. In: Chirikjian, G.S., Choset, H., Morales, M., Murphey, T. (eds.) Algorithmic Foundation of Robotics VIII. STAR, vol. 57, pp. 599–614. Springer, Heidelberg (2009)
- [22] Walsh, T.J., Goschin, S., Littman, M.L.: Integrating sample-based planning and model-based reinforcement learning. In: Proceedings of AAAI, vol. (1) (2010)
- [23] Wilfong, G.: Motion planning in the presence of movable obstacles. In: SCG 1988: Proceedings of the Fourth Annual Symposium on Computational Geometry, pp. 279–288. ACM, New York (1988)

Robust Complete Path Planning in the Plane

Victor Milenkovic, Elisha Sacks, and Steven Trac

Abstract. We present a complete path planning algorithm for a plane robot with three degrees of freedom and a static obstacle. The part boundaries consist of n linear and circular edges. The algorithm constructs and searches a combinatorial representation of the robot free space. Its computational complexity is $O((n^4 + c_3) \log n)$ with $c_3 \in O(n^6)$ the number of configurations with three simultaneous contacts between robot and obstacle edges. The algorithm is implemented robustly using our adaptive-precision controlled perturbation library. The program is fast and memory efficient, is provably accurate, and handles degenerate input.

1 Introduction

We present a complete path planning algorithm for a plane robot with three degrees of freedom and a static obstacle. The part boundaries consist of n linear and circular edges. The algorithm constructs and searches a combinatorial representation of the robot free space. Complete path planning has been deemed impractical because the free space complexity is $O(n^d)$ for an input of size n with d degrees of freedom. However, mild input restrictions reduce the complexity to $O(n)$ [17]. Our algorithm is practical for this class of inputs because it is sensitive to the reduced complexity.

Complete path planning has also been deemed impractical because it employs computational geometry algorithms that are hard to implement robustly, meaning

Victor Milenkovic

University of Miami, Coral Gables, FL

e-mail: vjm@miami.edu

Elisha Sacks

Purdue University, West Lafayette, IN

e-mail: elisha.sacks@gmail.com

Steven Track

Epic Systems Corporation, Madison, WI

e-mail: strac@epic.com

accurately and efficiently. We implement our algorithm using our adaptive-precision controlled perturbation robustness library. The program is fast and memory efficient, is provably accurate, and handles degenerate input.

Complete path planning solves the narrow passage problem of sample-based planning. Sample-based planning algorithms [8] build and search a graph whose vertices and edges are points and line segments in free space. As the sample size grows, the probability of finding a path converges to one. The outstanding problem is *narrow passages* where the robot clearance, ϵ , is small. The sample size that ensures a fixed probability of finding a path is $\Omega(\epsilon^{-d})$ for a robot with d degrees of freedom. Planning experiments confirm that narrow passages require large sample sizes in practice. Our planner is correct for any ϵ . We demonstrate that it is fast for $\epsilon = 10^{-8}$, a value that far exceeds application requirements.

Free space construction supports mechanical design [14] and part layout [11] algorithms by characterizing the space of potential robot configurations, whereas sample-based algorithms cannot provide this information.

2 Prior Work

Avnaim *et al* [1] present a free space construction algorithm for polygonal parts. Our algorithm has the same complexity even though circular edges increase the algebraic degree of the free space and complicate its combinatorial structure. Circular edges enable one to model curved parts to a given accuracy with many fewer edges and permit one to work with level sets and rotational sweeps without approximation. Sacks [13] presents a precursor to our algorithm that computes type 3 criticalities (Sec. 3) approximately, is not output sensitive, and is not robust. Stappen *et al* [18] develop efficient path planning algorithms for a translating robot with mild input restrictions. Sacks [14] computes the free space of two curved parts, in 2D or 3D, each of which rotates around or translates along a fixed axis.

Complete path planning has been implemented robustly via exact computation (Sec. 6) for translating polygons [19], translating polyhedra [5], and polygons with translation along an axis and rotation [16].

There is extensive research on sample-based planning with narrow passages. The approach closest to ours is a hybrid algorithm [21] that approximates the free space with an octree comprised of free, blocked, and mixed cells, builds a graph of free configurations for each mixed cell, and links the graphs into a global approximation of free space. In practice, this method is restricted to $d = 3$ because its computational complexity is r^{-d} with r the octree resolution.

3 Overview

The configuration space is $C = \mathfrak{R}^2 \times S$. Let $\theta M + a$ denote a robot, M , rotated by angle θ around the origin then translated by a . The free space of M with respect to an obstacle, F , is $\{(a, \theta) \in C | (\theta M + a) \cap F = \emptyset\}$. Define $-M = \{-m | m \in M\}$. When the robot translates at a fixed θ value, its free space is defined by the convex

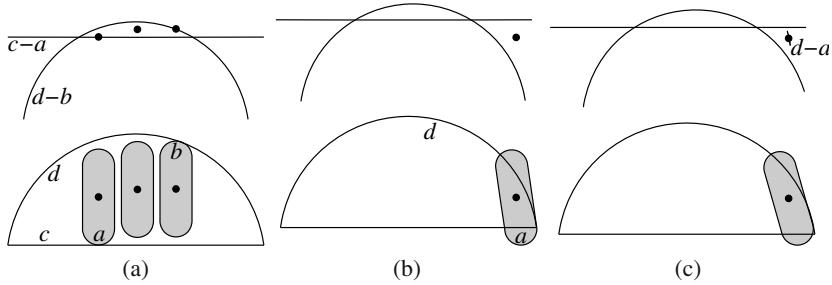


Fig. 1 Type 1 criticality: before (a), at critical angle (b), after (c)

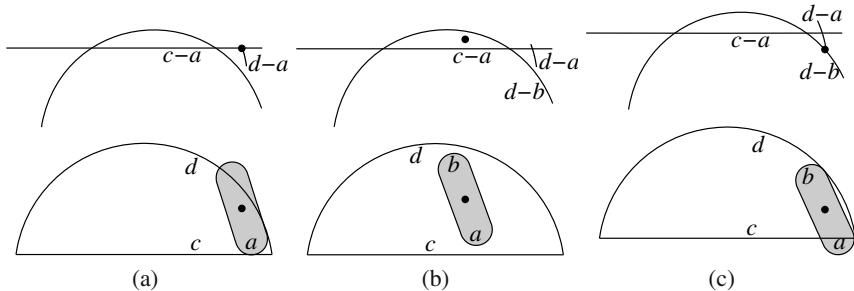


Fig. 2 Type 2 criticalities: at critical angle (a), between (b), at critical angle (c)

convolution [7] of $\theta(-M)$ and F . The convolution is the set of points, t , such that $\theta M + t$ has a local contact with F . It subdivides the plane into open regions: some regions comprise the free space and the others comprise the blocked space. Fig. 1a depicts an oval robot inside a dome-shaped room. The obstacle is the complement of the room. The convolution edges appear above. For $t \in c \oplus -a$, denoted $c - a$, $a + t$ contacts c , and similarly for b and d . The free space is the circular segment bounded by subsets of $c - a$ and $d - b$.

The subdivision is a smooth function of θ , except at a discrete set of critical angles where its structure changes. There are three types of criticality (Sec. 4), a change in: 1) the set of convolution edges, 2) the set of intersections among edges, or 3) the order of intersections on each edge.

Fig. 1b illustrates a type 1 criticality at which there exists t such that $a + t$ and d can be tangent at an endpoint (although t is in blocked space). Beyond this criticality, the edge $d - a$ appears (Fig. 1c). Figs. 2a and 2c depict the type 2 criticalities at which $d - a$ first intersects $c - a$ and $d - a$ first intersects $d - b$. Fig. 3b depicts a type 3 criticality in which $c - a$, $d - a$, and $d - b$ are coincident and a triple contact is possible. The triangle formed by $c - a$, $d - a$, and $d - b$ flips (Fig. 3a and Fig. 3c), and the boundary of the free space now contains a subset of $d - a$.

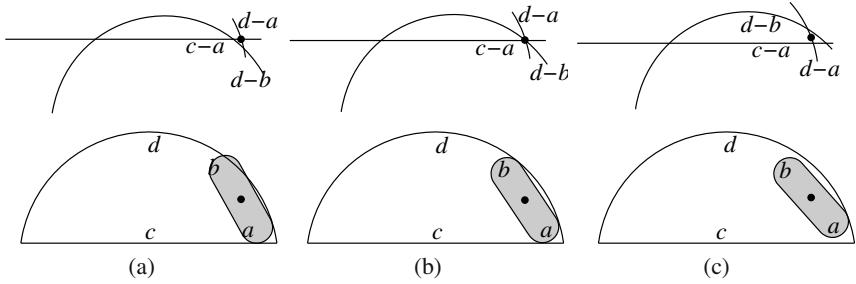


Fig. 3 Type 3 criticality: before (a), at critical angle (b), after (c)

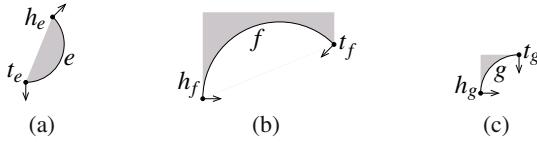


Fig. 4 Circular edges: convex e (a), concave f (b), concave $g = e \oplus f$ (c)

By sweeping a plane of constant θ , we compute a vertical decomposition of configuration space along the θ axis and extract the subset that comprises the free space (Sec. 5 and Fig. 11b). We describe the robust implementation in Sec. 6 and validate it in Sec. 7.

4 Criticality Computation

A part is a plane region with a boundary comprised of vertices and edges. A vertex, v , is a point, p_v , and an outward normal, n_v . An edge, e , is an open line segment or circular arc with tail vertex $t = t_e$ and head vertex $h = h_e$ such that $n_t \times n_h \geq 0$ (Fig. 4). A linear edge has normal $n_e = n_t (= n_h)$, normal interval $[n_e, n_e]$, and curvature $s_e = 0$. A circular edge has normal interval (n_t, n_h) , angular extent at most π , center m_e , signed radius r_e , and curvature $s_e = 1/r_e$. It is convex if $r_e > 0$ and is concave otherwise. The part interior lies to the left when a linear or convex edge is traversed from t to h , or when a concave edge is traversed from h to t .

The convex convolution [7] of F and $\theta(-M)$ consists of sum vertices and sum edges. A sum vertex, $w = v \oplus e$, is the sum of a vertex, v , on one part and a point, a , on an edge, e , of the other part such that n_v equals the normal of e at a . If $v \in F$, $\theta e \in \theta(-M)$ and $w = v \oplus \theta e$, then $p_w = p_v + \theta m_e + r_e n_v$ and $n_w = n_v$. If $\theta v \in \theta(-M)$, $e \in F$ and $w = \theta v \oplus e$, then $p_w = \theta p_v + m_e + r_e \theta n_v$ and $n_w = \theta n_v$. A sum edge, $g = e \oplus \theta f$, is the sum of edges $e \in F$ and $\theta f \in \theta(-M)$ with $s_e + s_f > 0$ whose normal intervals intersect. The curvature condition implies that F and $\theta M + t$ are in contact without local overlap for every $t \in g$, which is necessary for g to contribute to the free space boundary. Edge g is the set of sums, $p + q$, of points $p \in e$ and $q \in \theta f$ with equal normals. If e and f are circular, g is circular with $m_g = m_e + \theta m_f$,

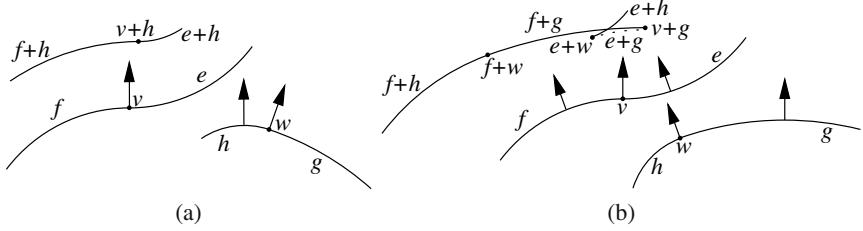


Fig. 5 Type 1 criticality: $\theta < 0$ (a), $\theta > 0$ (b)

$r_g = r_e + r_f$, $t_g = t_e \oplus \theta f$ or $t_g = \theta t_f \oplus e$, and $h_g = h_e \oplus \theta f$ or $h_g = \theta h_f \oplus e$ (Fig. 4). If one edge is linear, the other is circular by the convexity condition, so g is the offset of the linear edge by the circle. If e is circular, $n_g = \theta n_f$, $t_g = \theta t_f + m_e + r_e \theta n_f$ and $h_g = \theta h_f + m_e + r_e \theta n_f$; otherwise, $n_g = n_e$, $t_g = t_e + \theta m_f + r_f n_e$ and $h_g = h_e + \theta m_f + r_f n_e$.

Let edges e and f in counterclockwise order around the boundary of one part meet at vertices v and w , so $p_v = p_w$. If $n_v \times n_w > 0$, p_v forms sum edges with the compatible edges of the other part. These sum edges can be derived as above by introducing an artificial circular edge with tail v , head w , and radius zero [12]. However, they are subject to three type 2 criticalities not presented in the paper. We avoid this case by only working with offset shapes (Sec. 7).

4.1 Type 1 Criticality

A type 1 criticality occurs when vertices $v \in F$ and $\theta w \in \theta(-M)$ have equal normals. This criticality, denoted (v, w) , occurs at $\theta = n_v / n_w$. Here and throughout the paper, angles are equated with unit vectors and division denotes the complex quotient, so n_v / n_w is n_v rotated clockwise by the angle of n_w . The sum vertex $v \oplus \theta f$ with $f \in -M$ enters and exits the convolution at the (v, h_f) and (v, t_f) criticalities; $\theta w \oplus e$ with $e \in F$ enters and exits at the (t_e, w) and (h_e, w) criticalities. If $e \in F$ is circular and $f \in -M$ is linear, the sum edge $g = e \oplus \theta f$ enters and exits at the (t_e, n_f) and (h_e, n_f) criticalities (we use n_f instead of t_f or h_f because they have equal normals). If e is linear and f is circular, g enters and exits at the (n_e, h_f) and (n_e, t_f) criticalities. If both are circular, g enters and exits at the (t_e, h_f) and (h_e, t_f) criticalities. At the (t_e, t_f) criticality, t_g switches from $t_e \oplus \theta f$ to $\theta t_f \oplus e$. At the (h_e, h_f) criticality, h_g switches from $\theta h_f \oplus e$ to $h_e \oplus \theta f$.

Fig. 5 shows edges $e, f \in F$ that meet at v and $g, h \in \theta(-M)$ that meet at w . The (v, w) critical angle is $\theta = 0$, sum vertex $v \oplus h$ exits, sum vertices $v \oplus g$, $w \oplus e$, and $w \oplus f$ enter, and sum edge $f \oplus g$ enters. There is no $e \oplus g$ edge (drawn dashed) because $s_e + s_g < 0$. The normal vector out of v is vertical. The point on g with a parallel normal vector sums with v to form $v \oplus g$. Similarly, for $w \oplus e$, etc.

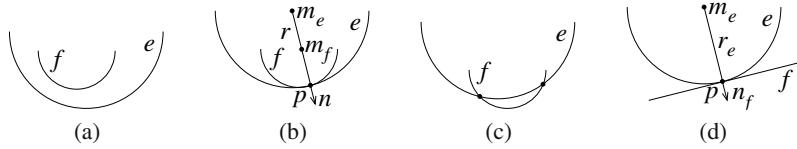


Fig. 6 Circle/circle (a–c) and circle/line (d) tangencies

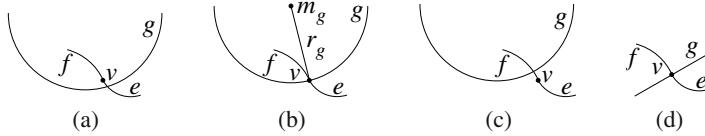


Fig. 7 Circle/circle (a–c) and circle/line (d) hits

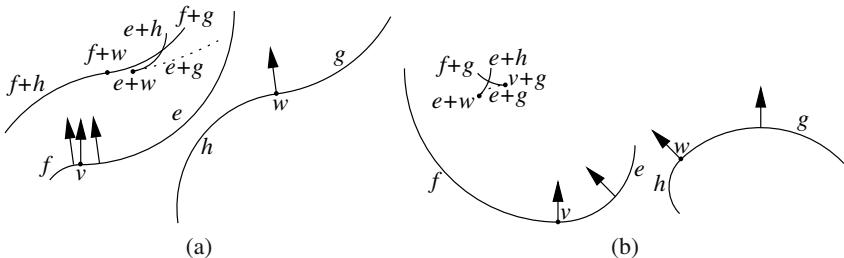


Fig. 8 Simultaneous type 1 and 2: Case 2 (a), Case 3 (b)

4.2 Type 2 Criticality

A type 2 criticality occurs when two sum edges are tangent (Fig. 6) or when a sum vertex hits a sum edge (Fig. 7). At a tangency, two vertices enter or exit the subdivision. At a hit, each incident edge gains or loses a vertex. A candidate tangency occurs when the lines or circles of two edges are tangent. It is a criticality when the point of tangency lies on both edges. Likewise for hits.

The tangency equations for circular edges e and f with $|r_e| > |r_f|$ (Fig. 6b) are $\|m_f - m_e\| = r$ with $r = |r_e| \pm |r_f|$. Let $m_e = a + \theta b$, $m_f = c + \theta d$, $u = c - a$, and $v = d - b$. Let b_i be the intersection points of the circle with center $(0,0)$ and radius r , and the circle with center u and radius $\|v\|$. The e normals at the tangent points are $n_i = b_i/r_e$. The f normals for $r = |r_e| \pm |r_f|$ are $\mp \text{sign}(r_e r_f) n_i$. The critical angles are $\theta_i = n_i/(v/\|v\|)$. The critical points are $p_i = a + \theta_i b + r_e n_i$.

If f is linear, the distance from m_e to the f line equals $|r_e|$ (Fig. 6d). Define a linear trigonometric expression (LTE) as $k_1 \sin \theta + k_2 \cos \theta + k_3$ with the k_i constants. The tangency equations are LTE's: $n_f \cdot m_e + d = \pm r_e$ or $\theta n_f \cdot m_e + d = \pm r_e$ with $m_e = \theta_a + b$ and with d an LTE. We can solve for θ then compute p as before. A tangency between linear edges is degenerate.

The hit equations for sum vertex v and circular edge g are $\|m_g - p_v\| = |r_g|$ (Fig. 7b). If g is linear, the equations are $n_g \cdot p_v + d = 0$ or $\theta n_g \cdot p_v + d = 0$ with d an LTE (Fig. 7d). The solutions are the same as for the tangency equations.

A type 1 criticality that coincides with a circular edge hit is not counted as a type 2 criticality. There are three cases based on the signed radii of the incident edges. We omit the proof that these are the only cases up to symmetry. Let edges $e, f \in F$ meet at v and $g, h \in -M$ meet at w . The analysis uses a coordinate system in which n_v is the y axis and the critical angle is 0.

Case 1 is $r_e < 0, r_f, r_g, r_h > 0, r_g + r_e > 0, r_h + r_e < 0$ (Fig. 5). Edge $f \oplus g$ is above $e \oplus g$ (drawn dashed) because they are tangent and $r_f + r_g > r_g > r_e + r_g$. Since $r_g + r_e > 0$, $e \oplus g$ is concave downward and hence $e + w$ with normal n_w is to the left of $v + g$ with normal n_v . Edge $e \oplus h$ is above $e \oplus g$ because it is tangent to $e \oplus g$ at $e \oplus w$ and it is concave upward ($r_h + r_e < 0$). Hence, $f \oplus g$ intersects $e \oplus h$ for all sufficiently small $\theta > 0$.

Case 2 is $r_e, r_g < 0, r_f, r_h > 0, r_f + r_g < r_e + r_h < 0$ (Fig. 8a). In the coordinate system in which n_w is up, $f \oplus w$ and $e \oplus w$ are the highest and lowest points of f and e , respectively, added to w . Hence, $e + w$ is below and to the right of $f + w$. But $e + w$ is also a lowest point of $f \oplus g$, hence $e \oplus w$ is below $f \oplus g$. For all sufficiently small $\theta > 0$ hence $e \oplus w$ sufficiently close to $f \oplus w$, $e \oplus h$ intersects $f \oplus g$ because $r_f + r_g < r_e + r_h < 0$.

Case 3, $r_e, r_f < 0, r_g, r_h > 0, r_e + r_h < 0, r_e + r_g > 0, r_f + r_g < 0$ (Fig. 8b), is similar to Case 1. Edges $f + g$ and $e + h$ are both externally tangent to $e + g$ (dashed), hence they intersect for all sufficiently small $\theta > 0$.

4.3 Type 3 Criticality

A type 3 criticality occurs when three edges intersect at a point. For circular edges e, f , and g (Fig. 9a), let $\phi_{ef} = \angle m_e p_m f$ and $d_{ef} = m_f - m_e = u + \theta v$. By the law of cosines and the identity $(\theta u) \cdot (\theta v) = u \cdot v$,

$$\cos \phi_{ef} = \frac{r_e^2 + r_f^2 - d_{ef} \cdot d_{ef}}{2r_e r_f} = \frac{r_e^2 + r_f^2 - u \cdot u - v \cdot v - 2u \cdot \theta v}{2r_e r_f}. \quad (1)$$

Define ϕ_{fg} and ϕ_{ge} likewise. The equation

$$\cos^2 \phi_{ef} + \cos^2 \phi_{fg} + \cos^2 \phi_{ge} - 2 \cos \phi_{ef} \cos \phi_{fg} \cos \phi_{ge} = 1$$

follows from $\phi_{ef} + \phi_{fg} + \phi_{ge} = 2\pi$, which implies $\cos \phi_{ef} = \cos(\phi_{fg} + \phi_{ge})$, and from the identity $\cos(x + y) = \cos x \cos y - \sin x \sin y$. Replace $\cos \phi_{ef}$ with the rightmost expression in Eq. 1 and replace $\cos \phi_{fg}$ and $\cos \phi_{ge}$ likewise to obtain the criticality equation, a cubic in $\cos \theta$ and $\sin \theta$.

If g is linear (Fig. 9b), let $\phi_{eg} = \angle p_m e q$ with q the projection of m_e onto g . Since $\cos \phi_{eg} = d/r_e$ with d the distance from m_e to g , it is an LTE. Define ϕ_{fg} likewise, define ϕ_{ef} as before, and use $\phi_{eg} + \phi_{fg} = \phi_{ef}$ to obtain a cubic. If f and g are linear (Fig. 9c), let ϕ_{ef} and ϕ_{eg} be the angles between $p_m e$ and the projections onto f

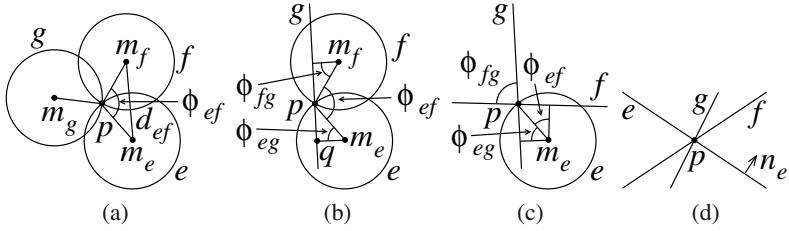


Fig. 9 Type 3 criticality involving 3 (a), 2 (b), 1 (c), and 0 (d) circular edges

and g , let ϕ_{fg} be the angle between f and g , and use $\phi_{ef} + \phi_{eg} = \phi_{fg}$ to obtain a cubic. If e , f , and g are linear (Fig. 9d), their line equations are $n_e \cdot (x, y) + d_e = 0$ with n_e and d_e LTE's and likewise for f and g . The criticality equation,

$$d_e(n_f \times n_g) - d_f(n_e \times n_g) + d_g(n_e \times n_f) = 0,$$

is quadratic in $\sin \theta$ and $\cos \theta$.

Given θ , the edge intersection point, p , is the solution of two linear equations: $e - f = 0$ and $e - g = 0$ for three circles, $e - f = 0$ and $g = 0$ for two circles and a line, and $f = 0$ and $g = 0$ otherwise. A criticality occurs if p lies on the three edges.

5 Free Space Construction

The free space construction algorithm consists of two parts. Part 1 is a plane sweep that computes the sum edges and their intersection points as θ increases from 0 to 2π (Sec. 5.1). Part 2 computes the free space boundary (Sec. 5.3). Part 1 dominates the computational complexity and the actual running time. We prove that it is output sensitive (Sec. 5.2).

5.1 Plane Sweep

Step 1 of the first part calculates the convolution edges and their intersections for $\theta = 0$. Each edge has a list of its endpoints and its intersections with other edges, ordered from tail to head. Each intersection is an ordered pair, (e, f) , of sum edges, denoting e crosses f from left to right, to distinguish it from the (f, e) intersection. Step 2 initializes a priority queue with all the type 1 and type 2 criticalities, and with the type 3 criticalities of the initial candidate triangles. A candidate is three sum edges each of whose lists contains adjacent intersection points with the other two. Step 3 handles the criticalities in increasing θ order. The output is the the initial set of lists plus the edit that occurs at each criticality.

A type 1 criticality is handled by adding or removing vertices and edges, and by updating intersection lists. When a linear edge is added or removed, its intersection points with the other edges at the critical angle are added to or removed from the

appropriate lists. When a circular edge with a coincident hit is added (Figs. 5b, 8a, 8b), a point is appended to one end of its list. A type 2 criticality is handled by updating the two intersection lists. For a tangency, two points are inserted or removed in each intersection list. For a hit, a point is appended to one end of the list or is removed. A type 3 criticality is handled by swapping three pairs of incident points on the lists of the three sum edges of the triangle. After each update, newly created candidate triangles are checked for type 3 criticalities which are added to the priority queue.

5.2 Analysis

The sweep algorithm is correct if every change in the structure of the subdivision is one of the three criticalities. This condition holds when every criticality equation has simple roots and every criticality occurs at a unique θ value. Correctness follows by verifying the criticality handling.

There are $m \in O(n^2)$ sum vertices and sum edges with n the input size. The complexity of the $\theta = 0$ slice is $c_0 \in O(m^2)$. There are $c_i \in O(n^{2i})$ type i criticalities. Hence, the complexity of the output is $N \in O(c_0 + c_1m + c_2 + c_3)$ because each criticality adds $O(1)$ complexity, except for type 1 criticalities that add or remove a line and hence add $O(m)$ complexity. The number of candidate triangles is $O(N)$ because each newly adjacent pair of intersections in a list defines a single candidate.

We can perform step 1 in $O(c_0 \log n)$ time with a sweep algorithm. Computing the type 1 criticalities takes $O(c_1)$ time. Computing the type 2 criticalities takes $O(n^4)$ time, so step 2 takes $O(n^4 \log n)$ time. Step 3 takes $O(N \log n)$ time using binary search on edge lists to handle criticalities. Since $N \in O(n^4 + c_3)$, the algorithm running time is $O((n^4 + c_3) \log n)$. The algorithm is output sensitive in that the dominant cost is proportional to c_3 , which is the number of configurations with simultaneous convex contacts between three pairs of moving/fixed part edges.

5.3 Faces, Shells, Cells, and Path Planning

We define a *subedge* (edge in the translational subdivision) to be an adjacent pair of elements (in a particular order) in the list of a sum edge. This pair is adjacent for a θ interval. The subedge plus its interval corresponds to a *face* in configuration space. For example, in Fig. 3a, sum edge $d - a$ ($d \oplus \theta(-a)$) is split into three subedges. The top subedge was created at the Fig. 2a criticality and the others at the Fig. 2c criticality. All three subedges end at the Fig. 3b criticality where the intersection points on $d - a$ swap.

Two faces are *horizontal neighbors* if they share a sum edge endpoint or if they share an edge intersection point and are on the sides of the outward normals of the intersecting edges. Hence, in Fig. 1, the middle subedges of $c - a$ and $d - b$ are neighbors because the normals point upward from $c - a$ and downward from $d - b$. In Fig. 3a, the intersection of $c - a$ and $d - a$ is in blocked space, but the subedges

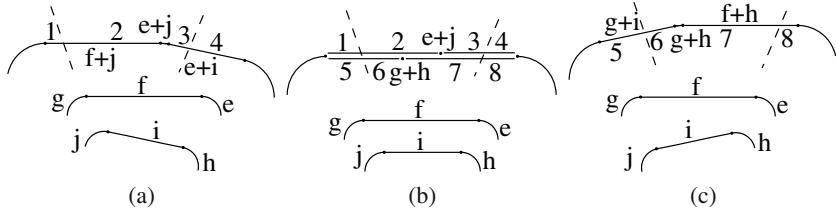


Fig. 10 Sums of linear edges before (a), at (b), and after (c) a criticality

to the left and above the intersection are neighbors because the outward normal to $d - a$ points to the left.

Two faces are *vertical neighbors* if the same criticality ends one and starts the other, they belong to the same subedge, and they share an endpoint. Hence the lower subedge of $d - a$ in Fig. 2b is a vertical neighbor of the trivial subedge $d - a$ in 1c because they share the tail of $d - a$. The former is a neighbor of the middle subedge of $d - a$ in 3a because they share the intersection of $c - a$ and $d - a$ as an upper endpoint. Two linear subedges can also be neighbors if they are subsets of *neighboring sum edges* (defined below) and they share a *corresponding list element* at the same endpoint or if one contains an endpoint of the other at the criticality. List elements of linear edges e and f correspond if they are both intersections with the same third sum edge g or if they are a *corresponding endpoint* (defined below).

If the same type 1 event (u, v) ends one linear sum edge and starts another, and either u or v is *not* the endpoint of a linear edge, then the two sum edges are neighbors and their tail and heads correspond. Otherwise, the situation is as depicted in Fig. 10. Before the criticality, linear sum edges $e + i$ and $f + j$ are joined by circular sum edge $e + j$. After, circular $g + h$ joins linear $f + h$ and $g + i$. Sum edges $e + j$ and $g + h$ shrink to points at the criticality. Linear sum edges $f + j$ and $e + i$ are neighbors of $g + i$ and $f + h$, and the tails of $e + i$ and $f + h$ (right endpoints) and the heads of $f + j$ and $g + i$ (left endpoints) correspond. Suppose another sum edge (dashed) splits $f + j$ into subedges 1 and 2 and $g + i$ into 5 and 6 and suppose yet another splits $e + i$ into 3 and 4 and $f + j$ into 7 and 8. Subedges 1, 2, 3, and 4 are neighbors of 5, 6, 7, and 8, respectively, because of corresponding list element endpoints. In addition 2 is a neighbor of 7 because 2 contains $g + h$ (also 7 contains $e + j$) at the criticality. If one or more of the arcs e, g, h, j are concave then some of the linear sum edges might be missing. If, for example, $e + i$ is missing, then the tail of $f + h$ has no corresponding endpoint.

We group the faces into connected components via graph traversal. The *shells* are the components where each face has a neighbor at every boundary point. Specifically, it should have two vertical neighbors. If it is linear, it should have both horizontal neighbors. If it contains a corresponding endpoint, as do subedges 2 and 7 in Fig. 10, it should have two neighbors at that criticality.

We select a sample point (t, θ) on each shell and discard the shell if $\theta M + t$ intersects F except at the point of tangency. Next, we compute the shell nesting order by ray casting. We pick a ray orthogonal to the θ axis, so ray/face intersection

reduces to ray/edge intersection. The result is a boundary representation of the cells (connected components) of the free space.

The path planner reports failure if the start and goal configurations are in different cells; otherwise it finds a path with a bug algorithm [9].

6 Robustness

The free space construction algorithm is formulated in the real-RAM model where real arithmetic has unit cost. The control logic is expressed in terms of predicates: polynomials in the input parameters whose signs are interpreted as truth values. The robustness problem is how to implement real-RAM algorithms accurately and efficiently. We wish to use floating point arithmetic, and a numerical solver for the criticality equations, because these are accurate, fast, and memory efficient. But even a tiny computation error can cause a predicate to be assigned the wrong sign, which can create a large error in the algorithm output. We wish to handle all inputs, whereas the real-RAM algorithm requires simple criticalities with unique θ values.

6.1 Prior Work

The mainstream robustness method is to evaluate predicates exactly via algebraic computation [20]. Algebraic computation increases bit complexity, hence running time. Floating point filtering techniques somewhat reduce this cost [2]. Exact evaluation cannot assign a sign to a *degenerate* predicate whose exact value is zero. Degeneracy is common due to design constraints and to symmetry.

The other popular robustness method, controlled perturbation (CP) [6, 4], evaluates predicates with floating point arithmetic. The computed sign of $f(a)$ is correct, and the predicate is called safe, when the computed magnitude satisfies $|f(a)| > \varepsilon$ with ε a function of f and a . The numerical input to the algorithm is perturbed randomly by up to δ and the algorithm is executed. If every predicate is safe, the output is returned. Otherwise, the algorithm is rerun with a different δ . The final δ bounds the error due to replacing the true input with a verifiable input. This error is inconsequential when δ is less than the required accuracy of the application that provides the input, such as the tolerances in path planning.

CP is faster than exact computation because all computations are in floating point. Whereas degenerate predicates defeat exact computation, CP handles them just like non-degenerate predicates. Their perturbed values are of order δ , so they are verifiable for reasonable δ values. However, CP performs poorly on *singular* predicates whose value and gradient are both zero. A singular predicate of degree d requires δ of order $\sqrt[d]{\varepsilon}$. This error is unacceptable with the reported ε values and is marginal even with ε equal to the rounding unit, $\approx 10^{-16}$.

Melhorn *et al* [10] handle singular predicates by rerunning the algorithm with extended precision arithmetic, which uses much more time and space than floating

point arithmetic. We [15] identify the cases where the predicates in a Minkowski sum algorithm can be singular and replace them by non-degenerate predicates in defined parameters. This approach does not generalize.

6.2 Adaptive-Precision Controlled Perturbation

We have developed an extension of CP, adaptive-precision controlled perturbation (ACP), that handles singular predicates by selectively increasing the arithmetic precision and that supports *implicit parameters* that denote the roots of polynomials.

The user specifies the required accuracy, δ . ACP replaces each input parameter, x , by $x + d$ with d uniform in $[-\delta, \delta]$. The user defines explicit parameters using arithmetic operators on prior parameters. ACP assigns them values using interval arithmetic. ACP computes predicates using the interval filter: evaluate the predicate polynomial in interval arithmetic to obtain $[l, u]$, return -1 if $u < 0$, return 1 if $l > 0$, and fail otherwise. If a predicate fails, the program reevaluates it after increasing the precision of its arguments. Each geometric object has pointers to the objects on which it depends. For example, a circle/line hit criticality (Fig. 7) points to the circle and the line. The program traverses these pointers and increases the precision recursively. The initial precision is double float and higher precision is implemented using MPFR [3].

An implicit parameter, x , is a root of a polynomial, f . ACP isolates the roots of f on an interval, $[L, U]$, using a straightforward recursive algorithm that is $O(d^2)$ in the degree, d , of f (there are faster algorithms in the literature). We describe how to implement this algorithm in interval arithmetic. If $d \leq 2$, use the explicit roots. If $d > 2$, let x_1, \dots, x_n be the roots of f' with $x_0 = L$ and $x_{n+1} = U$. For each pair, x_{i-1}, x_i , calculate $\text{sign}(f(x_{i-1}))$, $\text{sign}(f(x_i))$. If either ACP sign test fails, restart with higher precision. If $\text{sign}(f(x_{i-1})) \neq \text{sign}(f(x_i))$, shrink the root isolating interval $[x_{i-1}, x_i]$ using interval Newton's method. ACP increases the precision of x by further shrinking its interval after increasing the precision of f .

We use ACP to implement the free space construction algorithm. The input parameters specify the part boundary geometry. The type 3 critical angles are implicit parameters. Their polynomials are obtained by substituting the appropriate charts of the rational parameterization of the unit circle into the bivariate criticality equations in Sec. 4.3. The rest of the implementation follows the real-RAM algorithm. The implementation handles any input and generates an output that is correct for a δ -perturbation of the input.

7 Validation

We validate the path planning algorithm by constructing a maximal clearance path for a given robot, obstacle, and start/goal configurations. We find the largest number, s , for which the algorithm finds a path for s -offsets of the parts. The s -offset of a part is its Minkowski sum with an s -disk centered at the origin. A path for the offset parts is a $2s$ -clearance path for the original parts. We compute the maximal s by

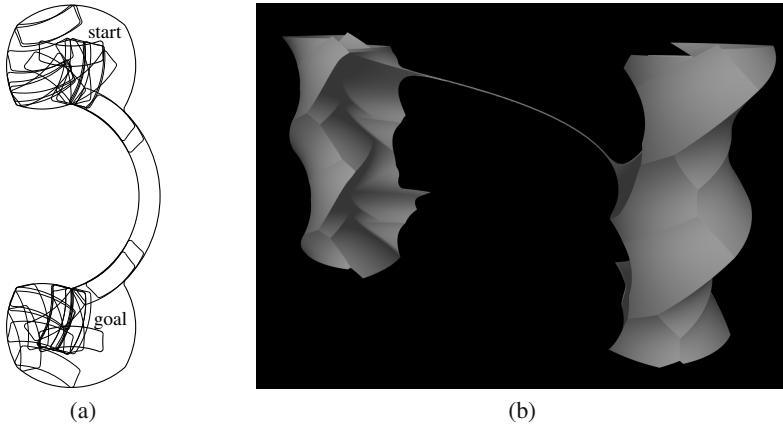


Fig. 11 Tightest clearance path $s = 0.27865$ in 2D (a). View of free space boundary from blocked space for $s = 0.265625$ (b).

bisection search on $[0, 1]$ to the floating point resolution, which takes 53 iterations. The clearance is $s - O(\delta)$ because the $2s$ -path is correct for a δ -perturbation of the input parameters, which causes an $O(\delta)$ perturbation of the parts.

The bisection algorithm tests how our program handles degenerate input. A zero-clearance path is degenerate because the robot has multiple simultaneous contacts with the obstacle. Hence, the final iterations of the algorithm are nearly degenerate. In every test, we obtain a correct output and the running time increases modestly from the first iteration to the last iteration. We use $\delta = 10^{-8}$ and extended precision of $p = 250$ binary digits. Table 1 shows the results.

Test 1 is an 8-edge robot and an 18-edge obstacle with two chambers connected by a narrow passage (Fig. 11). The start and goal configurations are in the top and bottom chambers. We ensure that the robot simultaneously rotates and translates for large s values, which forces the planner to search a large fraction of the configuration space, by making the passage boundary arcs concentric. Fig. 11a shows one sample configuration per face visited in the tightest clearance path, and Fig. 11b shows the 3D free space boundary for a smaller s so the narrow passage is visible.

Test 2 is a 5-pointed star with 15 edges inside a 6-sided container with 42 edges (Fig. 12a–b). The start and goal positions are identical, the start angle is 0° , and the goal angle is 72° ($1/5$ of a turn). The solution is for the star to pivot about each of its tips in turn. Fig. 12b shows it part way through a pivot. Sliding contacts occur at 1, 2, 3, and 4. The other two tips are close but not in contact with the obstacle. This motion is degenerate because three contacts should fix the configuration. For $s = 0.05$, there is no degenerate contact and no extended precision arithmetic is required ($r_e = 0$ and $p_e = 0$). The motion is doubly degenerate at the maximum offset, $s = 0.1$, because four points touch the container throughout each pivot. The input perturbation eliminates the degeneracy, but significant extended precision arithmetic

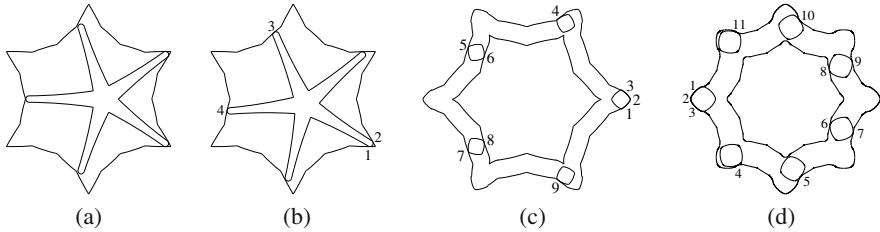


Fig. 12 Star in container: start configuration (a) and pivoting (b); wheel in channel (c), (d)

Table 1 Results: test i , iteration k , m sum edges, e edges in the subdivision, f free faces, t running time in seconds for one core of an Intel Core 2 Duo, t_{12} time computing type 1 and 2 criticalities, $c = 10^6 t / (\text{elge})$, $c_{12} = 10^3 t_{12} / m$, r_f and r_e number of polynomials with floating point and extended precision root isolation, and p_f and p_e number of floating point and extended precision predicate evaluations

| i | k | m | e | f | e/f | t | t_{12} | c | c_{12} | r_f | r_e | p_f | p_e |
|-----|-----|--------|--------|--------|-------|-----|----------|-----|----------|--------|-------|-------|--------|
| 1 | 1 | 70 | 2,000 | 450 | 4.4 | 0.0 | 0.0 | 0 | 0 | 14 | 0 | 1.4e5 | 900 |
| 1 | 53 | 70 | 2,000 | 450 | 4.4 | 0.0 | 0.0 | 0 | 0 | 12 | 0 | 1.4e5 | 800 |
| 2 | 1 | 400 | 26,000 | 1,000 | 26.0 | 0.4 | 0.3 | 1.0 | 0.8 | 600 | 0 | 2.7e6 | 0 |
| 2 | 53 | 400 | 36,000 | 2,000 | 18.0 | 2.6 | 0.3 | 4.8 | 0.8 | 1,700 | 400 | 3.7e6 | 3.2e5 |
| 3 | 1 | 700 | 89,000 | 3,000 | 29.7 | 1.4 | 0.9 | 1.0 | 1.3 | 3,200 | 0 | 8.9e6 | 0 |
| 3 | 53 | 700 | 1.0e5 | 4,000 | 25.0 | 1.8 | 1.0 | 1.1 | 1.4 | 5,600 | 5 | 1.1e7 | 0 |
| 4 | 1 | 3,000 | 48,000 | 3,000 | 16.0 | 1.4 | 0.9 | 1.9 | 0.3 | 5,300 | 60 | 1.2e7 | 45,000 |
| 4 | 35 | 2,500 | 1.0e5 | 3,000 | 33.3 | 2.6 | 1.0 | 1.6 | 0.4 | 3,300 | 200 | 1.5e7 | 1.7e5 |
| 5 | 1 | 4,400 | 1.4e5 | 15,000 | 9.3 | 3.1 | 1.8 | 1.3 | 0.4 | 3,500 | 36 | 2.7e7 | 68,000 |
| 5 | 22 | 3,900 | 4.1e5 | 10,000 | 41.0 | 9.7 | 2.1 | 1.3 | 0.5 | 24,000 | 335 | 4.6e7 | 6.9e5 |
| 6 | 1 | 8,000 | 2.2e5 | 20,000 | 11.0 | 6.1 | 3.7 | 1.6 | 0.5 | 2,800 | 300 | 5.4e7 | 2.1e5 |
| 6 | 34 | 6,800 | 5.1e5 | 11,000 | 46.4 | 13 | 3.9 | 1.3 | 0.6 | 21,000 | 900 | 7.0e7 | 9.0e5 |
| 7 | 1 | 13,000 | 5.5e5 | 31,000 | 17.7 | 14 | 8.9 | 1.3 | 0.7 | 12,000 | 200 | 1.4e8 | 2.1e5 |
| 7 | 27 | 12,000 | 1.4e6 | 28,000 | 50.0 | 33 | 10 | 1.2 | 0.8 | 81,000 | 1,700 | 2.1e8 | 1.7e6 |
| 8 | 1 | 18,000 | 7.1e5 | 75,000 | 9.5 | 22 | 16 | 1.6 | 0.9 | 12,000 | 300 | 2.3e8 | 2.6e5 |
| 8 | 24 | 15,000 | 1.5e6 | 55,000 | 27.3 | 55 | 14 | 1.8 | 0.9 | 72,000 | 3,600 | 2.6e8 | 5.0e6 |

still occurs ($r_e = 400$ and $p_e = 320,000$). Thanks to adaptive precision, running time only increases by a factor of six.

Test 3 is a 7-pointed star inside an 8-sided container. Unlike test 2, there is no sliding contact, so the input is not degenerate, the $s = 0.1$ offset is simply degenerate, and there is little extended precision arithmetic.

Tests 4–8 are an n -pin wheel inside a channel with $n+1$ segments for $n = 3, \dots, 7$. Figs. 12c and d show $n = 5$ and $n = 7$ and the points of contact during a pivot. The bisection search ends after $k < 53$ iterations because the start and goal become δ -close to blocked space. At minimum clearance (maximum k), symmetry causes multiple pin/channel contacts. Hence, there is significant extended precision arithmetic (r_e) for large k . Even so, the time increases by at most a factor of three.

As m increases, the time, t_{12} , for computing type 1 and 2 criticalities is a smaller fraction of the total. This is the $n^4 = m^2$ component of the predicted

$O((n^4 + c_3) \log n)$ running time. Although type 2 criticality computation is $O(m^2)$, we reduce the actual running time to $O(m)$ using kd-trees, bounding boxes on the area swept by a sum edge over its θ interval, and a geometric constraint on the relative sizes of the constituent robot and obstacle edges of intersecting sum edges. Hence, $c_{12} = 10^3 t_{12}/m \approx 1$, and $O(c_3 \log n)$ more accurately models the time. Looking at the value $c = 10^6 t/(e \lg e)$, we see that $c < 2$ except for an anomalous value of 4.8. Since, c_3 and e are proportional, these values agree with the model. The actual output size, f , is much smaller than e because most faces do not appear on the free space boundary. The ratio e/f ranges from 4.4 to 50. To this extent, the algorithm is not truly output sensitive.

8 Discussion

We have presented a complete path planning algorithm for a planar robot with three degrees of freedom where the robot and the obstacle boundaries consist of circular and linear edges. We have implemented the algorithm using our ACP robustness library. The program is complete in that for any input the output is correct for a δ -deformation of the input. By setting $\delta = 10^{-8}$, we make the deformed input indistinguishable from the actual input. We have demonstrated that the program is fast and output sensitive on problems that contain narrow passages. The running time is only slightly dependent on the passage width, ϵ , whereas the cost of sample-based planning is ϵ^{-d} with d the configuration space dimension.

We are developing a complete, output sensitive path planning algorithm for a polyhedral robot that translates freely and rotates around a fixed axis ($d = 4$). Our plane sweep generalizes to a volume sweep. The challenge is to derive the criticality equations and the subdivision update rules. Our next goal is to develop a hybrid algorithm for a polyhedral robot with six degrees of freedom whose complexity is $O(1/\epsilon^2)$ and that performs well on narrow channels. We will sample the configuration space with $O(1/\epsilon^2)$ $d = 4$ subspaces, construct their free spaces, and link them into a path planning graph.

Acknowledgements. This material is based upon work supported by the National Science Foundation under Grant Nos. 0904707 (Milenkovic) and 0904832 (Sacks). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. Fig. 11b created by Justin Stoecker.

References

1. Avnaim, F., Boissonnat, J.D.: Polygon placement under translation and rotation. *Informatique Théorique et Applications* 31(1), 5–28 (1989)
2. Exact computational geometry, <http://cs.nyu.edu/exact>

3. Fousse, L., Hanrot, G., Lefèvre, V., Péliſſier, P., Zimmermann, P.: MPFR: A multiple precision binary floating point library with correct rounding. *ACM Transactions on Mathematical Software* 33 (2007)
4. Funke, S., Klein, C., Mehlhorn, K., Schmitt, S.: Controlled perturbation for delaunay triangulations. In: *SODA 2005: Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1047–1056. ACM, Society for Industrial and Applied Mathematics, Philadelphia (2005)
5. Hachenberger, P.: Exact minkowski sums of polyhedra and exact and efficient decomposition of polyhedra into convex pieces. *Algorithmica* 55, 329–345 (2009)
6. Halperin, D., Leiserowitz, E.: Controlled perturbation for arrangements of circles. *International Journal of Computational Geometry and Applications* 14(4-5), 277–310 (2004)
7. Kaul, A., O'Connor, M.A., Srinivasan, V.: Computing Minkowski sums of regular polygons. In: *Proceedings of the Third Canadian Conference on Computational Geometry*, pp. 74–77 (1991)
8. LaValle, S.M.: *Planning Algorithms*. Cambridge University Press (2006)
9. Lumelski, V.J., Stepanov, A.A.: Dynamic path planning for a mobile automaton with limited information on the environment. *IEEE Transactions on Control AC-31(11)*, 1058–1063 (1986)
10. Mehlhornlow, K., Osbilda, R., Sagraloffa, M.: A general approach to the analysis of controlled perturbation algorithms. *Computational Geometry* 44(9), 507–528 (2011)
11. Milenkovic, V., Daniels, K.: Translational polygon containment and minimal enclosure using mathematical programming. *International Transactions in Operational Research* 6, 525–554 (1999)
12. Milenkovic, V., Sacks, E.: Two approximate minkowski sum algorithms. *International Journal of Computational Geometry and Applications* 20(4), 485–509 (2010)
13. Sacks, E.: Practical sliced configuration spaces for curved planar pairs. *International Journal of Robotics Research* 18(1), 59–63 (1999)
14. Sacks, E., Joskowicz, L.: The configuration space method for kinematic design of mechanical systems. MIT Press (2010)
15. Sacks, E., Milenkovic, V., Kyung, M.H.: Controlled linear perturbation. *Computer-Aided Design* 43(10), 1250–1257 (2011)
16. Salzman, O., Hemmer, M., Raveh, B., Halperin, D.: Motion planning via manifold samples. In: *Proceedings of the 19th European Symposium on Algorithms* (2011)
17. van der Stappen, A.F., Halperin, D., Overmars, M.H.: The complexity of the free space for a robot moving amidst fat obstacles. *Computational Geometry Theory and Applications* 3, 353–373 (1993)
18. van der Stappen, A.F., Overmars, M.H., de Berg A1, M., Vleugels, J.: Motion planning in environments with low obstacle density. *Discrete and Computational Geometry* 20(4), 561–587 (1998)
19. Wein, R.: Exact and efficient construction of planar Minkowski sums using the convolution method. In: *Proceedings of the 14th Annual European Symposium on Algorithms*, pp. 829–840 (2006)
20. Yap, C.: Robust geometric computation. In: Goodman, J.E., O'Rourke, J. (eds.) *Handbook of Discrete and Computational Geometry*, 2nd edn., ch. 41, pp. 927–952. CRC Press, Boca Raton (2004)
21. Zhang, L., Kim, Y.J., Manocha, D.: A hybrid approach for complete motion planning. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 7–14 (2007)

Configurations and Path Planning of Convex Planar Polygonal Loops^{*}

Li Han, Lee Rudolph, Michael Chou, Sean Corbett, Emily Eagle,
Dylan Glotzer, Jake Kramer, Jonathan Moran, Christopher Pietras,
Ammar Tareen, and Matthew Valko

Abstract. Polygonal loops are interesting both as classical geometric objects and in modeling practical engineering systems, *e.g.*, grasping systems with fingers having planar revolute joints. Convex loop configurations and path planning between them are important since many naturally occurring manipulation poses for human and robotic hands are convex or close to convex, and current collision-free path planning methods for polygonal loops use convex configurations in intermediate steps. We prove that, in a set of triangle-based parameters, the space *CConvex* of convex configurations of a planar polygonal loop with fixed edge lengths and orientation, and one link pinned to the plane, is **star-shaped** with respect to an easily computed triangular configuration; with a further condition on edge lengths, *CConvex* is actually a **convex polyhedron**. Thus reconfiguration between identically oriented convex configurations of a planar polygonal loop can be achieved by one or two straight-line motions within *CConvex*. We conjecture that, in our parameter space, the straight-line motion joining any two such configurations passes through only non-self-intersecting configurations, although it may leave *CConvex*. These results are substantially simpler and more efficient than prior work, and demonstrate the importance of suitable system parametrization.

Li Han · Lee Rudolph

Dept. of Mathematics and Computer Science, Clark University, Worcester, MA 01610
e-mail: {lhan, lrudolph}@clarku.edu

Michael Chou

Wesleyan University, Middletown, CT
e-mail: michael.chou@uconn.edu

Sean Corbett · Emily Eagle · Dylan Glotzer · Jake Kramer ·

Jonathan Moran · Christopher Pietras · Ammar Tareen · Matthew Valko
participated in the project as undergraduate research assistants.

Clark University, Worcester, MA

e-mail: {scorbett, jkramer, jmoran, cpietras}@clarku.edu,
eeagle@ahpnet.com, dylanglotzer@gmail.com,
atareen@physics.bu.edu, mvalko@schoolph.umass.edu

* The research reported here is supported in part by NSF grant IIS-0713335 and its REU supplements.

1 Introduction

Configurational problems are ubiquitous in the study of robotics—robots carry out their tasks by moving themselves and/or environmental objects around, which corresponds to changes of configurations of the robotic system. The set of all configurations of a system is its *configuration space* (*CSpace* for short) [17]. For a wide range of robotic problems [7, 15, 22, 20, 19, 5, 16], such as motion planning and kinematics-based system design, efficient algorithms for dealing with system configurations and analyzing *CSpace* structures are very useful.

Although system configurations are conceptually independent of the parameters used to describe them, the details of their study are strongly affected by the choice of parameters. Different choices lead to different formulations of system constraints, and to different ways of viewing *CSpace* structures. As an example, consider the systems addressed in this paper—planar loops with revolute joints (or, for simplicity, planar polygonal loops). For these, conventional constraint formulations are *non-linear equations*: quadratic equations in Cartesian coordinates for fixed link lengths, trigonometric equations in angular coordinates for the loop closure. Consequently, system configuration spaces in these traditional parameters are non-linear semi-algebraic or semi-analytic subsets (generically smooth manifolds) of higher-dimensional ambient spaces, and have complicated geometry.

In recent work, we developed simplex-based parameters for systems that allow simplicial construction trees. For a multi-body system, a simplicial construction tree is a tree of simplices (nodes), with edges between simplices sharing common sub-simplices, where the shapes and orientations of the simplices completely determine inter-body configurations of the system. Each such simplicial construction tree naturally defines a set of system parameters based on simplices. Planar polygonal loops allow triangle construction trees [12]; the corresponding triangle-based parameters can be used to formulate the loop closure constraints as *linear inequalities*. Consequently, *CSpace* has a natural stratified structure for which the strata are convex in the triangle-based parameters. Knowledge of this explicit, simple structure is very advantageous in the study of configuration problems such as motion planning. We have proven [11, 12] that, given a generic planar polygonal loop and any one of its triangle construction trees, there are two easily computed special configurations such that, for *any* pair of connectible configurations, there is a connecting (though possibly not collision-free) path joining them, consisting of $k \leq 3$ sub-paths going through $k - 1$ of the special configurations, such that in triangle-based parameters each sub-path is a *straight line segment*. Prior to our work, the best known algorithms [18, 14, 25, 21] (all also ignoring the collision-free constraint), which used conventional parameters, produced paths that in general call for polynomial numbers of moves and require small-step numerical computation.

2 New Results

In this paper, we impose the additional constraint that configurations should be *free of self-intersections* (*i.e.*, collisions), with a special focus on convex configurations

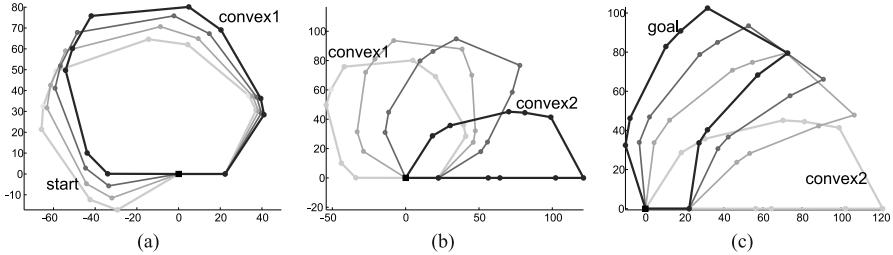


Fig. 1 A path in $CFree$ of an 11-sided polygon, connecting the two configurations labeled **start** and **goal** by going through two convex configurations labeled **convex1** and **convex2**. In each sub-figure, the beginning and end configurations are drawn in light and dark gray, and two intermediate configurations on the path are also shown; the end configuration in one sub-figure is the beginning configuration of the following sub-figure. The path segment between **start** and **convex1** is shown in (a), that between **convex1** and **convex2** in (b), and that between **convex2** and **goal** in (c).

of planar polygonal loops. A configuration of a planar polygonal loop is convex if there is no intersection among links and the enclosed polygonal area is convex. There are two opposite orientations among convex configurations, one with loop vertices in counterclockwise order and the other in clockwise order; we will refer to these as positive and negative orientations.

Polygonal loops are interesting both as classical geometrical objects and in modeling practical engineering systems, *e.g.*, grasping systems with fingers of planar revolute joints. Convex configurations of polygonal loops are important for motion planning. For example, the current approach for generating a path of collision-free configurations joining two given collision-free configurations [8] is to first move each of the non-convex given configurations into a convex configuration [6, 24, 3] and then to connect those two convex configurations [1]. This makes connecting two convex configurations an integral part of collision-free path planning for polygonal loops. Fig. 1 shows an example path constructed via this paradigm, but with computation done in our triangle-based parameters (specifically, the squared diagonal lengths, to be described in detail later).

In addition, it seems that many naturally occurring manipulation poses for human and robotic hands are convex or close to convex, so that results on convex polygonal loops are directly applicable to manipulation planning problems involving two convex poses. Fig. 2 shows example paths for a simulated manipulation system of two planar fingers grasping a block: the right finger has three revolute joints and the left one has two. This figure shows paths between two pairs of convex configurations, with two paths for each pair drawn in sub-figures in a row. Each sub-figure shows 4 sample configurations on a straight-line path (in squared diagonal lengths) between the two labeled configurations. For the first pair of convex configurations **start1** and **goal1**, sub-figures (a) and (b) show the path going through a special, easily computable triangular configuration **LW** (named for Lenhart and Whitesides, who first used these triangles in path planning; see [18]), and (c) shows the direct path

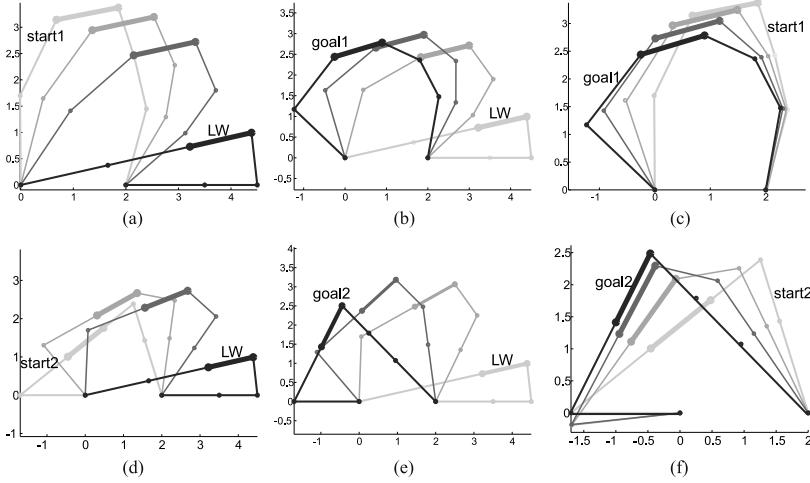


Fig. 2 Example paths for a simulated manipulation system of two planar fingers grasping a block

between the two labeled convex ones. For the second pair of convex configurations $\text{start}2$ and $\text{goal}2$, (d) and (e) show the path via LW , and (f) the direct path. Paths in sub-figures (a)–(e) stay convex; the path in (f) does not, but it stays collision-free.

These figures demonstrate (i) **one main result** in this paper: using squared diagonal lengths, any two identically oriented convex configurations of a polygonal loop can be connected by a path of two straight-line segments passing through the triangular configuration LW and consisted entirely of convex (in particular, collision-free) configurations; and (ii) **a conjecture**: the straight-line path between any two identically oriented convex configurations of a polygonal loop stays collision-free, but not necessarily convex.

These motion planning results are based on the **geometry of the set of convex configurations**. First, we note that link lengths, loop closure, convexity, and self-collision-free constraints on configurations all are intrinsic to the system and invariant under rigid motions. This indicates that all these constraints are effectively defined on the *deformation space* of the system [12], defined as the quotient space of the system $C\text{Space}$ modulo the group of rigid motions allowed by the system.

$$D\text{Space}(S) = C\text{Space}(S)/RigidMotion(S) \quad (1)$$

The idea of factoring out rigid motions in the study of intrinsic constraints and related configurations was used before, albeit in an intuitive fashion. For example, in papers [18, 25], it is assumed that each closed chain system under study has a link or two fixed in space, which effectively disallows any rigid motion, so in this case its deformation space is essentially identical to its configuration space. Paper [10] generates the configurations for such pinned linkages, then uses rigid motions to populate these configurations and create configurations of free-flying linkages. Configurations of pinned linkages are also deformations as defined here, and the procedure of

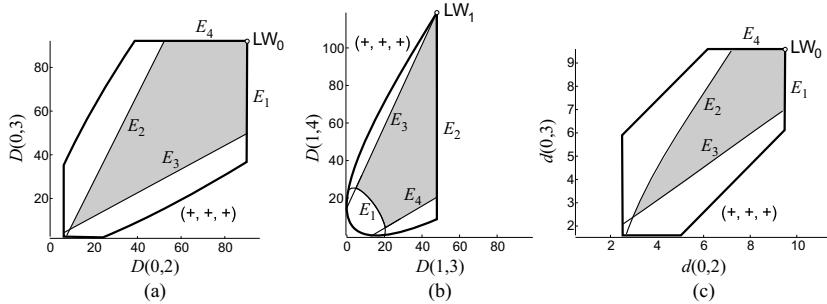


Fig. 3 The all-+ stratum of $D\text{Space}(\mathcal{P})$ for a pentagon with edge lengths $(6, 3.5, 3.4, 4, 5.6)$. The set $D\text{Convex}^+$ of convex deformations is the top-right, shaded subspace bounded by the set of E_l -loci (representing the non-strictly-convex deformations in which vertex P_l is fully Extended, *i.e.*, straight). For this particular pentagon, vertex P_0 cannot be fully extended, but all other vertices can. In (a), the anchor is P_0 and the shape parameters are squared diagonal; all boundary loci of $D\text{Convex}^+$ are straight line segments, and $D\text{Convex}^+$ is convex. In (b), the anchor is P_1 and the shape parameters are again squared diagonal. Here the boundary locus E_1 of $D\text{Convex}^+$, consisting of deformations fully extended at the anchor, is curved; the other boundary loci of $D\text{Convex}^+$ are straight; and $D\text{Convex}^+$ is not convex. In (c), the anchor is P_0 again, but the shape parameters are the diagonal lengths themselves (not their squares); here E_2 and E_3 are curved.

populating these deformations in space essentially corresponds to using the product of deformations and rigid motions to produce configurations. As another example, the terms “deformation” and “deformations space” were used in the grasping study of deformable objects [9]; while there deformation space is defined as the configuration space of mesh nodes, the potential energy-based study is really carried out in $D\text{Space}$ in sense defined here. This is because the potential energy of a deformable mesh depends on the deformational displacement from the initial (reference) configuration to a general configuration, and such displacements are invariant under rigid motions.

So far we have used configurations of pinned polygonal loops to describe our results in this paper (including in the abstract), since configurations are a well-known concept and the pinned loops effectively exclude rigid motions. But to reflect the applicability of our results to non-pinned loops and to formalize the intrinsic natures of the loop closure, convexity and non-self-intersection constraints, we describe our work in terms of deformations of general polygonal loops, and focus henceforth on $D\text{Space}$ and its subsets $D\text{Convex}$ and $D\text{Free}$ of convex and non-self-intersecting deformations, respectively. As an example, Fig. 3 shows parts of the deformation space of a single 5-bar loop in three different sets of triangle-based parameters (explained in detail in the figure caption). The part in each subfigure contains that half of $D\text{Convex}$ for the given loop in which the loop vertices form the counterclockwise order; this half, which we call $D\text{Convex}^+$, is shaded gray. The other half of $D\text{Convex}$, consisting of configurations with loop vertices in clockwise order, is the mirror image of $D\text{Convex}^+$ and has the identical structure.

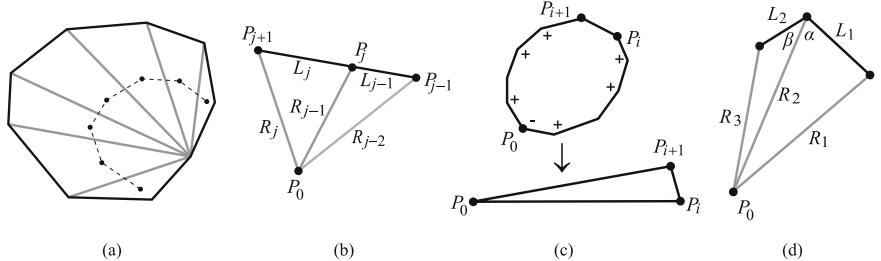


Fig. 4 (a) An anchored tree for a 9-link polygonal loop. (b) Part of a deformation with vertex P_j extended; the notations are used in the proof of Lemma 1. (c) Angle labeling from a generic convex deformation to the triangular deformation LW. (d) A general quadrilateral; used in the proof of Lemma 3, sketched in the appendix.

We prove that, in a set of triangle-based parameters for $D\text{Space}$ of a polygonal loop with fixed edge lengths, the set $D\text{Convex}^+$ is bounded by linear hyperplanes together with one (possibly empty) curved hypersurface (as in Figs. 3(a) and (b)). If (as in Fig. 3(a)) the curved hypersurface is empty, then $D\text{Convex}^+$ is actually convex, so any two convex deformations can be linearly connected through convex deformations. Even when the curved hypersurface is not empty, $D\text{Convex}^+$ is **star-shaped** with respect to the triangular deformation LW: that is, the straight-line path from any convex, positively oriented deformation to LW remains in $D\text{Convex}$. It follows that reconfiguration between any two identically oriented convex deformations of an n -bar polygonal loop can be done in at most two straight-line (in the parameter space) motions that remain in $D\text{Convex}$, as in Figs. 2(a) and (b) and Figs. 2(d) and (e). Prior to our work, the best algorithm [1] for reconfiguring convex polygons uses $O(n)$ moves, each designed to change four vertex angles at a time, with all configurations on the path staying convex.

3 The Set of Convex Polygonal Deformations

3.1 Triangle-Based Parameters for Polygonal Loops

Let \mathcal{P} be an n -sided planar polygonal loop, with vertices P_0, \dots, P_{n-1} (in cyclic order) and edge lengths $l_i = |P_{i+1} - P_i| > 0$. Throughout, index arithmetic is modulo n . Here we focus on the set $D\text{Convex}(\mathcal{P})$ of convex deformations of a given polygon \mathcal{P} with specified, fixed edge lengths.

We use a tree of anchored triangles (an example is shown in Fig. 4(a)) to parametrize $D\text{Convex}(\mathcal{P})$. Without loss of generality, we take vertex P_0 to be the anchor and use $(n-3)$ diagonals $[P_0, P_i]$, $i = 2, \dots, n-2$, together with links, to define the anchored triangles. An anchored triangle $[P_0, P_i, P_{i+1}]$ is considered to have orientation + (resp., -) if the three vertices in the specified order form a counterclockwise (resp., clockwise) cycle. If a triangle degenerates into a line segment, we can denote its orientation by another symbol (e.g., 0 as suggested in [12]); for

practical purposes related to the computation in this paper, we consider a degenerate triangle to have both + and – orientations.

These anchored triangles form a construction tree, since their shapes and orientations completely determine the deformations of the polygonal loop. Refer to our earlier paper [12] for general triangle construction trees for planar polygonal loops.

Clearly, in a convex deformation all anchored triangles have the same orientation, either + or –. The converse fails: a deformation with all orientations the same (e.g., start in Fig. 1(a)) need not be convex. It turns out that the set of all deformations with all anchored triangles having the + orientation forms a stratum of $D\text{Space}$, which we call the all+ stratum and denote by $D\text{Space}^+$. The all– stratum $D\text{Space}^-$ is defined similarly. It is obvious that half of $D\text{Convex}$ is contained in $D\text{Space}^+$ and half in $D\text{Space}^-$. Further, $D\text{Space}^+$ and $D\text{Space}^-$ are mirror images, so we need only consider the properties of the all+ stratum $D\text{Space}^+$ and its subset $D\text{Convex}^+ = D\text{Space}^+ \cap D\text{Convex}$.

3.2 Diagonal Length Parameters

For a planar polygonal loop and a construction tree of anchored triangles, an all+ deformation is completely described by its values of diagonal lengths, since the triangle orientations are already fixed and the diagonal lengths along with link lengths determine the shapes of the triangles. Therefore, diagonal lengths can serve as parameters for the $D\text{Space}^+$.

For our triangle-tree-based approaches, the polygonal loop and link length constraints are satisfied **if and only if** the triangles in the tree can be successfully formed with the given link lengths and appropriate values of the diagonal lengths. Since the constraints on three non-negative real numbers a, b, c to serve as the side lengths of a triangle are the so-called triangle inequality constraints $a + b \leq c$, $b + c \leq a$, $c + a \leq b$, the linear (first-power) diagonal lengths only need to satisfy triangle inequalities imposed by the triangles in the trees. As shown in our earlier papers [11, 12, 13], this explicit, linear inequality constraint formulation in triangle-based parameters provides substantial advantages over the conventional non-linear equality constraint formulation in Cartesian or joint angle parameters.

In the rest of this paper, we use squared diagonal lengths instead of linear diagonal lengths as parameters for $D\text{Space}^+$. As shown later, $D\text{Convex}^+$, when parametrized in squared lengths of anchored diagonals, is star-shaped and has simple boundary loci (such as in Figs. 3(a) and (b)), which are not the case in general for linear diagonal lengths (such as in Fig. 3(c)).

3.3 The Boundary of $D\text{Convex}^+$

Our study of $D\text{Convex}^+$ focuses on its boundary, where it makes contact with non-convex deformations in $D\text{Space}^+$. Intuitively it is easy to see that this boundary is exactly the set of convex, but not strictly convex, deformations, where as usual a convex deformation is called strictly convex if each of its interior angles is strictly less than π . In other words, a convex deformation is non-strictly-convex if at least

one vertex is fully extended, i.e., has angle π . Denote by E_j the (possibly empty) set of deformations with vertex P_j fully extended.

For any polygon \mathcal{P} , if E_j has a non-empty intersection with $D\text{Space}^+(\mathcal{P})$, then it partitions $D\text{Space}^+$ into two closed subsets (at most one of which can have empty interior); in one of these subsets, which we call the subspace for the convex vertex P_i , deformations have vertex angle $\angle P_{j-1}P_jP_{j+1}$ less than or equal to π , and in the other they have this angle greater than or equal to π . In $D\text{Space}^+(\mathcal{P})$ (resp., $D\text{Space}^-(\mathcal{P})$) the intersection (over all nonempty E_j) of these subspaces for convex vertices P_j is a closed subset; it is in fact exactly the set $D\text{Convex}^+(\mathcal{P})$ (resp., $D\text{Convex}^-(\mathcal{P})$) defined above. Furthermore, the sets E_j have the following nice property when parametrized by squared diagonal lengths.

Lemma 1. *For a planar n -bar polygonal loop with fixed edge lengths, each of the fully extended loci except that corresponding to the anchor lies in a linear hyperplane in squared lengths of anchored diagonals.*

Without loss of generality, in the proof below and for the remainder of the paper, let P_0 be the anchor, and assume all edge lengths are strictly positive. Denote the edge length $\|P_i - P_{i-1}\|$ ($i = 1, \dots, n$, $P_n = P_0$) by l_i and the anchored diagonal length $\|P_{i+1} - P_0\|$ ($i = 1, \dots, n-3$) by r_i respectively, and write $R_i = r_i^2$, $L_i = l_i^2$. We fix $0 < j \leq n-1$ and consider the locus E_j of all the deformations in $D\text{Space}^+(\mathcal{P})$ with vertex P_j fully extended.

Proof. First take the case where $j \in \{1, n-1\}$, i.e., the fully extended vertex P_j is a neighbor of P_0 . If $j = 1$, then r_1 is equal to $l_0 + l_1$, which is also its (possibly, non-tight) upper bound, and so E_1 is the intersection of $D\text{Space}^+(\mathcal{P})$ with the linear hyperplane (in the space with linear coordinates (R_1, \dots, R_{n-3})) defined by the equation $R_1 = (l_0 + l_1)^2$. If $j = n-1$, the situation is analogous, and E_{n-1} is defined by the equation $R_{n-3} = (l_{n-1} + l_n)^2$.

Now take the remaining cases, in which P_j is not a neighbor of P_0 (refer to Fig. 4(b)). Note that $\cos \angle P_0 P_j P_{j-1} = -\cos \angle P_0 P_j P_{j+1}$. Using the Law of Cosines to write down these cosine values, we get the following equation.

$$E_j : \frac{R_{j-1} + L_{j-1} - R_{j-2}}{2r_{j-1}l_{j-1}} = -\frac{R_{j-1} + L_j - R_j}{2r_{j-1}l_j} \quad (2)$$

all edge lengths are strictly positive, as are all anchored diagonal lengths for a non-self-intersecting (in particular, convex) deformation, so r_{j-1} can be factored out of both sides of equation (2), yielding an equation that is linear in R_{j-2} , R_{j-1} , and R_j . As before, this equation defines a linear hyperplane. \square

The set E_0 of deformations with the anchor joint P_0 fully extended is generally not a hyperplane in the anchored diagonal lengths, although it is of course a hypersurface (empty in case there are no valid system deformations with the anchor joint fully extended). In the proof above, we have only considered the conditions in the relevant squared diagonal lengths for the polygon to be fully extended at the vertex

in question. These diagonal lengths as well as other ones are also subject to triangle inequality constraints, which may make it impossible for a polygon to be fully extended at some vertices, leading to empty E_j for some P_j .

For example, the pentagon \mathcal{P} used in Fig. 3, with link lengths (6.0, 3.5, 3.4, 4.0, 5.6), can never be fully extended at P_0 because the sum of its two incident link lengths 5.6 and 6.0 strictly exceeds the sum of all other link lengths. Therefore $D\text{Convex}^+(\mathcal{P})$ parametrized in the squared diagonal lengths anchored at P_0 (Fig. 3(a)) has empty E_0 and is a polygon defined by straight-line loci labeled E_u , $j = 1, \dots, 4$, along which some other vertex is fully extended. But when using P_1 (or any other joint) as the anchor, and its corresponding squared anchored diagonal lengths as parameters, the locus E_1 is a non-empty curve (Fig. 3(b)), and $D\text{Convex}^+(\mathcal{P})$ is not convex. As a side note, boundary loci of $D\text{Convex}^+$ that are guaranteed to be hyperplanes in squared diagonal lengths can be curved in linear diagonal lengths (Fig. 3(c)).

Our main results on $D\text{Convex}^+(\mathcal{P})$ are summarized in the following theorem.

Theorem 1. (a) *Given a planar polygonal loop \mathcal{P} and an anchor vertex P_α for \mathcal{P} , the set of all convex deformations with edges in counterclockwise (resp., clockwise) order corresponds to the closed subset $D\text{Convex}^+(\mathcal{P})$ (resp., $D\text{Convex}^-(\mathcal{P})$) of the all-+ (resp., all--) stratum of $D\text{Space}(\mathcal{P})$ defined as the intersection of the subspace of the convex vertex P_j bounded by the non-empty fully-extended loci E_j . (b) The interior of $D\text{Convex}$ corresponds to strictly convex deformations. (c) In squared diagonal lengths, $D\text{Convex}$ is bounded by the linear hyperplanes E_j ($j \neq \alpha$) and the (non-linear, possibly empty) hypersurface E_α .*

Clearly, if E_0 is empty, $D\text{Convex}^+$ (and $D\text{Convex}^-$) is an $n - 3$ -dimensional convex polyhedron and any two identically oriented convex deformations therein can be linearly connected by a path of convex deformations. Thus, for a given polygon and convex reconfiguration problem, it would be desirable to use a non-fully-extendible vertex as the anchor, should one exist. It is easy to find such a vertex or determine none exists based on the following theorem.

Theorem 2. (a) *For a given polygonal loop \mathcal{P} , one vertex cannot be fully extended if and only if the sum of the lengths of its two incident edges is strictly greater than the sum of the rest of the edges.* (b) *If such a non-fully-extendible vertex exists, its two adjacent edges are among the top three longest edges in \mathcal{P} and satisfy the condition in (a).*

Proof Sketch. When vertex P_j is fully extended, its two incident edges can be considered as one (virtual) edge with length $l_{j-1} + l_j$. Property (a) follows, based on the well-known property that no edge length of a polygon can be strictly greater than the sum of all other edge lengths. Property (b) is also simple to prove, for example using proof by contradiction. \square

4 Reconfiguration of Convex Polygonal Deformations

Now consider the non-self-intersection path planning problem for convex deformations. As explained in the previous section, if a given n -bar polygonal loop \mathcal{P} has one non-fully-extendible joint, that joint should be used as an anchor; then, in the corresponding squared diagonal lengths, each component of the set $D\text{Convex}(\mathcal{P})$ is an $(n - 3)$ -dimensional convex polyhedron. In this case, any two convex deformations with the same global orientation (counterclockwise or clockwise) can be linearly connected by a path of convex, and thus self-intersection-free, deformations.

If a given polygonal loop has no non-extendible joint, its $D\text{Convex}$ in squared diagonal lengths is generally not convex. But even in this case, we prove that we can still linearly connect any convex deformation to the canonical LW triangle, with the whole path staying convex. For anchor P_0 , its LW triangle is defined by vertices P_0 , P_i and P_{i+1} , with P_i satisfying the following properties: $\sum_{j=0}^{i-1} l_j \leq L/2$ and $\sum_{j=0}^i l_j > L/2$, where L is the sum of all edge lengths. It is easy to see that the LW triangle achieves simultaneous maxima for all the anchored diagonal lengths; its positively (resp. negatively) oriented version corresponds to the "top right corner" of $D\text{Space}^+$ (resp. $D\text{Space}^-$), which we call the LW vertex and denote, as before, by LW . The main theorem of this section is stated as follows.

Theorem 3. *Given a polygonal loop \mathcal{P} with fixed edge lengths and an (arbitrary) anchor vertex, the straight-line path (in squared diagonal lengths) between any positively (resp. negatively) oriented convex deformation and LW lies in $D\text{Convex}^+(\mathcal{P})$ (resp. $D\text{Convex}^-(\mathcal{P})$). In other words, $D\text{Convex}^+(\mathcal{P})$ (resp. $D\text{Convex}^-(\mathcal{P})$) is star-shaped with respect to positively (resp. negatively) oriented LW .*

Proof Sketch. As stated in Theorem 1, $D\text{Convex}^+(\mathcal{P})$ is bounded by hyperplanes E_j ($j \neq 0$) and the hypersurface E_0 . Ignoring E_0 , all other hyperplane E_j boundaries of $D\text{Convex}^+(\mathcal{P})$ define a (possibly non-bounded) convex polytope Q . A straight-line path between any two points in Q stays in Q , never crossing any E_j with $j \neq 0$. Thus the straight-line path between any convex deformation and LW stays in Q , and at each point in the path the deformation is strictly convex at all non-anchor vertices. Thus the path can leave $D\text{Convex}^+(\mathcal{P})$ only if it crosses E_0 and becomes non-convex at the anchor P_0 . We prove that this never happens by showing that on the linear segment between any convex deformation and LW , the anchor angle θ_0 changes monotonically. Since θ_0 is less than or equal to π at both endpoints, it is less than or equal to π on the entire path. Therefore all angles of all deformations on the path are less than or equal to π , so all deformations on the path are convex. \square

In the rest of the section we sketch our proof of this monotonicity property. We use the Cauchy-Steinitz Lemma [4, 23] and the labeling of polygon angles. Given two convex deformations D_0 and D_1 of \mathcal{P} , with D_0 considered to be the source and D_1 the destination, label an angle of D_0 by + (resp., $-$; 0) if it is smaller than (resp., larger than; equal to) the corresponding angle of D_1 . Our statement of the Cauchy-Steinitz Lemma follows [1], to which the reader is referred for a proof sketch.

Lemma 2. (Cauchy-Steinitz Lemma) *If D_0 and D_1 are distinct, then the labeling defined above has at least four sign alternations.*

Now let our given convex deformation be D_0 and let D_1 be LW . Let the vertices of LW be P_0, P_i and P_{i+1} . In LW the two sub-chains of vertices $(P_0, P_1, \dots, P_{i-1}, P_i)$ and $(P_{i+1}, P_{i+2}, \dots, P_{n-1}, P_0)$ are straightened out completely, and the interior angles of these two subchains—namely $\theta_1, \dots, \theta_{i-1}$ and $\theta_{i+2}, \dots, \theta_{n-1}$ —all equal π , their maximum values on convex deformations. Therefore the labeling for all these angles in D_0 is + or 0. Also, at least one subchain has to have at least one vertex labeled +, for if not, D_0 would equal LW . Now, it is possible that one of the two subchains has all labels 0, so that subchain is already full extended at D_0 . Both in this special case and the generic case in which both subchains have at least one vertex labeled +, the anchor angle must be labeled – labeling, as must at least one angle at the other two vertices of LW . (Fig. 4(c) illustrates such a labeling.)

It is important to note that the labeling property given in the Cauchy-Steinitz Lemma is about two convex configurations. But the labeling for the subchain interior angles and the anchor angle $\theta(0)$ is maintained **on the whole straight-line path** (in squared diagonal lengths) between the specified deformation and LW , as summarized in Lemmas 3 and proved in the appendix. Only the angles at the other two LW vertices, namely θ_i and θ_{i+1} , may exhibit non-monotonicity in the path.

Lemma 3. *In a straight-line path (in squared diagonal lengths) directed from a convex deformation to LW , (a) of the interior angles of the two subchains, all those labeled with + increases monotonically, and all those labeled with 0 stay constant (at π); and (b) the anchor angle decreases monotonically.*

Proof Sketch. Here we consider the generic case, where each subchain has at least one positively labeled interior angle and the initial anchor angle $\theta_0(0)$ is less than π ; for lack of space, we do not present our proofs in the special case. Our basic idea is to prove that properties (a) and (b) hold for the motion at $t = 0$ and make the next system deformation at $t = \varepsilon$ continue to be convex, which can serve as the new start convex deformation and continue the reconfiguration process.

First we prove that (b) is true if (a) is true. Note that D_0 is convex and $\theta_0(0) < \pi$ in the generic case. Assume θ_0 increases at time 0 and all the subchain interior angles behave well as stated in (a). Consider the initial convex deformation D_0 and its immediately successor deformation $D(\varepsilon)$ as the source and destination deformations. Both subchains as well as the anchor angle have + labels. So there cannot be 4 sign alternations, and $D(\varepsilon)$ has to be non-convex. And based on some of the reasoning in the proof sketch for Theorem 3, vertex P_0 becomes non-convex and $\theta_0(\varepsilon) > \pi$, which requires discontinuity for the anchor angle θ_0 . But the motion is continuous, and thus so are all the angles including the anchor angle. Therefore it is impossible for θ_0 to jump from a strictly convex angle to a strictly non-convex angle. Therefore, (b) is true if (a) is true. And (b) is what is needed for proving Theorem 3.

As for the proof of (a), it is straightforward to use the Law of Cosines to verify the stay-put (at π) case. As for the monotonically increasing case, we compute the derivative of the cosine of the vertex angle and prove the derivative to be negative. The proof sketch is given in the appendix. \square

An immediate result from Theorem 3 is the following.

Corollary 1. *Any two identically oriented convex deformations of an n -bar polygonal loop with fixed edge lengths can be connected by two straight-line moves in squared lengths of anchor diagonals, via LW , with the whole path staying convex.*

5 A Conjecture on $DConvex$

While $DConvex^+$ (and its mirror image, $DConvex^-$) has the nice star-shaped geometry when parametrized by squared diagonal lengths, for some pairs of deformations (e.g., those in Fig. 2) our two-segment straight-line paths via LW may seem to take a long detour. We have been exploring the possibility of directly connecting convex deformation pairs. As illustrated in Figs. 2(c) and (f), some direct paths may still stay convex, while some others may stay collision-free though not convex at all times. One might also suspect that some direct paths may have pass through deformations with self-intersections. We report here that our extensive numeric experiments have **not** found any direct paths between triangular deformations of convex polygonal loops that leave $DFree$. We focus on triangular deformations since they are the corners of $DConvex$ and direct paths between some of them do leave $DConvex$ and thus might also leave $DFree$. For *hundreds* of polygonal loops with 7 to 12 links, direct paths between triangular deformation pairs were *all* found to be self-intersection-free in our simulation study. This prompts us to propose the following conjecture: the convex hull of $DConvex^+$, as parametrized by squared lengths of anchored diagonals, is a subset of the collision-free deformation subspace $DFree^+$.

$$\text{ConvexHull}(DConvex^+(\mathcal{P})) \subset DFree^+(\mathcal{P}) \quad (3)$$

Part of our ongoing research is to prove or disprove this conjecture. In this process, we have found the following nice property.

Lemma 4. *Given a polygonal loop \mathcal{P} with an arbitrary number of links and any two triangular deformations of \mathcal{P} , at most 7 vertices—namely, the anchor joint and the joints that serve as the vertices of the two triangles—can change their joint angles on the direct straight-line path (in squared diagonal lengths) between the two triangles; all other joints stay fully extended all along the path.*

This is based on the stay-at- π case for Lemma 3(a) (proved in the appendix). Fig. 5 shows an example path for an 11-bar loop with all 7 vertices marked. In effect, the lemma indicates that any direct path between two triangular deformations of an n -bar polygonal loop can be viewed effectively as a path for a loop only involving those up-to-7 vertices with changing joint angles. Therefore validation of the conjecture can focus on polygonal loops with up to 7 links. Even with this insight and our results on the collision-free space that we have no space to describe here, a theoretical proof or counter-example for the conjecture has been elusive.

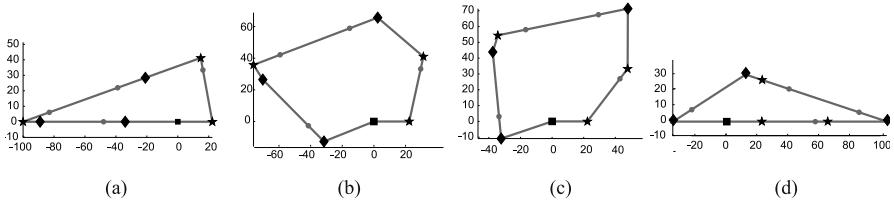


Fig. 5 Example path for an 11-bar polygonal loop: (a) and (d) are its two triangle deformations, (b) and (c) are two deformation on the straight line path (in squared diagonal lengths) between these two triangles. As explained in section 5, the only joints that can change their joint angles on such a path are the anchor joint and the joints that serve as vertices of the triangles, for a total of at most 7, as marked in the figure. All other joints remain fully extended throughout the path.

6 Summary

In this paper we proved that, in squared diagonal lengths of an anchored triangle tree, each of the two components of the set of convex deformations of a planar polygonal loop with fixed edge lengths is bounded by linear hyperplanes together with one (possibly empty) curved hypersurface. In case the curved hypersurface does not appear, each component (which is the set of all convex deformations of a fixed orientation) actually is a convex polyhedron, so that any two identically oriented convex deformations can be directly connected by a path of convex deformations. Even if the curved hypersurface does appear, each component is star-shaped with respect to the LW triangle for the anchor. In any case, therefore, path planning between any two identically oriented convex deformations can be carried out **within the set of convex deformations** by using at most two straight-line (in squared diagonal lengths) motions. We also presented our conjecture that the straight-line path between any two identically oriented convex deformations stays self-intersection-free, along with our motivations and progress. The results presented here provide substantial improvements in efficiency and simplicity over prior work and make it very easy to plan self-intersection-free paths for convex deformations of polygonal loops. They also reemphasize a general point made in our previous work [11, 12, 13]: for multibody systems, a good choice of coordinates can greatly simplify many important problems. (A nice example of a different sort [2] was presented at WAFR2012.)

In our ongoing research, in addition to working on validating our conjecture, we are developing geometric descriptions and motion planning algorithms for general non-self-intersection deformations.

7 Appendix – Proof Sketch for Lemma 3(a)

To prove Lemma 3.(a) for each subchain interior angle, it suffices to look at one angle at a time. For each vertex angle θ_j , there is a quadrilateral defined by the

vertices $(P_0, P_{j-1}, P_j, P_{j+1})$, similar to that in Fig. 4(b) and with vertex P_j having a convex ($\leq \pi$) interior angle. For simplicity, we use the notations in Fig. 4(d) to derive the formula for the cosine of the vertex angle $\cos(\alpha(t) + \beta(t))$ and prove that its derivative $d(\cos(\alpha + \beta))/dt$ evaluated at $t = 0$ has value less than or equal to 0, so that the vertex angle $\alpha(t) + \beta(t)$ is not decreasing at $t = 0$. For reasons of space, only a proof sketch for the generic case can be included in this appendix.

We will use $x(t)$ to indicate variable x at $t \in [0, 1]$ (sometimes we suppress (t) to save some space, where doing so seems unlikely to cause confusion). For length entities, we use lower (resp., upper) case letters to denote lengths (resp., squared lengths); e.g., $R_i = r_i^2$, and $L_i = l_i^2$. Without loss of generality, we assume that all link and diagonal lengths are positive.

For the general case, the initial quadrilateral is convex; and we let $\alpha(0) \in (0, \pi)$, $\beta(0) \in (0, \pi)$ and $\alpha(0) + \beta(0) \in (0, \pi]$. At the end $t = 1$, the polygonal loop reaches its LW triangular deformation and our quadrilateral degenerates into a line segment with $r_2(1) = r_1(1) + l_1 = r_3(1) - l_2$, $\alpha(1) = 0$, $\beta(1) = \pi$. For the straight-line path under consideration, the intermediate squared diagonal lengths are linear interpolations of their end values: $R_i(t) = (1-t)R_i(0) + tR_i(1)$.

[Stay-at- π case]. Assume initially $\alpha(0) + \beta(0) = \pi$. Then at both end points of the time segment $[0, 1]$, we have $\alpha + \beta = \pi$. We want to show that this property is also true for the interior points. By repeatedly using the law of cosines and the linear interpolation of $R_i(t)$, we get

$$\begin{aligned}\cos \alpha(t) &= \frac{(1-t)r_2(0)\cos \alpha(0) + t \cdot r_2(1)\cos \alpha(1)}{r_2(t)} \\ \cos \beta(t) &= \frac{(1-t)r_2(0)\cos \beta(0) + t \cdot r_2(1)\cos \beta(1)}{r_2(t)}\end{aligned}\quad (4)$$

Since $\cos \alpha(0) = -\cos \beta(0)$ and $\cos \alpha(1) = -\cos \beta(1)$, from the computation above we also have $\cos \alpha(t) = -\cos \beta(t)$. Since $\alpha(t) + \beta(t)$ is continuous on the path and has an initial value of π , the property above indicates that the sum stays at π throughout the whole path. \square

[General case: angle increases monotonically]. From trigonometry, $\cos(\alpha + \beta) = \cos \alpha \cos \beta - \sin \alpha \sin \beta$. The cosine terms can be obtained from equation (4), along with substitution $\alpha(1) = 0$ and $\beta(1) = \pi$. To get the sine terms, we compare two calculations of the area Δ of the triangle sides $r_1(t), l_1, r_2(t)$, one using $\sin \alpha(t)$, the other using Heron's formula, as follows.

$$\begin{aligned}\Delta(r_1(t), l_1, r_2(t)) &= r_2(t)l_1 \sin \alpha(t)/2 \\ \Delta(r_1(t), l_1, r_2(t)) &= \sqrt{2(R_2R_1 + R_2L_1 + R_1L_1) - (R_2^2 + R_1^2 + L_1^2)/4}\end{aligned}$$

After much algebraic and trigonometric computation, we get

$$\sin \alpha(t) = \frac{\sqrt{(1-t)A(t)}}{r_2(t)}, \sin \beta(t) = \frac{\sqrt{(1-t)B(t)}}{r_2(t)}, \text{ where} \quad (5)$$

$$A(t) = (1-t)R_1(0)\sin^2 \gamma_1(0) + t(R_1(0) + R_1(1) + 2r_1(1)r_1(0)\cos \gamma_1(0)),$$

$$B(t) = (1-t)R_3(0)\sin^2 \gamma_2(0) + t(R_3(0) + R_3(1) - 2r_3(1)r_3(0)\cos \gamma_2(0))$$

Combining all the terms in (4) and (5), we get

$$\cos(\alpha + \beta) = \frac{N_1(t) - N_2(t)}{D(t)}, \text{ where} \quad (6)$$

$$N_1(t) = 1-t)^2 R_2(0) \cos \alpha(0) \cos \beta(0) - t^2 R_2(1) +$$

$$t(1-t)r_2(0)r_2(1)(\cos \beta(0) - \cos \alpha(0))$$

$$N_2(t) = (1-t)\sqrt{A(t)}\sqrt{B(t)}, \quad D(t) = R_2(t)$$

We further compute the derivative of $\cos(\alpha + \beta)$ to get

$$\begin{aligned} \frac{d(\cos(\alpha + \beta))}{dt} &= D^{-1}(t) \left(\frac{d(N_1(t) - N_2(t))}{dt} - \cos(\alpha + \beta)(t)(R_2(1) - R_2(0)) \right) \\ &:= D^{-1}(t)T(t) \end{aligned}$$

Since the denominator $D(t) = R_2(t)$ is always positive and we want to show that the derivative is non-positive at $t = 0$, we just need to show the numerator $T(t)$ to be non-positive at $t = 0$. After much computation and simplification based on trigonometric laws and properties of the initial quadrilateral being convex and the final quadrilateral being a line segment, we get

$$T(0) = r_2(0)r_2(1)k_1 - (R_2(1) + R_2(0))k_2, \quad \text{where} \quad (7)$$

$$k_1 = \frac{(\sin \beta(0) - \sin \alpha(0))\sin(\alpha(0) + \beta(0))}{\sin \alpha(0) \sin \beta(0)} \quad (8)$$

$$k_2 = \frac{\sin^2(\alpha(0) + \beta(0))}{2 \sin \alpha(0) \sin \beta(0)} \quad (9)$$

Clearly, for a generic convex quadrilateral with $\alpha(0) \in (0, \pi)$, $\beta(0) \in (0, \pi)$, and $\alpha(0) + \beta(0) \in (0, \pi]$, we have $k_2 \geq 0$. For the case of $k_2 = 0$, we have $\alpha(0) + \beta(0) = \pi$, whence both k_1 and $T(0)$ are zero. This gives us an alternative proof for the stay-at- π case. For the case of $k_2 > 0$, we can prove that the ratio k_1/k_2 stays in the interval $(-2, 2)$ and thus can define angle $\rho = \arccos((k_1/k_2)/2) \in (0, \pi)$. In other words, $k_1 = 2k_2 \cos \rho$. Therefore $T(0) = -k_2(R_2(1) + R_2(0) - 2r_2(0)r_2(1)\cos \rho)$ and is negative since the expression inside the parentheses has the form of the Law of Cosines and corresponds to the squared side length of a triangle with $r_2(0)$ and $r_2(1)$ as the other two sides and ρ as the angle between them. This concludes our proof for $T(0) \leq 0$ and the generic case of Lemma 3.(a). \square

References

1. Aichholzer, O., Demaine, E.D., Erickson, J., Hurtado, F., Overmars, M., Soss, M.A., Toussaint, G.T.: Reconfiguring convex polygons. In: Proc. 12th Annual Canadian Conf. on Computational Geometry (CCCG 2000), pp. 17–20 (2000)
2. Bretl, T., McCarthy, Z.: Equilibrium configurations of a Kirchhoff elastic rod under quasi-static manipulation. In: Workshop on Algorithmic Foundations of Robotics, WAFR 2012 (2012)
3. Cantarella, J., Demaine, E.D., Iben, H., O'Brien, J.: An energy-driven approach to linkage unfolding. In: Proc. 20th Annual ACM Symp. on Computational Geometry (SoCG 2004), pp. 134–143 (June 2004)
4. Cauchy, A.L.: Deuxième mémoire sur les polygons et polyhèdres. Journal de l'École Polytechnique (9), 87–98 (1813)
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. MIT Press, Cambridge (2005)
6. Connelly, R., Demaine, E.D., Rote, G.: Straightening polygonal arcs and convexifying polygonal cycles. In: Proc. 41st Annual Symp. on Foundations of Computer Science (FOCS 2000), pp. 432–442 (2000)
7. Craig, J.J.: Introduction to robotics: Mechanics and control, 2nd edn. Addison-Wesley Publishing Company, Reading (1989)
8. Demaine, E.D., O'Rourke, J.: Geometric folding algorithms: Linkages, origami, polyhedra. Cambridge University Press (July 2007)
9. Gopalakrishnan, K., Goldberg, K.: D-Space and Deform Closure Grasps of Deformable Parts. Int. J. Robot. Res. 24(11), 899–910 (2005)
10. Han, L., Amato, N.M.: A Kinematics-Based Probabilistic Roadmap Method for Closed Chain Systems. In: Donald, B.R., Lynch, K.M., Rus, D. (eds.) Algorithmic and Computational Robotics: New Directions (WAFR 2000), pp. 233–246 (2000)
11. Han, L., Rudolph, L., Blumenthal, J., Valodzin, I.: Stratified Deformation Space and Path Planning for a Planar Closed Chain with Revolute Joints. In: Akella, S., Amato, N.M., Huang, W.H., Mishra, B. (eds.) Algorithmic Foundation of Robotics VII. STAR, vol. 47, pp. 235–250. Springer, Heidelberg (2008)
12. Han, L., Rudolph, L., Blumenthal, J., Valodzin, I.: Convexly stratified deformation space and efficient path planning for a planar closed chain with revolute joints. Int. J. Robot. Res. 27, 1189–1212 (2008)
13. Han, L., Rudolph, L., Dorsey-Gordon, S., Glotzer, D., Menard, D., Moran, J., Wilson, J.R.: Bending and kissing: Computing self-contact configurations of planar loops with revolute joints. In: ICRA (2009)
14. Kapovich, M., Millson, J.: On the moduli spaces of polygons in the euclidean plane. J. Diff. Geom. 42, 133–164 (1995)
15. Latombe, J.C.: Robot motion planning. Kluwer Academic Publishers, Boston (1991)
16. LaValle, S.M.: Planning algorithms. Cambridge University Press, Cambridge (2006), <http://planning.cs.uiuc.edu/>
17. Lozano-Pérez, T.: Spatial Planning: A Configuration Space Approach. IEEE Trans. Computing 2, 108–120 (1983)
18. Lenhart, W.J., Whitesides, S.H.: Reconfiguring closed polygon chains in euclidean d-space. Disc. Comput. Geom. 13, 123–140 (1995)
19. Mason, M.: Mechanics of robotic manipulation. The MIT Press (2001)
20. Merlet, J.-P.: Parallel robots. Springer, New York (2000)

21. Milgram, R.J., Trinkle, J.C.: The geometry of configuration spaces for closed chains in two and three dimensions. In: Homology Homotopy and Applications (2002)
22. Murray, R.M., Li, Z., Sastry, S.S.: A mathematical introduction to robotic manipulation. CRC Press, Boca Raton (1994)
23. Steinitz, E., Rademacher, H.: Vorlesungen über die Theorie der Polyeder. Springer, Berlin (1934) (reprinted 1976)
24. Streinu, I.: A combinatorial approach to planar non-colliding robot arm motion planning. In: Proc. IEEE Symp. Foundations of Computer Science (FOCS), pp. 443–453 (2000)
25. Trinkle, J.C., Milgram, R.J.: Complete path planning for closed kinematic chains with spherical joints. Int. J. Robot. Res. 21(9), 773–789 (2002)

Equilibrium Configurations of a Kirchhoff Elastic Rod under Quasi-static Manipulation

Timothy Bretl and Zoe McCarthy

Abstract. Consider a thin, flexible wire of fixed length that is held at each end by a robotic gripper. The curve traced by this wire can be described as a local solution to a geometric optimal control problem, with boundary conditions that vary with the position and orientation of each gripper. The set of all local solutions to this problem is the configuration space of the wire under quasi-static manipulation. We will show that this configuration space is a smooth manifold of finite dimension that can be parameterized by a single chart. Working in this chart—rather than in the space of boundary conditions—makes the problem of manipulation planning very easy to solve. Examples in simulation illustrate our approach.

1 Introduction

Figure 1 shows a thin, flexible wire of fixed length that is held at each end by a robotic gripper. Our basic problem of interest is to find a path of each gripper that causes the wire to move between start and goal configurations while remaining in static equilibrium and avoiding self-collision. As will become clear, it is useful to think about this problem equivalently as finding a path of *the wire* through its set of equilibrium configurations (i.e., the set of all configurations that would be in equilibrium if both ends of the wire were held fixed).

There are two reasons why this problem seems hard to solve. First, the configuration space of the wire has infinite dimension. Elements of this space are framed

Timothy Bretl
Department of Aerospace Engineering,
University of Illinois at Urbana-Champaign, Urbana, IL, 61801
e-mail: tbretl@illinois.edu

Zoe McCarthy
Department of Electrical and Computer Engineering,
University of Illinois at Urbana-Champaign, Urbana, IL, 61801
e-mail: zoemccarthy12@gmail.com

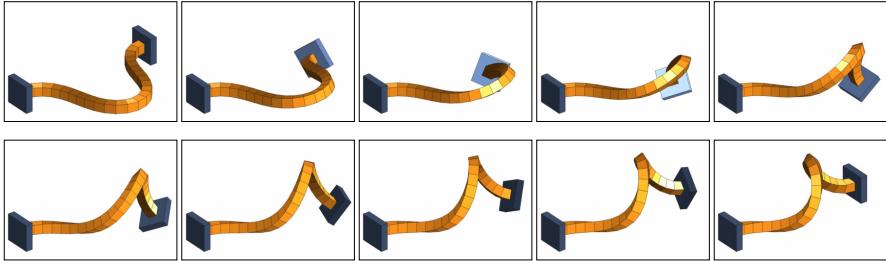


Fig. 1 Quasi-static manipulation of an elastic rod (orange) by robotic grippers (blue). Notice that the grippers begin and end in the same position and orientation. Remarkably, this motion corresponds to a straight-line path in the global coordinate chart we derive in this paper.

curves, i.e., continuous maps $q: [0, 1] \rightarrow SE(3)$, the shape of which in general must be approximated. Second, a countable number of configurations may be in static equilibrium for given placements of each gripper, none of which (typically) can be computed in closed form. For these two reasons, the literature on manipulation planning suggests exploring the set of equilibrium configurations indirectly, by sampling displacements of each gripper and using numerical simulation to approximate their effect on the wire. This approach was developed in the seminal work of Lamiriaux and Kavraki [21] and was applied by Moll and Kavraki [29] to manipulation of elastic “deformable linear objects” like the flexible wire we consider here.

Our contribution in this paper is to show that the set of equilibrium configurations for the wire is a smooth manifold of finite dimension that can be parameterized by a single (global) coordinate chart. We model the wire as a Kirchhoff elastic rod [8]. The framed curve traced by this elastic rod in static equilibrium can be described as a local solution to a geometric optimal control problem, with boundary conditions that vary with the position and orientation of each gripper [41, 8]. Coordinates for the set of *all* local solutions over *all* boundary conditions are provided by the initial value of costates that arise in necessary and sufficient conditions for optimality. These coordinates describe all possible configurations of the elastic rod that can be achieved by quasi-static manipulation, and make manipulation planning—the seemingly “hard problem” described above—very easy to solve.

Our approach builds on a long history in analysis of elasticity [3]. Recent work gives a more or less complete picture of planar elastic rods [32, 31], and this work rests on similar foundations as our own [1]. We have also been influenced by analysis of conjugate points in elastic filament models of DNA [14] and by an earlier sequence of papers initiated by Langer and Singer [22]. In addition, we note the emergence of new approaches to dynamic simulation of elastic rods based on discrete geometry [7], which has started to find application in robotics [18]. However, none of this previous work answers our questions about the set of equilibrium configurations: is it a finite-dimensional manifold, what are its coordinate charts, etc. These questions are the foundation of our approach to manipulation planning.

We are motivated by applications that require manipulation of deformable objects: knots and suturing [15, 38, 40, 33, 5], cable routing [16], folding clothes [6, 43], compliant parts handling [26, 13] and assembly [4], surgical retraction of tissue [17], and protein folding [2]. Related applications include haptic exploration with “whisker” sensors, often modeled as elastic rods [36, 12]. We are also motivated by the link, pointed out by Tanner [39], between manipulation of deformable objects and control of hyper-redundant [10] and continuum [30, 42] robots.

Section 2 establishes our theoretical framework. The two key parts of this framework are optimal control on manifolds and Lie-Poisson reduction. We derive coordinate formulae for necessary and sufficient conditions—in the former case these formulae are well known, but in the latter case they are not. Section 3 shows how our framework applies to the elastic rod. We prove that the set of equilibrium configurations for this rod is a smooth manifold of finite dimension that can be parameterized by a single chart, and we explain why this result makes the problem of manipulation planning easy to solve. We note in particular that the computations required for planning are trivial to implement—the example of Figure 1 was generated by about a dozen lines of code. Section 4 identifies several research directions that are enabled by our analysis of the elastic rod. Our ideas follow from but significantly extend earlier work on a simpler model (a planar elastic kinematic chain [28]).

2 Theoretical Framework

We will see in Section 3 that the framed curve traced by an elastic rod in equilibrium is a local solution to a geometric optimal control problem. Here, we provide the framework to characterize this solution. Section 2.1 gives our notation for smooth manifolds. It is not a review (for this, see [25]), and is included only because notation varies widely in the literature. Section 2.2 states necessary and sufficient conditions for optimality on manifolds in a form that is useful for us. Section 2.3 derives coordinate formulae to compute these necessary and sufficient conditions. Most of these results are a translation of [1] in a style more consistent with [25, 27]. We conclude with coordinate formulae to test sufficiency for left-invariant systems on Lie groups (Theorem 4), an important result that is not in [1] and is hard to find elsewhere.

2.1 Smooth Manifolds

Let M be a smooth manifold. The space of smooth real-valued functions on M is $C^\infty(M)$. The space of smooth vector fields on M is $\mathfrak{X}(M)$. The action of $v \in T_m M$ on $f \in C^\infty(M)$ is $v \cdot f$. The action of $w \in T_m^* M$ on $v \in T_m M$ is $\langle w, v \rangle$. The action of $X \in \mathfrak{X}(M)$ on $f \in C^\infty(M)$ produces the function $X[f] \in C^\infty(M)$ satisfying $X[f](m) = X(m) \cdot f$ for all $m \in M$. The Jacobi-Lie bracket of $X, Y \in \mathfrak{X}(M)$ is the vector field $[X, Y] \in \mathfrak{X}(M)$ satisfying $[X, Y][f] = X[Y[f]] - Y[X[f]]$ for all $f \in C^\infty(M)$. If $F : M \rightarrow N$ is a smooth map between manifolds M and N , then the pushforward of F at $m \in M$ is the linear map $T_m F : T_m M \rightarrow T_{F(m)} N$ satisfying $T_m F(v) \cdot f = v \cdot (f \circ F)$ for all $v \in T_m M$ and $f \in C^\infty(N)$. The pullback of F at $m \in M$ is the dual map

$T_m^*F: T_{F(m)}^*N \rightarrow T_m^*M$ satisfying $\langle T_m^*F(w), v \rangle = \langle w, T_mF(v) \rangle$ for all $v \in T_mM$ and $w \in T_{F(m)}^*N$. We say F is degenerate at $m \in M$ if there exists non-zero $v \in T_mM$ such that $T_mF(v) = 0$. It is equivalent that the Jacobian matrix of any coordinate representation of F at m have zero determinant. The Poisson bracket generated by the canonical symplectic form on T^*M is $\{\cdot, \cdot\}: C^\infty(T^*M) \times C^\infty(T^*M) \rightarrow C^\infty(T^*M)$. The cotangent bundle T^*M together with the bracket $\{\cdot, \cdot\}$ is a Poisson manifold. The Hamiltonian vector field of $H \in C^\infty(T^*M)$ is the unique vector field $X_H \in \mathfrak{X}(T^*M)$ satisfying $X_H[K] = \{K, H\}$ for all $K \in C^\infty(T^*M)$. We use this same notation when H is time-varying. Finally, let $\pi: T^*M \rightarrow M$ satisfy $\pi(m, w) = m$ for all $w \in T_m^*M$.

2.2 Optimal Control on Manifolds

Let $U \subset \mathbb{R}^m$ for some $m > 0$. Assume $g: M \times U \rightarrow \mathbb{R}$ and $f: M \times U \rightarrow TM$ are smooth maps. Consider the optimal control problem

$$\begin{aligned} \text{minimize}_{q,u} \quad & \int_0^1 g(q(t), u(t)) dt \\ \text{subject to} \quad & \dot{q}(t) = f(q(t), u(t)) \text{ for all } t \in [0, 1] \\ & q(0) = q_0, \quad q(1) = q_1, \end{aligned} \tag{1}$$

where $q_0, q_1 \in M$ and $(q, u): [0, 1] \rightarrow M \times U$. Define the parameterized Hamiltonian $\hat{H}: T^*M \times \mathbb{R} \times U \rightarrow \mathbb{R}$ by $\hat{H}(p, q, k, u) = \langle p, f(q, u) \rangle - kg(q, u)$, where $p \in T_q^*M$.

Theorem 1 (Necessary Conditions). Suppose $(q_{opt}, u_{opt}): [0, 1] \rightarrow M \times U$ is a local optimum of (1). Then, there exists $k \geq 0$ and an integral curve $(p, q): [0, 1] \rightarrow T^*M$ of the time-varying Hamiltonian vector field X_H , where $H: T^*M \times \mathbb{R} \rightarrow \mathbb{R}$ is given by $H(p, q, t) = \hat{H}(p, q, k, u_{opt}(t))$, that satisfies $q(t) = q_{opt}(t)$ and

$$H(p(t), q(t), t) = \max_{u \in U} \hat{H}(p(t), q(t), k, u) \tag{2}$$

for all $t \in [0, 1]$. Furthermore, if $k = 0$, then $p(t) \neq 0$ for all $t \in [0, 1]$.

Proof. See Theorem 12.10 of [1]. □

We call the integral curve (p, q) in Theorem 1 an *abnormal extremal* when $k = 0$ and a *normal extremal* otherwise. As usual, when $k \neq 0$ we may simply assume $k = 1$. We call (q, u) *abnormal* if it is the projection of an abnormal extremal. We call (q, u) *normal* if it is the projection of a normal extremal and it is not abnormal.

Theorem 2 (Sufficient Conditions). Suppose $(p, q): [0, 1] \rightarrow T^*M$ is a normal extremal of (1). Define $H \in C^\infty(T^*M)$ by

$$H(p, q) = \max_{u \in U} \hat{H}(p, q, 1, u), \tag{3}$$

assuming the maximum exists and $\partial^2 \hat{H} / \partial u^2 < 0$. Define $u: [0, 1] \rightarrow U$ so $u(t)$ is the unique maximizer of (3) at $(p(t), q(t))$. Assume that X_H is complete and that there

exists no other integral curve (p', q') of X_H satisfying $q(t) = q'(t)$ for all $t \in [0, 1]$. Let $\varphi: \mathbb{R} \times T^*M \rightarrow T^*M$ be the flow of X_H and define the endpoint map $\phi_t: T_{q(0)}^*M \rightarrow M$ by $\phi_t(w) = \pi \circ \varphi(t, w, q(0))$. Then, (q, u) is a local optimum of (1) if and only if there exists no $t \in (0, 1]$ for which ϕ_t is degenerate at $p(0)$.

Proof. See Theorem 21.8 of [1]. \square

2.3 Lie-Poisson Reduction

Let G be a Lie group with identity element $e \in G$. Let $\mathfrak{g} = T_e G$ and $\mathfrak{g}^* = T_e^* G$. For any $q \in G$, define the left translation map $L_q: G \rightarrow G$ by $L_q(r) = qr$ for all $r \in G$. A function $H \in C^\infty(T^*G)$ is left-invariant if $H(T_r^*L_q(w), r) = H(w, s)$ for $w \in T_s^*G$ and $q, r, s \in G$ satisfying $s = L_q(r)$. For $\zeta \in \mathfrak{g}$, let X_ζ be the vector field that satisfies $X_\zeta(q) = T_e L_q(\zeta)$ for all $q \in G$. Define the Lie bracket $[\cdot, \cdot]: \mathfrak{g} \times \mathfrak{g} \rightarrow \mathfrak{g}$ by $[\zeta, \eta] = [X_\zeta, X_\eta](e)$ for all $\zeta, \eta \in \mathfrak{g}$. For $\zeta \in \mathfrak{g}$, the adjoint operator $\text{ad}_\zeta: \mathfrak{g} \rightarrow \mathfrak{g}$ satisfies $\text{ad}_\zeta(\eta) = [\zeta, \eta]$ and the coadjoint operator $\text{ad}_\zeta^*: \mathfrak{g}^* \rightarrow \mathfrak{g}^*$ satisfies $\langle \text{ad}_\zeta^*(\infty), \eta \rangle = \langle \infty, \text{ad}_\zeta \eta \rangle$ for all $\eta \in \mathfrak{g}$ and $\infty \in \mathfrak{g}^*$. The functional derivative of $h \in C^\infty(\mathfrak{g}^*)$ at $\infty \in \mathfrak{g}^*$ is the unique element $\delta h / \delta \infty$ of \mathfrak{g} that satisfies the following for all $\delta \in \mathfrak{g}^*$:

$$\lim_{s \rightarrow 0} \frac{h(\infty + s\delta\infty) - h(\infty)}{s} = \left\langle \delta\infty, \frac{\delta h}{\delta\infty} \right\rangle$$

Theorem 3 (Reduction of Necessary Conditions). Suppose $H: T^*G \times [0, 1] \rightarrow \mathbb{R}$ is both smooth and left-invariant for all $t \in [0, 1]$. Denote the restriction of H to \mathfrak{g}^* by $h = H|_{\mathfrak{g}^* \times [0, 1]}$. Given $p_0 \in T_{q_0}^*G$, let $\infty: [0, 1] \rightarrow \mathfrak{g}^*$ be the solution of

$$\dot{\infty} = \text{ad}_{\delta h / \delta \infty}^*(\infty) \tag{4}$$

with initial condition $\infty(0) = T_e^*L_{q_0}(p_0)$. The integral curve $(p, q): [0, 1] \rightarrow T^*G$ of X_H with initial condition $p(0) = p_0$ satisfies $p(t) = T_{q(t)-1}^*L_{q(t)}(\infty(t))$ for all $t \in [0, 1]$, where q is the solution of $\dot{q} = X_{\delta h / \delta \infty}(q)$ with initial condition $q(0) = q_0$.

Proof. See Theorem 13.4.4 of [27]. \square

It is convenient for us to introduce coordinates on \mathfrak{g} and \mathfrak{g}^* . Let $\{X_1, \dots, X_n\}$ be a basis for \mathfrak{g} and let $\{P_1, \dots, P_n\}$ be the dual basis for \mathfrak{g}^* that satisfies $\langle P_i, X_j \rangle = \delta_{ij}$, where δ_{ij} is the Kronecker delta. We write ζ_i to denote the i th component of $\zeta \in \mathfrak{g}$ with respect to this basis, and so forth. Define the structure constants $C_{ij}^k \in \mathbb{R}$ associated with our choice of basis by

$$[X_i, X_j] = \sum_{k=1}^n C_{ij}^k X_k \tag{5}$$

for $i, j \in \{1, \dots, n\}$. We require two lemmas before our main result (Theorem 4).

Lemma 1. Let $q: U \rightarrow G$ be a smooth map, where $U \subset \mathbb{R}^2$ is simply connected. Denote its partial derivatives $\zeta: U \rightarrow \mathfrak{g}$ and $\eta: U \rightarrow \mathfrak{g}$ by

$$\zeta(t, \varepsilon) = T_{q(t, \varepsilon)} L_{q(t, \varepsilon)^{-1}} \left(\frac{\partial q(t, \varepsilon)}{\partial t} \right) \quad \eta(t, \varepsilon) = T_{q(t, \varepsilon)} L_{q(t, \varepsilon)^{-1}} \left(\frac{\partial q(t, \varepsilon)}{\partial \varepsilon} \right). \quad (6)$$

Then,

$$\partial \zeta / \partial \varepsilon - \partial \eta / \partial t = [\zeta, \eta]. \quad (7)$$

Conversely, if there exist smooth maps ζ and η satisfying (7), then there exists a smooth map q satisfying (6).

Proof. See Proposition 5.1 of [9]. \square

Lemma 2. Let $\alpha, \beta, \gamma \in \mathfrak{g}$ and suppose $\gamma = [\alpha, \beta]$. Then $\gamma_k = \sum_{r=1}^n \sum_{s=1}^n \alpha_r \beta_s C_{rs}^k$.

Proof. This result is easily obtained from the definition (5). \square

Theorem 4 (Reduction of Sufficient Conditions). Suppose that $H \in C^\infty(T^*G)$ is left-invariant and that X_H is complete. Let $h = H|_{\mathfrak{g}^*}$ be the restriction of H to \mathfrak{g}^* and let $\varphi: \mathbb{R} \times T^*G \rightarrow T^*G$ be the flow of X_H . Given $q_0 \in G$, define the endpoint map $\phi_t: T_{q_0}^*G \rightarrow G$ by $\phi_t(p) = \pi \circ \varphi(t, p, q_0)$. Given $p_0 \in T_{q_0}^*G$, let $a \in \mathbb{R}^n$ be the coordinate representation of $T_e^*L_{q_0}(p_0)$, i.e.,

$$T_e^*L_{q_0}(p_0) = \sum_{i=1}^n a_i P_i. \quad (8)$$

Solve the ordinary differential equations

$$\dot{\alpha}_i = - \sum_{j=1}^n \sum_{k=1}^n C_{ij}^k \frac{\delta h}{\delta \alpha_j} \alpha_k \quad i \in \{1, \dots, n\} \quad (9)$$

with the initial conditions $\alpha_i(0) = a_i$. Define matrices $\mathbf{F}, \mathbf{G}, \mathbf{H} \in \mathbb{R}^{n \times n}$ as follows:

$$[\mathbf{F}]_{ij} = - \frac{\partial}{\partial \alpha_j} \sum_{r=1}^n \sum_{s=1}^n C_{ir}^s \frac{\delta h}{\delta \alpha_r} \alpha_s \quad [\mathbf{G}]_{ij} = \frac{\partial}{\partial \alpha_j} \frac{\delta h}{\delta \alpha_i} \quad [\mathbf{H}]_{ij} = - \sum_{r=1}^n \frac{\delta h}{\delta \alpha_r} C_{rj}^i$$

Solve the (linear, time-varying) matrix differential equations

$$\dot{\mathbf{M}} = \mathbf{F}\mathbf{M} \quad (10)$$

$$\dot{\mathbf{J}} = \mathbf{G}\mathbf{M} + \mathbf{H}\mathbf{J} \quad (11)$$

with initial conditions $\mathbf{M}(0) = I$ and $\mathbf{J}(0) = 0$. The endpoint map ϕ_t is degenerate at p_0 if and only if $\det(\mathbf{J}(t)) = 0$.

Proof. Define the smooth map $\rho: \mathbb{R}^n \rightarrow T_{q_0}^*G$ by $\rho(a) = T_{q_0}^*L_{q_0}^{-1}(\sum_{i=1}^n a_i P_i)$. This same expression defines $\rho: \mathbb{R}^n \rightarrow T_{p_0}(T_{q_0}^*G)$ if we identify $T_{q_0}^*G$ with $T_{p_0}(T_{q_0}^*G)$ in the usual way. Given $p_0 = \rho(a)$ for some $a \in \mathbb{R}^n$, there exists non-zero

$\lambda \in T_{p_0}(T_{q_0}^*G)$ satisfying $T_{p_0}\phi_t(\lambda) = 0$ if and only if there exists non-zero $s \in \mathbb{R}^n$ satisfying $T_{\rho(a)}\phi_t(\rho(s)) = 0$. Define $q: [0, 1] \times \mathbb{R}^n \rightarrow G$ by $q(t, a) = \phi_t \circ \rho(a)$. Noting that $\partial q(t, a)/\partial a_j = T_{\rho(a)}\phi_t(T_{q_0}^*L_{q_0^{-1}}(P_j))$ for $j \in \{1, \dots, n\}$, we have

$$T_{\rho(a)}\phi_t(\rho(s)) = \sum_{j=1}^n s_j \frac{\partial q(t, a)}{\partial a_j}.$$

By left translation, $T_{\rho(a)}\phi_t(\rho(s)) = 0$ if and only if

$$0 = \sum_{j=1}^n s_j T_{q(t, a)} L_{q(t, a)^{-1}} \left(\frac{\partial q(t, a)}{\partial a_j} \right). \quad (12)$$

Let $\eta^j(t, a) = T_{q(t, a)} L_{q(t, a)^{-1}} (\partial q(t, a)/\partial a_j)$ for $j \in \{1, \dots, n\}$. Define $\mathbf{J}: [0, 1] \rightarrow \mathbb{R}^{n \times n}$ so that $\mathbf{J}(t)$ has entries $[\mathbf{J}]_{ij} = \eta_i^j(t, a)$, i.e., the j th column of $\mathbf{J}(t)$ is the coordinate representation of $\eta^j(t, a)$ with respect to $\{X_1, \dots, X_n\}$. Then, (12) holds for some $s \neq 0$ if and only if $\det(\mathbf{J}(t)) = 0$. We conclude that ϕ_t is degenerate at p_0 if and only if $\det(\mathbf{J}(t)) = 0$. It remains to show that $\mathbf{J}(t)$ can be computed as described in the theorem. Define $\zeta(t, a) = T_{q(t, a)} L_{q(t, a)^{-1}} (\partial q(t, a)/\partial t)$. Taking $\infty_1(t), \dots, \infty_n(t)$ as coordinates of $\infty(t)$, it is easy to verify that (4) and (9) are equivalent (see [27]). We extend each coordinate function in the obvious way to $\infty: [0, 1] \times \mathbb{R}^n \rightarrow \mathbb{R}$, so $\infty_i(t, a)$ solves (9) with initial condition $\infty_i(0, a) = a_i$. Define $\mathbf{M}: [0, 1] \rightarrow \mathbb{R}^{n \times n}$ by $[\mathbf{M}(t)]_{ij} = \partial \infty_i / \partial a_j$. Differentiating (9), we compute

$$\begin{aligned} [\dot{\mathbf{M}}]_{ij} &= \frac{\partial}{\partial t} \frac{\partial \infty_i}{\partial a_j} = \frac{\partial}{\partial a_j} \frac{\partial \infty_i}{\partial t} = \frac{\partial}{\partial a_j} \left(- \sum_{r=1}^n \sum_{s=1}^n C_{ir}^s \frac{\delta h}{\delta \infty_r} \infty_s \right) \\ &= \sum_{k=1}^n - \frac{\partial}{\partial \infty_k} \left(\sum_{r=1}^n \sum_{s=1}^n C_{ir}^s \frac{\delta h}{\delta \infty_r} \infty_s \right) \frac{\partial \infty_k}{\partial a_j} = \sum_{k=1}^n [\mathbf{F}]_{ik} [\mathbf{M}]_{kj}. \end{aligned}$$

It is clear that $[\mathbf{M}(0)]_{ij} = \delta_{ij}$, so we have verified (10). Next, we have

$$\dot{\eta}^j = \frac{\partial \zeta}{\partial a_j} - [\zeta, \eta^j] = \frac{\partial}{\partial a_j} \frac{\delta h}{\delta \infty} - \left[\frac{\delta h}{\delta \infty}, \eta^j \right]$$

from Lemma 1 and Theorem 3. We write this in coordinates by Lemma 2:

$$\begin{aligned} [\dot{\mathbf{J}}]_{ij} &= \dot{\eta}_i^j = \sum_{k=1}^n \left(\frac{\partial}{\partial \infty_k} \frac{\delta h}{\delta \infty} \right) \frac{\partial \infty_k}{\partial a_j} + \sum_{k=1}^n \left(- \sum_{r=1}^n \frac{\delta h}{\delta \infty_r} C_{rk}^i \right) \eta_k^j \\ &= \sum_{k=1}^n [\mathbf{G}]_{ik} [\mathbf{M}]_{kj} + \sum_{k=1}^n [\mathbf{H}]_{ik} [\mathbf{J}]_{kj}. \end{aligned}$$

It is clear that $[\mathbf{J}(0)]_{ij} = 0$, so we have verified (11). \square

3 Mechanics and Manipulation of an Elastic Rod

The previous section derived coordinate formulae to compute necessary and sufficient conditions for a particular class of optimal control problems on manifolds. Here, we apply these results to a Kirchhoff elastic rod. Section 3.1 recalls that the framed curve traced by the rod in static equilibrium is a local solution to a geometric optimal control problem [8, 41]. Section 3.2 proves that the set of all trajectories that are normal with respect to this problem is a smooth manifold of finite dimension that can be parameterized by a single chart (Theorem 6). Section 3.3 proves that the set of all normal trajectories that are also local optima is an open subset of this smooth manifold, and provides a computational test for membership in this subset (Theorem 7). Together, these two results suffice to describe all possible configurations of the elastic rod that can be achieved by quasi-static manipulation. Section 3.4 explains why these results make the problem of manipulation planning easy to solve.

3.1 Model

We refer to the object in Figure 1 as a *rod*. Assuming that it is thin, inextensible, and of unit length, we describe the shape of this rod by a continuous map $q: [0, 1] \rightarrow G$, where $G = SE(3)$. Abbreviating $T_e L_q(\zeta) = q\zeta$, we require this map to satisfy

$$\dot{q} = q(u_1 X_1 + u_2 X_2 + u_3 X_3 + X_4) \quad (13)$$

for some $u: [0, 1] \rightarrow U$, where $U = \mathbb{R}^3$ and

$$\{X_1, \dots, X_6\} = \left\{ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \right\}$$

is a basis for \mathfrak{g} . Denote the dual basis for \mathfrak{g}^* by $\{P_1, \dots, P_6\}$. We refer to q and u together as $(q, u): [0, 1] \rightarrow G \times U$ or simply as (q, u) . Each end of the rod is held by a robotic gripper, which we assume fix arbitrary $q(0)$ and $q(1)$. We further assume, without loss of generality, that $q(0) = e$. We denote the space of all $q(1)$ by $\mathcal{B} = G$. Finally, we assume that the rod is elastic in the sense of Kirchhoff [8], so has total elastic energy $\frac{1}{2} \int_0^1 (c_1 u_1^2 + c_2 u_2^2 + c_3 u_3^2) dt$ for given constants $c_1, c_2, c_3 > 0$. For fixed endpoints, the wire is motionless only if its shape locally minimizes energy. In particular, we say that (q, u) is in static equilibrium if it is a local optimum of

$$\begin{aligned} \underset{q, u}{\text{minimize}} \quad & \frac{1}{2} \int_0^1 (c_1 u_1^2 + c_2 u_2^2 + c_3 u_3^2) dt \\ \text{subject to} \quad & \dot{q} = q(u_1 X_1 + u_2 X_2 + u_3 X_3 + X_4) \\ & q(0) = e, \quad q(1) = b \end{aligned} \quad (14)$$

for some $b \in \mathcal{B}$.

3.2 Necessary Conditions for Static Equilibrium

Theorem 5. A trajectory (q, u) is normal with respect to (14) if and only if there exists $\infty: [0, 1] \rightarrow \mathfrak{g}^*$ that satisfies

$$\begin{aligned}\dot{\infty}_1 &= u_3 \infty_2 - u_2 \infty_3 & \dot{\infty}_4 &= u_3 \infty_5 - u_2 \infty_6 \\ \dot{\infty}_2 &= \infty_6 + u_1 \infty_3 - u_3 \infty_1 & \dot{\infty}_5 &= u_1 \infty_6 - u_3 \infty_4 \\ \dot{\infty}_3 &= -\infty_5 + u_2 \infty_1 - u_1 \infty_2 & \dot{\infty}_6 &= u_2 \infty_4 - u_1 \infty_5,\end{aligned}\quad (15)$$

$$\dot{q} = q(u_1 X_1 + u_2 X_2 + u_3 X_3 + X_4), \quad (16)$$

$$u_i = c_i^{-1} \infty_i \quad \text{for all } i \in \{1, 2, 3\}, \quad (17)$$

with initial conditions $q(0) = e$ and $\infty(0) = \sum_{i=1}^6 a_i P_i$ for some $a \in \mathcal{A}$, where

$$\mathcal{A} = \left\{ a \in \mathbb{R}^6 : (a_2, a_3, a_5, a_6) \neq (0, 0, 0, 0) \right\}.$$

Proof. We begin by showing that (q, u) is abnormal if and only if $u_2 = u_3 = 0$. Theorem 1 tells us that (q, u) is abnormal if and only if it is the projection of an integral curve (p, q) of X_H that satisfies (2), where $H(p, q, t) = \widehat{H}(p, q, 0, u(t))$ and

$$\widehat{H}(p, q, 0, u) = \langle p, q(u_1 X_1 + u_2 X_2 + u_3 X_3 + X_4) \rangle.$$

For any $g, r \in G$ satisfying $q = gr$, we compute

$$\begin{aligned}H(T_r^* L_g(p), r, t) &= \langle T_r^* L_g(p), g^{-1} q(u_1 X_1 + u_2 X_2 + u_3 X_3 + X_4) \rangle \\ &= \langle p, g(g^{-1} q(u_1 X_1 + u_2 X_2 + u_3 X_3 + X_4)) \rangle \\ &= \langle p, q(u_1 X_1 + u_2 X_2 + u_3 X_3 + X_4) \rangle = H(p, q, t),\end{aligned}\quad (18)$$

so H is left-invariant. Then, the existence of (p, q) satisfying Theorem 1 is equivalent to the existence of ∞ satisfying Theorem 3: $\dot{\infty} = \text{ad}_{\delta h/\delta \infty}^*(\infty)$ and $\dot{q} = q(\delta h/\delta \infty)$, where $h = H|_{\mathfrak{g}^*}$. Application of (9) produces the formulae (15)-(16), where we require $\infty_1 = \infty_2 = \infty_3 = 0$ to satisfy (2). We therefore have $\dot{\infty}_2 = \infty_6$ and $\dot{\infty}_3 = -\infty_5$, hence $\infty_5 = \infty_6 = 0$. Applying this result again to (15), we find $\dot{\infty}_5 = -u_3 \infty_4 = 0$ and $\dot{\infty}_6 = u_2 \infty_4 = 0$. Since ∞ cannot vanish when $k = 0$, we must have $\infty_4 \neq 0$, hence $u_2 = u_3 = 0$, with u_1 an arbitrary integrable function. Our result follows.

Now, we return to the normal case. As before, Theorem 1 tells us that (q, u) is normal if and only if it is not abnormal and it is the projection of an integral curve (p, q) of X_H that satisfies (2), where $H(p, q, t) = \widehat{H}(p, q, 1, u(t))$ and

$$\widehat{H}(p, q, 1, u) = \langle p, q(u_1 X_1 + u_2 X_2 + u_3 X_3 + X_4) \rangle - (c_1 u_1^2 + c_2 u_2^2 + c_3 u_3^2) / 2.$$

By a computation identical to (18), H is left-invariant. Application of (9) to the conditions of Theorem 3 produces the same formulae (15)-(16), where (17) follows from (2) because \widehat{H} is quadratic in u . It remains to show that trajectories produced by (15)-(17) are not abnormal if and only if $a \in \mathcal{A}$. We prove the converse.

First, assume $a \in \mathbb{R}^6 \setminus \mathcal{A}$, so $(a_2, a_3, a_5, a_6) = (0, 0, 0, 0)$. From (15) and (17), we see that $u_2 = u_3 = 0$, hence (q, u) is abnormal. Now, assume (q, u) is abnormal, so $u_2 = u_3 = 0$. From (17), we therefore have $\dot{\infty}_2 = \dot{\infty}_3 = 0$, and in particular $a_2 = a_3 = 0$. Plugging this result into (15), we see that $\dot{\infty}_2 = \dot{\infty}_6$ and $\dot{\infty}_3 = -\dot{\infty}_5$, hence also that $\ddot{\infty}_5 = \ddot{\infty}_6 = 0$, i.e., that $a_5 = a_6 = 0$. So, $a \in \mathbb{R}^6 \setminus \mathcal{A}$. Our result follows. \square

Theorem 5 provides a set of candidates for local optima of (14), which we now characterize. Denote the set of all smooth maps $(q, u): [0, 1] \rightarrow G \times U$ under the smooth topology by $C^\infty([0, 1], G \times U)$. Let $\mathcal{C} \subset C^\infty([0, 1], G \times U)$ be the subset of all (q, u) that satisfy Theorem 5. Any such $(q, u) \in \mathcal{C}$ is completely defined by the choice of $a \in \mathcal{A}$, as is the corresponding ∞ . Denote the resulting map by $\Psi(a) = (q, u)$ and $\Gamma(a) = \infty$. We require three lemmas before our main result (Theorem 6).

Lemma 3. *If $\Psi(a) = \Psi(a')$ for some $a, a' \in \mathcal{A}$, then $a = a'$.*

Proof. Suppose $(q, u) = \Psi(a)$ and $\infty = \Gamma(a)$ for some $a \in \mathcal{A}$. It suffices to show that a is uniquely defined by u (and its derivatives, since u is clearly smooth). From (17), we have $a_i = c_i u_i(0)$ for $i \in \{1, 2, 3\}$. From (15), we have

$$a_5 = -c_3 \dot{u}_3(0) + a_1 a_2 (c_2^{-1} - c_1^{-1}) \quad a_6 = c_2 \dot{u}_2(0) - a_1 a_3 (c_1^{-1} - c_3^{-1}). \quad (19)$$

It is now possible to compute $\dot{\infty}_i(0)$ and $\ddot{\infty}_i(0)$ for $i \in \{4, 5, 6\}$ by differentiation of

$$\dot{\infty}_4 = u_3 \infty_5 - u_2 \infty_6 \quad \dot{\infty}_5 = -\dot{\infty}_3 + u_2 \infty_1 - u_1 \infty_2 \quad \dot{\infty}_6 = \dot{\infty}_2 - u_1 \infty_3 - u_3 \infty_1. \quad (20)$$

Based on these results, we differentiate (15) again to produce

$$\begin{aligned} (c_3^{-1} a_3) a_4 &= c_1^{-1} a_1 a_6 - \dot{\infty}_5(0) \\ (c_2^{-1} a_2) a_4 &= c_1^{-1} a_1 a_5 + \dot{\infty}_6(0) \\ (-a_5 + a_1 a_2 (c_2^{-1} - c_1^{-1})) a_4 &= c_3 (c_1^{-1} (\dot{\infty}_1(0) a_6 + a_1 \dot{\infty}_6(0)) - \ddot{\infty}_5(0)) - a_3 \dot{\infty}_4(0) \\ (a_6 + a_1 a_3 (c_1^{-1} - c_3^{-1})) a_4 &= c_2 (c_1^{-1} (\dot{\infty}_1(0) a_5 + a_1 \dot{\infty}_5(0)) + \ddot{\infty}_6(0)) - a_2 \dot{\infty}_4(0). \end{aligned} \quad (21)$$

At least one of these four equations allows us to compute a_4 unless $(a_2, a_3, a_5, a_6) = (0, 0, 0, 0)$, which would violate our assumption that $a \in \mathcal{A}$. Our result follows. \square

Lemma 4. *The map $\Psi: \mathcal{A} \rightarrow \mathcal{C}$ is a homeomorphism.*

Proof. The map Ψ is clearly a bijection—it is well-defined and onto by construction, and is one-to-one by Lemma 3. Continuity of Ψ also follows from Theorem 5. It remains to show that $\Psi^{-1}: \mathcal{C} \rightarrow \mathcal{A}$ is continuous. This result is a corollary to the proof of Lemma 3. It is immediate that a_1, a_2, a_3 depend continuously on $u(0)$. From (19), we see that a_5, a_6 depend continuously on $a_1, a_2, a_3, \dot{u}(0)$, hence on $u(0), \dot{u}(0)$. From (20), we see in the same way that $\dot{\infty}_4(0), \dot{\infty}_5(0), \dot{\infty}_6(0), \ddot{\infty}_5(0), \ddot{\infty}_6(0)$ depend continuously on $u(0), \dot{u}(0), \ddot{u}(0)$. Hence, all of the quantities in (21) depend continuously on $u(0), \dot{u}(0), \ddot{u}(0)$, so a_4 does as well. Our result follows. \square

Lemma 5. If the topological n -manifold M has an atlas consisting of the single chart (M, α) , then $N = \alpha(M)$ is a topological n -manifold with an atlas consisting of the single chart (N, id_N) , where id_N is the identity map. Furthermore, both M and N are smooth n -manifolds and $\alpha: M \rightarrow N$ is a diffeomorphism.

Proof. Since (M, α) is a chart, then N is an open subset of \mathbb{R}^n and α is a bijection. Hence, our first result is immediate and our second result requires only that both α and α^{-1} are smooth maps. For every $p \in M$, the charts (M, α) and (N, id_N) satisfy $\alpha(p) \in N$, $\alpha(M) = N$, and $\text{id}_N \circ \alpha \circ \alpha^{-1} = \text{id}_N$, so α is a smooth map. For every $q \in N$, the charts (N, id_N) and (M, α) again satisfy $\alpha^{-1}(q) \in M$, $\alpha^{-1}(N) = M$, and $\alpha \circ \alpha^{-1} \circ \text{id}_N = \text{id}_N$, so α^{-1} is also a smooth map. Our result follows. \square

Theorem 6. \mathcal{C} is a smooth 6-manifold with smooth structure determined by an atlas with the single chart (\mathcal{C}, Ψ^{-1}) .

Proof. Since $\Psi: \mathcal{A} \rightarrow \mathcal{C}$ is a homeomorphism by Lemma 4 and $\mathcal{A} \subset \mathbb{R}^6$ is open, then (\mathcal{C}, Ψ^{-1}) is a chart whose domain is \mathcal{C} . Our result follows from Lemma 5. \square

3.3 Sufficient Conditions for Static Equilibrium

Theorem 7. Let $(q, u) = \Psi(a)$ and $\infty = \Gamma(a)$ for some $a \in \mathcal{A}$. Define

$$\mathbf{F} = \begin{bmatrix} 0 & \infty_3(c_3^{-1} - c_2^{-1}) & \infty_2(c_3^{-1} - c_2^{-1}) & 0 & 0 & 0 \\ \infty_3(c_1^{-1} - c_3^{-1}) & 0 & \infty_1(c_1^{-1} - c_3^{-1}) & 0 & 0 & 1 \\ \infty_2(c_2^{-1} - c_1^{-1}) & \infty_1(c_2^{-1} - c_1^{-1}) & 0 & 0 & -1 & 0 \\ 0 & -\infty_6/c_2 & \infty_5/c_3 & 0 & \infty_3/c_3 & -\infty_2/c_2 \\ \infty_6/c_1 & 0 & -\infty_4/c_3 & -\infty_3/c_3 & 0 & \infty_1/c_1 \\ -\infty_5/c_1 & \infty_4/c_2 & 0 & \infty_2/c_2 & -\infty_1/c_1 & 0 \end{bmatrix}$$

$$\mathbf{G} = \text{diag}(c_1^{-1}, c_2^{-1}, c_3^{-1}, 0, 0, 0)$$

$$\mathbf{H} = \begin{bmatrix} 0 & \infty_3/c_3 & -\infty_2/c_2 & 0 & 0 & 0 \\ -\infty_3/c_3 & 0 & \infty_1/c_1 & 0 & 0 & 0 \\ \infty_2/c_2 & -\infty_1/c_1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \infty_3/c_3 & -\infty_2/c_2 \\ 0 & 0 & 1 & -\infty_3/c_3 & 0 & \infty_1/c_1 \\ 0 & -1 & 0 & \infty_2/c_2 & -\infty_1/c_1 & 0 \end{bmatrix}.$$

Solve the (linear, time-varying) matrix differential equations

$$\dot{\mathbf{M}} = \mathbf{F}\mathbf{M} \quad \dot{\mathbf{J}} = \mathbf{G}\mathbf{M} + \mathbf{H}\mathbf{J} \quad (22)$$

with initial conditions $\mathbf{M}(0) = I$ and $\mathbf{J}(0) = 0$. Then, (q, u) is a local optimum of (14) for $b = q(1)$ if and only if $\det(\mathbf{J}(t)) \neq 0$ for all $t \in (0, 1]$.

Proof. As we have already seen, normal extremals of (14) are derived from the parameterized Hamiltonian function

$$\widehat{H}(p, q, 1, u) = \langle p, q(u_1 X_1 + u_2 X_2 + u_3 X_3 + X_4) \rangle - (c_1 u_1^2 + c_2 u_2^2 + c_3 u_3^2) / 2.$$

This function satisfies $\partial^2 \hat{H} / \partial u^2 = -\text{diag}(c_1, c_2, c_3) < 0$ and admits a unique maximum at $u_i = \langle p, qX_i \rangle / c_i$ for $i \in \{1, 2, 3\}$. The maximized Hamiltonian function is

$$H(p, q) = \langle p, qX_4 \rangle + \left(c_1^{-1} \langle p, qX_1 \rangle^2 + c_2^{-1} \langle p, qX_2 \rangle^2 + c_3^{-1} \langle p, qX_3 \rangle^2 \right) / 2.$$

It is clear that X_H is complete. By Lemma 3, the mapping from (q, u) to a and hence to $\infty = \Gamma(a)$ is unique. By Theorem 3, it is equivalent that the mapping from (q, u) to (p, q) is unique. As a consequence, we may apply Theorem 2 to establish sufficient conditions for optimality. Since a computation identical to (18) shows that H is left-invariant, we may apply the equivalent conditions of Theorem 4. Noting that the restriction $h = H|_{\mathfrak{g}^*} \in C^\infty(\mathfrak{g}^*)$ is given by

$$h(\infty) = \infty + (c_1^{-1} \infty_1^2 + c_2^{-1} \infty_2^2 + c_3^{-1} \infty_3^2) / 2$$

it is easy to verify that \mathbf{F} , \mathbf{G} and \mathbf{H} take the form given above. Our result follows. \square

Theorem 7 provides a computational test of which points $a \in \mathcal{A}$ actually produce local optima $\Psi(a) \in \mathcal{C}$ of (14). Let $\mathcal{A}_{\text{stable}} \subset \mathcal{A}$ be the subset of all a for which the conditions of Theorem 7 are satisfied and let $\mathcal{C}_{\text{stable}} = \Psi(\mathcal{A}_{\text{stable}}) \subset \mathcal{C}$. An important consequence of membership in $\mathcal{A}_{\text{stable}}$ is smooth local dependence of (14) on variation in b . Define $\mathcal{B}_{\text{stable}} = \{q(1) \in \mathcal{B} : (q, u) \in \mathcal{C}_{\text{stable}}\}$ and let $\Phi : \mathcal{C} \rightarrow \mathcal{B}$ be the map taking (q, u) to $q(1)$. Clearly $\mathcal{A}_{\text{stable}}$ is open, so $\Psi|_{\mathcal{A}_{\text{stable}}} : \mathcal{A}_{\text{stable}} \rightarrow \mathcal{C}_{\text{stable}}$ is a diffeomorphism. We arrive at the following result:

Theorem 8. *The map $\Phi \circ \Psi|_{\mathcal{A}_{\text{stable}}} : \mathcal{A}_{\text{stable}} \rightarrow \mathcal{B}_{\text{stable}}$ is a local diffeomorphism.*

Proof. The map $\Phi \circ \Psi|_{\mathcal{A}_{\text{stable}}}$ is smooth and by Theorem 7 has non-singular Jacobian $\mathbf{J}(1)$. Our result follows from the Implicit Function Theorem [25, Theorem 7.9]. \square

3.4 Application to Manipulation Planning

Recall that we want to find a path of the gripper that causes the rod to move between given start and goal configurations while remaining in static equilibrium. It is equivalent to find a path of the rod through its set of equilibrium configurations. We showed that any equilibrium configuration can be represented by a point in $\mathcal{A}_{\text{stable}} \subset \mathcal{A} \subset \mathbb{R}^6$. Think of \mathcal{A} as the “configuration space” of the rod during quasi-static manipulation and of $\mathcal{A}_{\text{stable}}$ as the “free space.” Theorems 5-6 say how to map points in \mathcal{A} to configurations of the rod. Theorem 7 says how to test membership in $\mathcal{A}_{\text{stable}}$, i.e., it provides a “collision checker.” Theorem 8 says that paths in $\mathcal{A}_{\text{stable}}$ can be “implemented” by the gripper, by establishing a well-defined map between differential changes in the rod (represented by $\mathcal{A}_{\text{stable}}$) and in the gripper (represented by $\mathcal{B}_{\text{stable}}$). We have expressed the manipulation planning problem for an elastic rod as a standard motion planning problem in a configuration space of dimension 6, for which there are hundreds of solution approaches [23, 11, 24].

For the sake of completeness, here is one way to implement a sampling-based planning algorithm like PRM [19]:

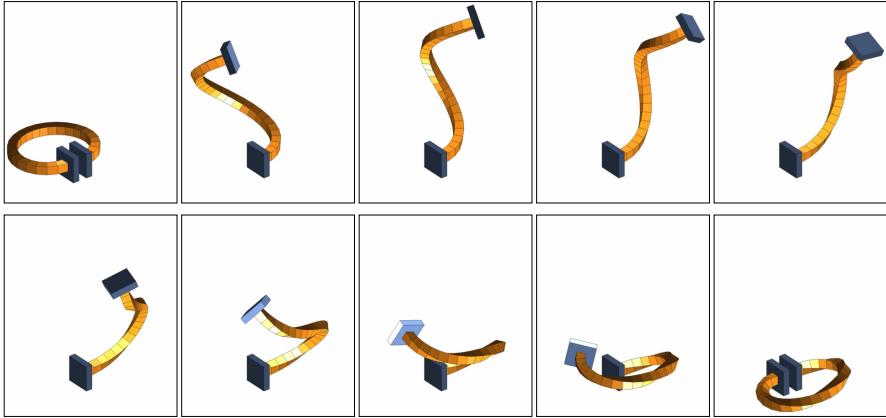


Fig. 2 A second example of quasi-static manipulation by robotic grippers (blue) of an elastic rod (orange). Again, the grippers begin and end in the same position and orientation. And again, this motion corresponds to a straight-line path in the global coordinate chart \mathcal{A} that we derived.

- Sample points in \mathcal{A} , for example uniformly at random in $\{a \in \mathcal{A} : \|a\|_\infty \leq w\}$ for some $w > 0$. Note that it is possible to choose w in practice by taking advantage of the direct correspondence (which we do not discuss here) between a and forces/torques at the base of the elastic rod.
- Keep points that are in $\mathcal{A}_{\text{stable}}$ and add them as nodes in the roadmap. This test requires only solving the ordinary differential equations (15)-(17) in 6 variables and the matrix differential equations (22) in 72 variables.
- Try to connect each pair of nodes a and a' with a straight-line path in \mathcal{A} , adding this path as an edge in the roadmap if it lies entirely in $\mathcal{A}_{\text{stable}}$. This test can be approximated in the usual way by sampling points along the straight-line path at some resolution, again solving (15)-(17) and (22) for each point.
- Declare $a_{\text{start}}, a_{\text{goal}} \in \mathcal{A}_{\text{stable}}$ to be path-connected if they are connected by a sequence of nodes and edges in the roadmap. This sequence is a continuous and piecewise-smooth map $\alpha : [0, 1] \rightarrow \mathcal{A}_{\text{stable}}$, where $\alpha(0) = a_{\text{start}}$ and $\alpha(1) = a_{\text{goal}}$.
- Move the robotic gripper along the path $\Phi \circ \Psi|_{\mathcal{A}_{\text{stable}}} \circ \alpha : [0, 1] \rightarrow \mathcal{B}_{\text{stable}}$. This path is again continuous and piecewise-smooth, and can be evaluated at waypoints $s \in [0, 1]$ by solving the matrix differential equation (16) on $SE(3)$.

Each step is trivial to implement using modern numerical methods. It is also easy to include other constraints, such as self-collision, within this basic framework.

We emphasize that the “start” and “goal” for the manipulation planning problem must be points in $\mathcal{A}_{\text{stable}}$, or equivalently points in $\mathcal{C}_{\text{stable}}$ through the diffeomorphism Ψ . It is insufficient to specify start and goal by points in $\mathcal{B}_{\text{stable}}$, since these points do not uniquely define configurations of the elastic rod.

Figure 2 shows another example result. The start and goal configurations are both associated with the same boundary conditions, each one being a different local

minimum of total elastic energy. The motion of the rod therefore could not possibly correspond to a single straight-line path in $\mathcal{B}_{\text{stable}}$, where planning has traditionally been done (e.g., [21, 29]). However, this motion does indeed correspond to a single straight-line path in $\mathcal{A}_{\text{stable}}$ and was trivial to generate with our approach. We have constructed many similar examples, all of which point to favorable visibility properties of $\mathcal{A}_{\text{stable}}$ and lead us to expect standard motion planning algorithms to perform well in this context [19, 11, 24]. We note further that a number of planning heuristics like lazy collision-checking [34]—which bring huge speed-ups in practice—are easy to apply when planning in $\mathcal{A}_{\text{stable}}$ but hard to apply when planning in $\mathcal{B}_{\text{stable}}$. Finally, should we still want to plan in $\mathcal{B}_{\text{stable}}$ (i.e., to connect nearby configurations by straight-line paths in $\mathcal{B}_{\text{stable}}$ rather than in $\mathcal{A}_{\text{stable}}$), it is now easy to do so by using the Jacobian matrix $\mathbf{J}(1)$, which is non-singular in $\mathcal{B}_{\text{stable}}$ by construction. In particular, we have the relationship $\delta b = \mathbf{J}(1)\delta a$, which can be inverted to move along straight lines in $\mathcal{B}_{\text{stable}}$. Without this relationship, we would be forced to apply gradient descent in the infinite-dimensional space of inputs $u: [0, 1] \rightarrow U$, prompting methods of approximation like the one described in [29].

4 Conclusion

Our contribution in this paper was to show that the set of equilibrium configurations for a Kirchhoff elastic rod held at each end by a robotic gripper is a smooth manifold of finite dimension that can be parameterized by a single (global) coordinate chart. The fact that we ended up with a finite-dimensional smooth manifold is something that might have been guessed in hindsight (it’s dimension—six—is intuitive given that the grippers move in $SE(3)$), but the fact that this manifold admitted a global chart is something that we find remarkable. Our results led to a simple algorithm for manipulation planning, which at the outset had seemed very hard to solve.

A straightforward extension is to implement a sampling-based planner as described in Section 3.4 and perform experiments that compare our approach to others (e.g., [29]) in terms of standard metrics like running time, failure probability, etc. This implementation requires consideration of certain details that we did not address explicitly. For example, to verify static equilibrium, Theorem 7 requires a check that $\det(\mathbf{J}(t))$ does not vanish on $(0, 1]$. We can approximate this check by sampling t , but would prefer an approach with guarantees (as in “exact” collision checking [35]). This is problematic since $\det(\mathbf{J}(t))$ and all its derivatives vanish at $t = 0$.

There are several other opportunities for future work. First, the coordinates we derive can be interpreted as forces and torques at the base of the elastic rod, so \mathcal{A} is exactly the space over which to perform inference in state estimation with a force/torque sensor. Second, our model of an elastic rod depends on three physical parameters $c_1, c_2, c_3 > 0$. Finding these parameters from observations of equilibrium configurations can be cast as an inverse optimal control problem [18]. The structure established by Theorem 6 allows us to define a notion of orthogonal distance between \mathcal{C} and these observations, similar to [20], and may lead to an efficient method of solution. Third, we note that an elastic inextensible strip (or “ribbon”) is a

developable surface whose shape can be reconstructed from its centerline [37]. This centerline conforms to a similar model as the elastic rod and is likely amenable to similar analysis, which may generalize to models of other developable surfaces.

Acknowledgements. This material is based upon work supported by the National Science Foundation under CPS-0931871 and CMMI-0956362. The authors would like to thank Don Shimamoto for helpful comments, leading to the current proof of Lemma 4. Thanks also to the organizers, anonymous reviewers, and audience at WAFR 2012, whose suggestions improved this paper.

References

1. Agrachev, A.A., Sachkov, Y.L.: Control theory from the geometric viewpoint, vol. 87. Springer, Berlin (2004)
2. Amato, N.M., Song, G.: Using motion planning to study protein folding pathways. *Journal of Computational Biology* 9(2), 149–168 (2002)
3. Antman, S.S.: Nonlinear Problems of Elasticity, 2nd edn. Applied Mathematical Sciences, vol. 107. Springer, New York (2005)
4. Asano, Y., Wakamatsu, H., Morinaga, E., Arai, E., Hirai, S.: Deformation path planning for manipulation of flexible circuit boards. In: IEEE/RSJ Int. Conf. Int. Rob. Sys. (2010)
5. Bell, M., Balkcom, D.: Knot tying with single piece fixtures. In: Int. Conf. Rob. Aut. (2008)
6. van den Berg, J., Miller, S., Goldberg, K., Abbeel, P.: Gravity-based robotic cloth folding. In: WAFR (2011)
7. Bergou, M., Wardetzky, M., Robinson, S., Audoly, B., Grinspun, E.: Discrete elastic rods. *ACM Trans. Graph.* 27(3), 1–12 (2008)
8. Biggs, J., Holderbaum, W., Jurdjevic, V.: Singularities of optimal control problems on some 6-d lie groups. *IEEE Trans. Autom. Control* 52(6), 1027–1038 (2007)
9. Bloch, A., Krishnaprasad, P., Marsden, J., Ratiu, T.: The Euler-Poincaré equations and double bracket dissipation. *Communications In Mathematical Physics* 175(1), 1–42 (1996)
10. Chirikjian, G.S., Burdick, J.W.: The kinematics of hyper-redundant robot locomotion. *IEEE Trans. Robot. Autom.* 11(6), 781–793 (1995)
11. Choset, H., Lynch, K., Hutchinson, S., Kanto, G., Burgard, W., Kavraki, L., Thrun, S.: Principles of Robot Motion: Theory, Algorithms, and Implementations. MIT Press (2005)
12. Clements, T.N., Rahn, C.D.: Three-dimensional contact imaging with an actuated whisker. *IEEE Trans. Robot.* 22(4), 844–848 (2006)
13. Gopalakrishnan, K., Goldberg, K.: D-space and deform closure grasps of deformable parts. *International Journal of Robotics Research* 24(11), 899–910 (2005)
14. Hoffman, K.A.: Methods for determining stability in continuum elastic-rod models of dna. *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* 362(1820), 1301–1315 (2004)
15. Hopcroft, J.E., Kearney, J.K., Krafft, D.B.: A case study of flexible object manipulation. *The International Journal of Robotics Research* 10(1), 41–50 (1991)
16. Inoue, H., Inaba, H.: Hand-eye coordination in rope handling. In: ISRR, pp. 163–174 (1985)

17. Jansen, R., Hauser, K., Chentanez, N., van der Stappen, F., Goldberg, K.: Surgical retraction of non-uniform deformable layers of tissue: 2d robot grasping and path planning. In: IEEE/RSJ Int. Conf. Int. Rob. Sys., pp. 4092–4097 (2009)
18. Javdani, S., Tandon, S., Tang, J., O'Brien, J.F., Abbeel, P.: Modeling and perception of deformable one-dimensional objects. In: Int. Conf. Rob. Aut., Shanghai, China (2011)
19. Kavraki, L.E., Svetska, P., Latombe, J.C., Overmars, M.: Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. Autom.* 12(4), 566–580 (1996)
20. Keshavarz, A., Wang, Y., Boyd, S.: Imputing a convex objective function. In: IEEE Multi-Conference on Systems and Control (2011)
21. Lamiriaux, F., Kavraki, L.E.: Planning paths for elastic objects under manipulation constraints. *International Journal of Robotics Research* 20(3), 188–208 (2001)
22. Langer, J., Singer, D.: The total squared curvature of closed curves. *Journal of Differential Geometry* 20, 1–22 (1984)
23. Latombe, J.C.: *Robot Motion Planning*. Kluwer Academic Publishers, Boston (1991)
24. LaValle, S.M.: *Planning algorithms*. Cambridge University Press, New York (2006)
25. Lee, J.M.: *Introduction to smooth manifolds*, vol. 218. Springer, New York (2003)
26. Lin, Q., Burdick, J., Rimon, E.: A stiffness-based quality measure for compliant grasps and fixtures. *IEEE Trans. Robot. Autom.* 16(6), 675–688 (2000)
27. Marsden, J.E., Ratiu, T.S.: *Introduction to mechanics and symmetry: a basic exposition of classical mechanical systems*, 2nd edn. Springer, New York (1999)
28. McCarthy, Z., Bretl, T.: Mechanics and manipulation of planar elastic kinematic chains. In: IEEE Int. Conf. Rob. Aut. St. Paul, MN (2012)
29. Moll, M., Kavraki, L.E.: Path planning for deformable linear objects. *IEEE Trans. Robot.* 22(4), 625–636 (2006)
30. Rucker, D.C., Webster, R.J., Chirikjian, G.S., Cowan, N.J.: Equilibrium conformations of concentric-tube continuum robots. *Int. J. Rob. Res.* 29(10), 1263–1280 (2010)
31. Sachkov, Y.: Conjugate points in the euler elastic problem. *Journal of Dynamical and Control Systems* 14(3), 409–439 (2008)
32. Sachkov, Y.: Maxwell strata in the euler elastic problem. *Journal of Dynamical and Control Systems* 14(2), 169–234 (2008)
33. Saha, M., Isto, P.: Manipulation planning for deformable linear objects. *IEEE Trans. Robot.* 23(6), 1141–1150 (2007)
34. Sánchez, G., Latombe, J.C.: On delaying collision checking in PRM planning: Application to multi-robot coordination. *Int. J. Rob. Res.* 21(1), 5–26 (2002)
35. Schwarzer, F., Saha, M., Latombe, J.C.: Exact collision checking of robot paths. In: WAFR, Nice, France (2002)
36. Solomon, J.H., Hartmann, M.J.Z.: Extracting object contours with the sweep of a robotic whisker using torque information. *Int. J. Rob. Res.* 29(9), 1233–1245 (2010)
37. Starostin, E.L., van der Heijden, G.H.M.: Tension-induced multistability in inextensible helical ribbons. *Physical Review Letters* 101(8), 084,301 (2008)
38. Takamatsu, J., Morita, T., Ogawara, K., Kimura, H., Ikeuchi, K.: Representation for knot-tying tasks. *IEEE Trans. Robot.* 22(1), 65–78 (2006)
39. Tanner, H.: Mobile manipulation of flexible objects under deformation constraints. *IEEE Trans. Robot.* 22(1), 179–184 (2006)
40. Wakamatsu, H., Arai, E., Hirai, S.: Knotting/unknotting manipulation of deformable linear objects. *The International Journal of Robotics Research* 25(4), 371–395 (2006)

41. Walsh, G., Montgomery, R., Sastry, S.: Optimal path planning on matrix lie groups. In: IEEE Conference on Decision and Control, vol. 2, pp. 1258–1263 (1994)
42. Webster, R.J., Jones, B.A.: Design and kinematic modeling of constant curvature continuum robots: A review. *Int. J. Rob. Res.* 29(13), 1661–1683 (2010)
43. Yamakawa, Y., Namiki, A., Ishikawa, M.: Motion planning for dynamic folding of a cloth with two high-speed robot hands and two high-speed sliders. In: Int. Conf. Rob. Aut., pp. 5486–5491 (2011)

From Discrete to Continuous Motion Planning

Nicolas Perrin

Abstract. In this paper, we demonstrate an equivalence between a large class of discrete motion planning problems, and piano mover’s problems, which we refer to as “continuous motion planning problems”. We first prove that under some assumptions, discrete motion planning in d dimensions can be transformed into continuous motion planning in $2d + 1$ dimensions. Then we prove a more specific, similar equivalence for which the number of dimensions of the configuration space does not necessarily have to be increased. We study two simple cases where this theorem applies, and show that it can lead to original and efficient motion planning algorithms, which could probably be applied to a wide range of multi-contact planning problems. We apply this equivalence to a simulation of legged locomotion planning for a hexapod robot.

1 Introduction

After decades of research in motion planning, we now have plenty of tools to solve the quintessential piano mover’s problem. Several sampling-based algorithms are known to be very efficient in practice, such as PRM [13], RRT [17], etc. Some methods have been improved in order to get better convergence properties [12]. There exist also several libraries that contain state-of-the-art implementations of these algorithms and can be used in almost any configuration space, as long as the user defines the validity tests (i.e. collision checks) via the API. Examples include OMPL [18] and KineoWorks(TM). There are also several algorithms for path optimization ([7], [20]), and algorithms that take advantage of parallel architectures to reduce the computation costs [19].

Nicolas Perrin
Department of Advanced Robotics,
Istituto Italiano di Tecnologia,
via Morego, 30, 16163 Genova
e-mail: nicolas.perrin@iit.it

On the other hand, some classes of motion planning problems have been much less studied, and among them hybrid motion planning problems that have a continuous component but whose output must be a finite sequence of configurations (we call them “discrete motion planning problems”). A typical example is multi-contact planning [2] where the desired output is a finite sequence of contact configurations, two consecutive configurations differing by exactly one contact. In this paper we consider a general class of such discrete motion planning problems, and prove that they can be converted into continuous motion planning problems that are in essence nothing else but the piano mover’s problem. More precisely, we first prove that discrete motion planning problems in d -dimensional configuration spaces can be converted into continuous motion planning problems in $(2d + 1)$ -dimensional configuration spaces. Then, we prove a more specific equivalence where an increase of dimensionality is not compulsory, and study two basic examples where it can be applied and leads to original motion planning algorithms. We then show that this equivalence can be advantageously used in real applications, and in particular apply it in simulation to plan the walking motion of a hexapod robot.

2 Notations and Definitions

We only consider metric configuration spaces, and denote by $dist()$ their distance functions. Let CS be a metric configuration space. Here are some important notations that we will extensively use:

- For $X \subset CS$ we denote by \mathring{X} the interior of X , by \bar{X} its closure, and by X^c its complement.
- For $X \subset CS$ and $s \in CS$ we pose $d(s, X) = \inf\{dist(s, s') | s' \in X\}$.
- While $\mathcal{P}(CS)$ denotes the set of subsets of CS , we denote by $\mathcal{P}_K(CS)$ the set of non-empty compact subsets of CS , and by $RegOp^*(CS)$ the set of non-empty bounded regular open subsets of CS , i.e. the non-empty bounded open subsets that are equal to the interior of their closure.
- We denote by $d_H()$ the Hausdorff distance on $\mathcal{P}(CS)$. It turns $\mathcal{P}_K(CS)$ into a metric space in its own right.
- For $s \in CS$ and $r > 0$, we denote by $\mathcal{S}(s, r)$ the closed sphere of center s and radius r , and by $\mathring{\mathcal{S}}(s, r)$ the open sphere of center s and radius r .

We non-ambiguously use the same notations for any metric space other than CS .

Definition 1 (Uniformly Followable Functions)

We say that a function $X : CS_1 \rightarrow RegOp^*(CS_2)$ is “uniformly Γ -followable” for $\Gamma > 0$ if it verifies the three following properties:

1. There exists $0 < \gamma < \Gamma$ such that $\forall s \in CS_1, \forall (s_a, s_b) \in X(s)^2, dist(s_a, s_b) < \gamma$.
2. The function $s \mapsto \bar{X}(s)$ is continuous on CS_1 w.r.t. the Hausdorff metric on $\mathcal{P}_K(CS_2)$, and the function $(s, s') \mapsto d(s, X(s')^c)$ is continuous on CS_1^2 (on CS_1^2 we use the distance $dist_\infty((s_\alpha, s'_\alpha), (s_\beta, s'_\beta)) = \max(dist(s_\alpha, s_\beta), dist(s'_\alpha, s'_\beta))$).
3. For all $\eta > 0$, there exists $\lambda > 0$ such that $\forall (s, s') \in CS_1^2, d_H(\bar{X}(s), \bar{X}(s')) < \lambda$ implies that $\bar{X}(s) \cap \bar{X}(s')$ is non-empty and $d_H(\bar{X}(s), \bar{X}(s) \cap \bar{X}(s')) < \eta$.

Common functions that map elements of a metric configuration space to geometric shapes in another configuration space are often uniformly Γ -followable. This property can be seen as a form of uniform continuity.

3 Problem Definition and Related Work

We first define the class of continuous motion planning problems. As we previously mentioned, we are interested in problems that are in essence nothing but the piano mover's problem, i.e. problems for which classical sampling-based algorithms (e.g. PRM, RRT) readily apply. In particular, we do not let the possibility of adding non-holonomic constraints, or allowing only curvature-bounded paths, etc. So, let CS be a metric configuration space and let $dist()$ be its distance function. The property $C()$ defines the notion of collision-freeness, and the free space $FS = \{s \in CS | C(s)\}$ is assumed to be an open subset of CS .

Continuous motion planning problems are defined as follows:

Definition 2 (Continuous Motion Planning Problems)

INPUT: $CS, C(), s_i \in FS$ and $s_f \in FS$.

OBJECTIVE: find a continuous path $(s(t))_{t \in [0,1]}$ such that $\forall t \in [0, 1], C(s(t))$ (we call "valid" such a continuous path), and such that $s(1) = s_i$ and $s(1) = s_f$.

We will also consider slight variants of these problems where the initial and final configurations are not fixed but must simply belong to some sets.

Now, for discrete motion planning, instead of continuous paths the outputs are finite sequences of configurations: the motion is abrupt between a configuration and the next one. Configurations must still be collision-free, but there is an additional relation $R()$ that defines a relationship between consecutive configurations. Such discrete motion planning problems arise in particular when simplified models are used to solve hybrid motion planning problems, i.e. problems that have both continuous and discrete aspects. For example, in footstep planning for humanoid robots, the motion of the robot is continuous, but the sequence of contacts with the ground is discrete. To make the problem easier we can use a simplified model in which the feasibility of a sequence of footsteps is not directly related to the actual continuous motion of the robot. In that case, $R()$ is the relation that defines which next steps are feasible.

Here is our general definition of discrete motion planning problems:

Definition 3 (Discrete Motion Planning Problems)

INPUT: $CS, C(), R(), s_i \in FS$ and $s_f \in FS$.

OBJECTIVE: find a finite sequence of configurations (s_1, s_2, \dots, s_n) such that $\forall k \in \{1, \dots, n-1\}$, we have $C(s_k)$, $C(s_{k+1})$, and $R(s_k, s_{k+1})$ (we call "valid" such a finite sequence), and such that $s_1 = s_i$ and $s_n = s_f$.

The purpose of this paper is to make a bridge between these two classes, from discrete to continuous motion planning problems.

There is a lot of related work, but the goal is often to transform a continuous motion planning problem into a discrete one. The simple fact that polygonal chains can approximate arbitrarily closely any continuous path can already be seen as a simple equivalence result that has for consequence the probabilistic completeness of most sampling-based algorithms for the piano mover's problem. A similar result for more complex problems is the small-time local controllability property [16] which allows to solve problems that require continuous solutions by looking for discrete sequences of small motions, which can for example belong to a finite collection of motion primitives [3]. Another bridge from continuous to discrete problems concerns collision checks: it has been shown that checking the validity of configurations along a continuous path can be done in a sound way without necessarily having to perform an infinite number of checks [6]. In any case, it is compulsory to make problems discrete if one wants to solve them with computers.

The objective of the present paper, i.e. transforming discrete problems into continuous ones, is much less common, but we can mention [1] where a reduction property shows that for some class of manipulation problems, the existence of a solution path with discrete "grasp" and "release" events is equivalent to the existence of a path where the grasp is continuously modified. This kind of equivalence can be especially useful when the hybrid nature of a problem makes it difficult to be solved. For example, in the problem of footstep planning for humanoid robots, the relationship between a footprint and the next one is continuous, but it is a discrete sequence of footprints that must be found. It is not easy to design an algorithm that would deal with both continuous and discrete aspects of the problem, and that is why the standard approach is to make the problem completely discrete by deciding in advance on a finite set of possible steps ([15], [4]). Some orthogonal approaches try to make the problem completely continuous. For example in [11] and [5], the robot first slides its feet on the ground. In both approaches it is shown that the continuous motions can always be transformed into finite sequences of steps, so the approaches are sound. Unfortunately, they are not complete: if the robot has no other choice but to step over some obstacles a solution can never be found. In [22], a sound and complete approach is proposed for a specific 2D walking robot whose discrete sequences of footsteps are produced from continuous paths. It is also used in [21] for more complex footstep planning. Basically, the result shown in [22] is a particular case of a more general theorem that we state in the present paper, and which can probably have many applications other than footstep planning.

Several algorithms have already been proposed to solve hybrid motion planning problems such as multi-modal or more specifically multi-contact motion planning, and it would be interesting to compare their efficiency to that of the algorithms based on the equivalence proposed in the present paper. In multi-modal motion planning, a finite or discrete number of *modes* correspond each to a submanifold of the configuration space, and the planner must choose a discrete sequence of modes as well as continuous single-mode paths through them. A general algorithm with good convergence properties has been introduced in [9]. It combines a graph search algorithm to find the sequence of modes, together with probabilistic roadmaps to plan the single-mode paths.

4 From Discrete Motion Planning in d Dimensions to Continuous Motion Planning in $2d + 1$ Dimensions

In this section we demonstrate a quite general theorem that shows a strong relationship between discrete and continuous motion planning problems as we have defined them in the previous section. Our goal is to prove that for an arbitrary discrete motion planning problem (with just a few assumptions), it is possible to define an equivalent continuous motion planning problem. So, let us consider a discrete motion planning problem in an d -dimensional metric configuration space CS , defined by the collision-freeness property $C()$, the relation $R()$ and initial and final configurations s_i and s_f . We make only 3 not so restrictive but important assumptions related to the regularity of $R()$:

1. R is symmetric: $R(s_a, s_b) \Rightarrow R(s_b, s_a)$.
2. The sets $R(s) = \{s' \in CS | R(s, s')\}$ are path-connected: $\forall s \in CS, \forall (s_a, s_b) \in R(s)^2$, there exists a continuous path inside $R(s)$ from s_a to s_b .
3. $s \mapsto R(s)$ is a uniformly Γ -followable function from CS to $RegOp^*(CS)$, for some $\Gamma > 0$.

Under these assumptions, we prove that we can always define an equivalent continuous motion planning problem. To do so, the key is to define a new configuration space \tilde{CS} and a new notion of collision-freeness $\tilde{C}()$. \tilde{CS} is simply $CS^2 \times (0, \Gamma)$, i.e. a metric space of dimension $2d + 1$. The definition of $\tilde{C}()$ is a bit more complex.

Definition 4 ($\tilde{C}()$)

$\tilde{C}(s, s', \rho)$ is verified if and only if the two following properties are verified:

1. The set $A(s, s', \rho) = R(s) \cap \mathcal{S}'(s', \rho)$ has a non-empty intersection with the free space: $\exists s_\alpha \in A(s, s', \rho) | C(s_\alpha)$. Note that the function $(s, s', \rho) \mapsto \bar{A}(s, s', \rho)$ is continuous.
2. The set $B(s, s', \rho) = \{s_b \in CS | R(s_b) \supset A(s, s', \rho)\}$ has also a non-empty intersection with the free space. Note that we always have $s \in B(s, s', \rho)$.

Obviously, verifying the property $\tilde{C}()$ might be much more difficult than verifying $C()$, but we will discuss this later. We first demonstrate the following theorem:

Theorem 1. *The discrete motion planning problems in CS defined by $C()$ and $R()$ are equivalent to the continuous motion planning problems in \tilde{CS} defined by $\tilde{C}()$: for any initial and final configurations $s_i \in CS$ and $s_f \in CS$, there exists a valid discrete sequence of configurations from s_i to s_f if and only if there exists a continuous path $(v(t))_{t \in [0, 1]} \in \tilde{CS}^{[0, 1]}$ such that $\forall t \in [0, 1], \tilde{C}(v(t))$, and $s_i \in B(v(0))$, and $s_f \in A(v(1))$ or $s_f \in B(v(1))$.*

Section 4.1 and Section 4.2 are dedicated to the proof of each implication of this equivalence, but first we state the following lemma (the proof is straightforward and not given here):

Lemma 1. Let $Y : [0, 1] \rightarrow \text{RegOp}^*(\text{CS})$ be a function such that $t \mapsto \bar{Y}(t)$ is continuous on $[0, 1]$, and let $\kappa : \text{CS} \times [0, 1] \rightarrow \mathbb{R}$ be a continuous function. Then $t \mapsto \sup\{\kappa(s, t) | s \in Y(t)\}$ is continuous on $[0, 1]$.

4.1 From a Valid Discrete Sequence in CS to a Valid Continuous Path in $\widetilde{\text{CS}}$

Theorem 2. If there exists a valid sequence $(s_1 = s_i, s_2, \dots, s_n = s_f)$ in CS , then there exists a valid continuous path $(v(t))_{t \in [0, 1]}$ in $\widetilde{\text{CS}}$ such that $v(0)$ is of the form (s_1, s_2, ρ) , and $v(1)$ of the form (s_{n-1}, s_n, ρ) or (s_n, s_{n-1}, ρ) .

Proof. We prove this implication by induction on n , the size of the valid sequence. For $n = 2$ the result is obvious since for any $0 < \rho < \Gamma$ we have $s_2 \in A(s_1, s_2, \rho)$, and, $A(s_1, s_2, \rho)$ being a subset of $R(s_1)$, we have also $s_1 \in B(s_1, s_2, \rho)$. Hence, the stationary path such that $\forall t \in [0, 1], v(t) = (s_1, s_2, \rho)$, is valid.

Let us now assume that the result is true for any sequence of size n , and consider a valid sequence of size $n + 1$: $(s_1 = s_i, s_1, \dots, s_{n+1} = s_f)$. Let $(v(t))_{t \in [0, 1]} \in \widetilde{\text{CS}}^{[0, 1]}$ be a valid path such that $v(0) = (s_1, s_2, \rho_0)$, and $v(1) = (s_{n-1}, s_n, \rho_1)$ or $v(1) = (s_n, s_{n-1}, \rho_1)$.

First, we suppose that $v(1) = (s_{n-1}, s_n, \rho_1)$.

We have $s_{n-1} \in R(s_n)$ and $s_{n+1} \in R(s_n)$. Besides, $R(s_n)$ is path-connected so there exists a continuous path $\infty : [0, 1] \rightarrow \text{CS}$ from s_{n-1} to s_{n+1} inside $R(s_n)$. For any $t \in [0, 1]$, $R(\infty(t))$ is an open set that contains s_n . Since $s \mapsto R(s)$ is uniformly Γ -followable, $t \mapsto d(s_n, R(\infty(t)^c))$ is continuous on $[0, 1]$. Besides, we always have $d(s_n, R(\infty(t)^c)) > 0$. Thus, we deduce that there exists $0 < \xi_{\min} < \Gamma$ such that $\forall t \in [0, 1], R(\infty(t)) \supset \mathcal{S}(s_n, \xi_{\min})$. First, we move continuously from (s_{n-1}, s_n, ρ_1) to $(s_{n-1}, s_n, \xi_{\min})$ (this is a valid path). Then, we follow the path ∞ to move continuously from $(s_{n-1}, s_n, \xi_{\min})$ to $(s_{n+1}, s_n, \xi_{\min})$. Along this path any state $(\infty(t), s_n, \xi_{\min})$ is such that $A(\infty(t), s_n, \xi_{\min}) = \mathcal{S}(s_n, \xi_{\min}) \subset R(s_{n-1}) \cap R(s_{n+1})$, and thus we have $s_n \in A(\infty(t), s_n, \xi_{\min})$, $s_{n-1} \in B(\infty(t), s_n, \xi_{\min})$, and $s_{n+1} \in B(\infty(t), s_n, \xi_{\min})$. As a result, the path is valid. We deduce that appending these two paths to $(v(t))_{t \in [0, 1]}$ gives us a valid continuous path from (s_1, s_2, ρ_0) to $(s_{n+1}, s_n, \xi_{\min})$. This reasoning is illustrated in Fig. 1.

We now suppose that $v(1) = (s_n, s_{n-1}, \rho_1)$.

For $\xi_{\max} > 0$ close enough to Γ , we have $\forall s \in R(s_n), A(s_n, s, \xi_{\max}) = R(s_n)$. First, we move continuously from (s_n, s_{n-1}, ρ_1) to $(s_n, s_{n-1}, \xi_{\max})$ (this is a valid path). Then, we follow a path inside $R(s_n)$ to go from $(s_n, s_{n-1}, \xi_{\max})$ to $(s_n, s_{n+1}, \xi_{\max})$. Along this path, any element (s_n, s, ξ_{\max}) is such that $A(s_n, s, \xi_{\max}) = R(s_n)$, which contains both s_{n-1} and s_{n+1} , and such that $s_n \in B(s_n, s, \xi_{\max})$. Therefore the path is valid (cf. Fig. 2), and appending this path and the previous one to $(v(t))_{t \in [0, 1]}$ gives us a valid continuous path from (s_1, s_2, ρ_0) to $(s_n, s_{n+1}, \xi_{\min})$. That concludes the demonstration of Theorem 2. \square

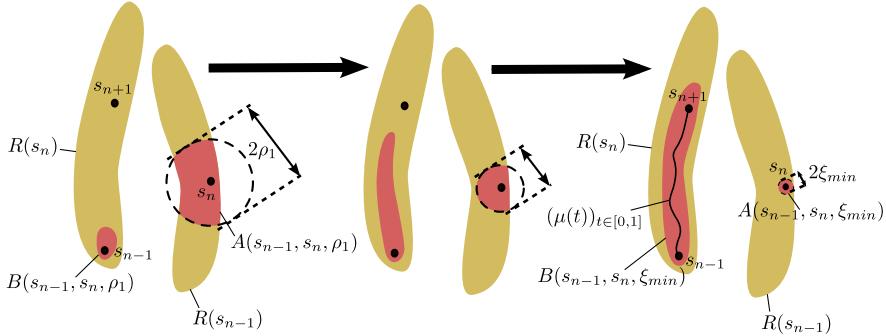


Fig. 1 We illustrate how to go continuously from $v(1) = (s_{n-1}, s_n, \rho_1)$ to $(s_{n+1}, s_n, \xi_{min})$ with a valid path. When decreasing ρ , the size of $B(s_{n-1}, s_n, \rho)$ increases, and when ρ tends to zero, $B(s_{n-1}, s_n, \rho)$ “converges” towards $R(s_n)$. For some value ξ_{min} , we not only have $s_{n+1} \in B(s_{n-1}, s_n, \xi_{min})$, we have $s_{n+1} \in B(\infty(t), s_n, \xi_{min})$ for all configurations $\infty(t)$ along the path from s_{n-1} to s_{n+1} . As a result, the continuous path $(v'(t))$ from $v(1)$ to $(s_{n+1}, s_n, \xi_{min})$ is such that at all time, $A(v'(t))$ contains s_n , which is collision-free, and $B(v'(t))$ contains either s_{n-1} or s_{n+1} , which are both collision-free, and thus $\tilde{C}(v'(t))$ is verified. As a consequence, the path is valid.

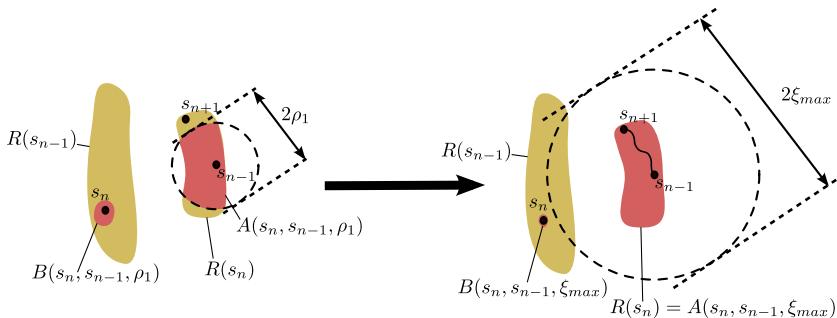


Fig. 2 We illustrate how to go continuously from $v(1) = (s_n, s_{n-1}, \rho_1)$ to $(s_n, s_{n+1}, \xi_{max})$ with a valid path. When increasing ρ , the size of $A(s_n, s_{n-1}, \rho)$ increases, and at some point it becomes equal to $R(s_n)$. For $\rho = \xi_{max}$, we even have: $\forall s \in R(s_n), A(s_n, s, \rho) = R(s_n)$. It follows that we can first go from $v(1)$ to $(s_n, s_{n-1}, \xi_{max})$ with a valid path, and then from $(s_n, s_{n-1}, \xi_{max})$ to $(s_n, s_{n+1}, \xi_{max})$, again with a valid path.

4.2 From a Valid Continuous Path in \widetilde{CS} to a Valid Discrete Sequence in CS

Theorem 3. If there exists a continuous path $(v(t))_{t \in [0,1]} \in \widetilde{CS}^{[0,1]}$ such that $\forall t \in [0,1], \tilde{C}(v(t))$, and $s_i \in B(v(0))$, and $s_f \in A(v(1))$ or $s_f \in B(v(1))$, then there exists in CS a valid finite sequence of configurations from s_i to s_f .

Proof. Let us write: $v(t) = (s_\alpha(t), s_\beta(t), \rho(t))$.

For any collision-free configuration $s \in CS$ and $t \in [0, 1]$ we define:

$$d_{obs}(s, t) = \min(\min\{dist(s, s') | s' \in FS^c\}, \min\{dist(s, s') | s' \in \mathcal{S}(s_\beta(t), \rho(t))^c\})$$

It can be verified that $(s, t) \mapsto d_{obs}(s, t)$ is a continuous function on $CS \times [0, 1]$. We know that $t \mapsto \bar{R}(s_\alpha(t))$ is continuous on $[0, 1]$. Using Lemma 1, it follows that $t \mapsto \delta_{obs}(t) = \sup\{d_{obs}(s, t) | s \in \bar{R}(s_\alpha(t))\}$ is also continuous on $[0, 1]$. Since $\forall t \in [0, 1]$, $\tilde{C}(v(t))$, we can deduce that $\forall t \in [0, 1]$, $\delta_{obs}(t) > 0$. As a result there exists $\Delta_{obs} > 0$ such that $\forall t \in [0, 1]$, $\delta_{obs}(t) > \Delta_{obs}$.

Since $t \mapsto v(t)$ is uniformly continuous on $[0, 1]$, and $s \mapsto R(s)$ is uniformly Γ -followable, there exists $\eta_1 > 0$ such that:

$$\forall (t, t') \in [0, 1], |t - t'| < \eta_1 \Rightarrow d_H(\bar{R}(s_\alpha(t)), \bar{R}(s_\alpha(t')) \cap \bar{R}(s_\alpha(t'))) < \frac{1}{8}\Delta_{obs},$$

with $\bar{R}(s_\alpha(t)) \cap \bar{R}(s_\alpha(t'))$ non-empty.

There also exists $\eta_2 > 0$ such that $\forall (t, t') \in [0, 1], |t - t'| < \eta_2 \Rightarrow dist(s_\beta(t), s_\beta(t')) < \frac{1}{8}\Delta_{obs}$, and $\forall (t, t') \in [0, 1], |t - t'| < \eta_2 \Rightarrow |\rho(t) - \rho(t')| < \frac{1}{8}\Delta_{obs}$. Let N be a positive integer such that $1/N < \min(\eta_1, \eta_2)$.

We now consider $v(0/N), v(1/N), \dots, v(N/N)$ and try to construct a valid sequence $(s_1, s_2, \dots, s_{2N+1})$ such that $s_1 \in B(v(0))$ and $\forall i \in \{1, \dots, N\}$, $s_{2i} \in A(v(i/N))$ and $s_{2i+1} \in B(v(i/N))$. We pose $s_1 = s_i \in B(v(0))$. Now, let us assume that we have been able to construct such a sequence up to s_{2k+1} with $0 \leq k < N$. We try to construct s_{2k+2} and s_{2k+3} . Let us write: $A(v(\frac{k}{N})) = A(s_\alpha, s_\beta, \rho)$ and $A(v(\frac{k+1}{N})) = A(s'_\alpha, s'_\beta, \rho')$. We have:

$$dist(s_\beta, s'_\beta) < \frac{1}{8}\Delta_{obs} \text{ and } |\rho - \rho'| < \frac{1}{8}\Delta_{obs} \text{ and } d_H(\bar{R}(s_\alpha), \bar{R}(s_\alpha) \cap \bar{R}(s'_\alpha)) < \frac{1}{8}\Delta_{obs}$$

There exists s collision-free in $R(s_\alpha)$ such that $d_{obs}(s, k/N) > \frac{1}{2}\Delta_{obs} > 0$. s belongs to $R(s_\alpha) \cap \mathcal{S}(s_\beta, \rho) = A(v(k/N))$. We have $dist(s, s_\beta) < \rho - \frac{1}{2}\Delta_{obs}$ and:

$$\begin{aligned} dist(s, s'_\beta) &< \rho - \frac{1}{2}\Delta_{obs} + dist(s_\beta, s'_\beta) < \rho - \frac{1}{2}\Delta_{obs} + \frac{1}{8}\Delta_{obs} < \rho - |\rho - \rho'| - \frac{1}{4}\Delta_{obs} \\ dist(s, s'_\beta) &< \rho' - \frac{1}{4}\Delta_{obs} \end{aligned}$$

Besides, $d_H(\bar{R}(s_\alpha), \bar{R}(s_\alpha) \cap \bar{R}(s'_\alpha)) < \frac{1}{8}\Delta_{obs}$. Therefore, there exists $s_\cap \in \bar{R}(s_\alpha) \cap \bar{R}(s'_\alpha)$ such that $dist(s_\cap, s) < \frac{1}{8}\Delta_{obs}$. Since $\frac{1}{8}\Delta_{obs} < \frac{1}{2}\Delta_{obs}$, s_\cap is necessarily collision-free. We also have $dist(s_\cap, s'_\beta) < \rho' - \frac{1}{4}\Delta_{obs}$ and $dist(s_\cap, s_\beta) < \rho - \frac{3}{8}\Delta_{obs}$. We deduce that s_\cap is a collision-free configuration that belongs to $A(v(k/N)) \cap A(v((k+1)/N))$. Since we have $s_{2k+1} \in B(v(k/N))$, it follows that $s_\cap \in R(s_{2k+1})$. Therefore, we can set $s_{2k+2} = s_\cap$. For s_{2k+3} , we can take any collision-free configuration inside $B(v((k+1)/N))$.

By iteration, we can obtain a valid sequence $(s_1 = s_i, s_2, \dots, s_{2N+1})$. However, at this point s_{2N+1} is not necessarily equal to s_f . But $s_{2N+1} \in B(v(1))$ and $s_{2N} \in A(v(1))$. If $s_f \in B(v(1))$, we have $R(s_{2N}, s_f)$ and thus we can set $s_{2N+1} = s_f$, while if $s_f \in A(v(1))$ we have $R(s_{2N+1}, s_f)$, and thus we can add a new configuration $s_{2N+2} = s_f$. The sequence constructed is valid (we illustrate its construction in Fig. 3), and this concludes the demonstration of Theorem 3 as well as the demonstration of Theorem 1. \square

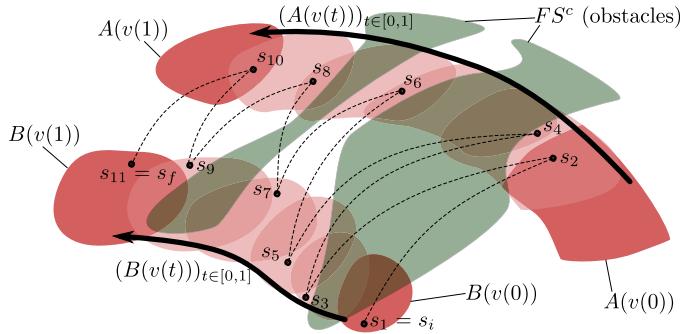


Fig. 3 From a valid continuous path to a valid sequence

What shows Theorem 1 is that a large range of d -dimensional discrete motion planning problems can be converted into equivalent $(2d + 1)$ -dimensional continuous motion planning problems. This result is interesting in itself, and might have a wide scope of potential applications, but because of the curse of dimensionality which particularly affects sampling-based motion planning algorithms, using such a conversion to actually solve discrete motion planning problems might be convenient but not be very efficient. In the next section, we show that with different assumptions on the relation $R()$, it is possible to obtain a similar equivalence without necessarily having to increase the dimensionality of the configuration space.

5 More Specific Reductions with Less Increase of Dimensionality

In this section, we relax the previous assumptions on $R()$. However, we assume that $R()$ is reflexive, and that there exists another metric configuration space Ω and a function $f : \Omega \rightarrow RegOp^*(CS)$ such that the four following properties are verified:

1. f is uniformly Γ -followable for some $\Gamma > 0$.
2. $\forall \varphi \in \Omega, f(\varphi)$ is such that $\forall (s, s') \in f(\varphi)^2, R(s, s')$.
3. $\forall (s, s') \in CS^2$ such that $R(s, s')$, there exists $\varphi \in \Omega$ such that $s \in f(\varphi)$ and $s' \in f(\varphi)$.
4. $\forall (s, s', s'') \in CS^3$ such that $R(s, s')$ and $R(s', s'')$, and $\forall \varphi_0 \in \Omega$ such that $s \in f(\varphi_0)$ and $s' \in f(\varphi_0)$, there exists a continuous path from φ_0 to a configuration φ_1 verifying $s'' \in f(\varphi_1)$, such that for any configuration φ along this path, we have $s' \in f(\varphi)$.

We define yet another notion of collision-freeness $C_\Omega()$:

Definition 5 ($C_\Omega()$). $\varphi \in \Omega$ verifies $C_\Omega(\varphi)$ if and only if the intersection between $f(\varphi)$ and the free space is non-empty, i.e. $\exists s \in f(\varphi)$ such that $C(s)$.

We have the following equivalence:

Theorem 4. *There exists a valid finite sequence from s_i to s_f in CS if and only if there exists a continuous path $(\chi(t))_{t \in [0,1]} \in \Omega^{[0,1]}$ such that $s_i \in f(\chi(0))$, $s_f \in f(\chi(1))$, and $\forall t \in [0, 1], C_\Omega(\chi(t))$.*

The next two sections are dedicated to the proof of each implication of this equivalence, while in sections 5.3 and 5.4, we study two examples of discrete motion planning problems where Theorem 4 applies.

5.1 From a Valid Discrete Sequence in CS to a Valid Continuous Path in Ω

Theorem 5. *If there exists a valid sequence $(s_1 = s_i, s_2, \dots, s_n = s_f)$, then there exists a valid continuous path $(\chi(t))_{t \in [0,1]}$ such that $s_1 \in f(\chi(0))$, $s_2 \in f(\chi(0))$, $s_{n-1} \in f(\chi(1))$ and $s_n \in f(\chi(1))$.*

Proof. We prove this implication by induction on n , the size of the valid sequence. For $n = 2$, we have $R(s_i, s_f)$, and there exists $\varphi \in \Omega$ such that $s_i \in f(\varphi)$ and $s_f \in f(\varphi)$. The stationary path such that $\forall t \in [0, 1], \chi(t) = \varphi$, is valid.

Let us now assume that the result is true for any sequence of size $n \geq 2$, and consider a valid sequence of size $n+1$: $(s_1 = s_i, s_2, \dots, s_{n+1} = s_f)$. Let $(\chi(t))_{t \in [0,1]} \in \Omega^{[0,1]}$ be a valid path such that $s_0 \in f(\chi(0))$, $s_1 \in f(\chi(0))$, $s_{n-1} \in f(\chi(1))$ and $s_n \in f(\chi(1))$. We have $R(s_{n-1}, s_n)$ and $R(s_n, s_{n+1})$, so there exists a continuous path $(\vartheta(t))_{t \in [0,1]} \in \Omega^{[0,1]}$ from $\chi(1)$ to a configuration $\varphi_f \in \Omega$ verifying $s_{n+1} \in f(\varphi_f)$, such that for any configuration ϑ along this path, we have $s_n \in f(\vartheta)$, and thus $C_\Omega(\vartheta)$. Appending this path to the path $(\chi(t))_{t \in [0,1]}$ gives us a valid continuous path, and this concludes the demonstration of Theorem 5. \square

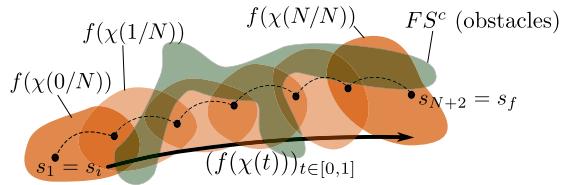
5.2 From a Valid Continuous Path in Ω to a Valid Discrete Sequence in CS

Theorem 6. *If there exists a valid continuous path $(\chi(t))_{t \in [0,1]} \in \Omega^{[0,1]}$ with $s_i \in \chi(0)$ and $s_f \in \chi(1)$, then there exists a valid sequence $(s_1 = s_i, s_2, \dots, s_n = s_f)$.*

Proof. The demonstration is a bit similar to the one of Theorem 3, and we give here only a sketch of the proof. Using the fact that f is uniformly Γ -followable, we can show that for any $\varepsilon > 0$, there exists $N \in \mathbb{N}^*$ such that the sequence $(\chi(0/N), \chi(1/N), \dots, \chi(N/N))$ verifies the following property: $\forall k \in \{0, \dots, N-1\}$, for any configuration $s \in f(\chi(k/N))$, the sphere $\mathcal{S}(s, \varepsilon)$ intersects the non-empty set $f(\chi(k/N)) \cap f(\chi((k+1)/N))$. For ε small enough, we can show that $\forall t \in [0, 1]$, there exists $s_t \in f(\chi(t))$ collision-free and such that the sphere $\mathcal{S}(s_t, \varepsilon)$ is entirely inside FS . We thus deduce that all the sets $f(\chi(k/N)) \cap f(\chi((k+1)/N))$ have a non-empty intersection with FS . Using the property that two elements s , s' of the same set $f(\chi(t))$ are always such that $R(s, s')$, we can construct a valid

sequence $(s_1, s_2, \dots, s_{N+2})$ such that $s_1 = s_i$, $s_{N+2} = s_f$, and $\forall k \in \{2, \dots, N+1\}$, we have $s_k \in f(\chi((k-2)/N)) \cap f(\chi((k-1)/N))$. Such a construction is illustrated in Fig. 4, and it concludes the demonstration of Theorem 6. \square

Fig. 4 From a valid continuous path $(\chi(t))_{t \in [0,1]} \in \Omega^{[0,1]}$ to a valid finite sequence of configurations in CS



5.3 Flea Motion Planning

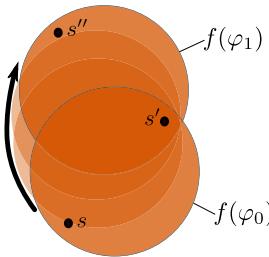
First, let us consider the simple example of flea motion planning, which has been introduced in [21] to illustrate the method used in [22] and [21] to convert footstep planning into classical continuous motion planning. Compared to the version presented in [21], we make some small modifications in the definition of the problem so as to fit the premises of our theorems (which could also be slightly modified in order to directly apply to the version of [21]).

The flea is a point in a 2D environment, and thus we use $CS = \mathbb{R}^2$ as the configuration space. There are obstacles in this 2D environment such that the free space $FS = \{s \in \mathbb{R}^2 \mid C(s)\}$ is an open set. The flea can make jumps in any direction and of any length strictly less than $l_{max} > 0$. The goal is to find a sequence of jumps from a location (x_A, y_A) to a location (x_B, y_B) while avoiding the obstacles. This problem clearly fits our definition of discrete motion planning problem, with, for $(s, s') \in (\mathbb{R}^2)^2$, $R(s, s') \Leftrightarrow dist(s, s') < l_{max}$. This relation $R()$ is reflexive. Let us consider the function $f : \mathbb{R}^2 \rightarrow RegOp^*(\mathbb{R}^2)$ such that $f(s) = \mathcal{S}(s, \frac{l_{max}}{2})$. It can be verified that f is uniformly $2l_{max}$ -followable. Besides, the other properties required by Theorem 4 are also verified:

- $\forall s \in \mathbb{R}^2$, any two points s', s'' in $f(s)$ are such that $dist(s', s'') < l_{max}$, and therefore we have $R(s', s'')$.
- For any two points s, s' verifying $R(s, s')$, there exists an open disk of radius $\frac{l_{max}}{2}$ containing both points, and thus there exists $s'' \in \mathbb{R}^2$ such that $s \in f(s'')$ and $s' \in f(s'')$.
- As explained in Fig. 5 the fourth property required for Theorem 4 to apply is also verified.

As a consequence of these properties, we can apply Theorem 4. This means that trying to solve the flea motion planning problem is equivalent to trying to find a valid continuous path for the disk of radius $\frac{l_{max}}{2}$. It turns out that this equivalence gives a very efficient algorithm to solve the flea motion planning problem. Indeed, the new notion of collision-freeness for the disk is the following one: a configuration of the disk is collision-free if and only if there exists a point inside the disk that is outside the obstacles. This new notion of collision-freeness is called “weakly

Fig. 5 For any three points s, s', s'' verifying $R(s, s')$ and $R(s', s'')$, and any $\varphi_0 \in \mathbb{R}^2$ such that $s \in f(\varphi_0)$ and $s' \in f(\varphi_0)$, there exists a continuous path from φ_0 to a configuration φ_1 verifying $s'' \in f(\varphi_1)$, and such that any configuration φ along this path verifies $s' \in f(\varphi)$



collision-freeness” in [21]. To check this property, we can apply to the obstacles a morphological operation of erosion by an open sphere of radius $\frac{l_{max}}{2}$ (see [23]), and the collision-freeness of the disk becomes equivalent to the classical collision-freeness of the center of the disk in the environment with the eroded obstacles. So, once the eroded obstacles are obtained, we can use any classical sampling-based algorithm to find a short continuous collision-free path for the disk. To actually convert this continuous path to a finite sequence of jumps, we can apply the greedy approach already used in [21] which consists in repeatedly trying to jump from the current disk $f(\varphi(t))$ to a disk $f(\varphi(t'))$ with t' as large as possible and obtained by dichotomy.

Let us add a few comments on the problem of flea motion planning. Firstly, the flea motion planning problem can be extended to \mathbb{R}^d for $d > 2$ (the $R(s)$ sets become d -dimensional spheres), and Theorem 4 still applies. Secondly, the conditions of Theorem 1 are also verified. Therefore, we could have used Theorem 1 to convert flea motion planning into a continuous motion planning problem in a configuration space with $2 \times 2 + 1 = 5$ dimensions. But, because with flea motion planning the relation $R()$ is geometrically simple, it was possible to use a similar equivalence to convert it into a 2-dimensional continuous motion planning problem. A question of prime importance is raised by this remark: for a given discrete motion planning problem, how can we simplify or approximate the relation R so as to obtain an equivalent continuous motion planning problem in a configuration space with as few dimensions as possible? We show an example of such simplification in the next section.

5.4 A Variant of the Flea Motion Planning Problem

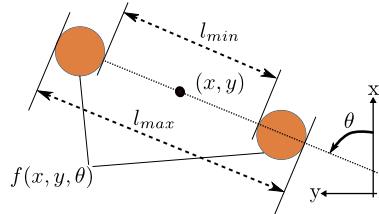
Let us first consider a variant of the flea motion planning problem where the flea cannot make jumps smaller than some fixed length. The relation $R()$ becomes: $R(s, s') \Leftrightarrow l_{min} < dist(s, s') < l_{max}$. With this variant, $R()$ is not reflexive and Theorem 4 cannot be applied. In order not to have to apply Theorem 1 and do the continuous motion planning in a configuration space of dimension 5, we modify a bit the relation $R()$. We pose:

$$R(s, s') \Leftrightarrow l_{min} < dist(s, s') < l_{max} \vee dist(s, s') < l_{max} - l_{min}$$

It still seems difficult to apply Theorem 4 with a new configuration space Ω of dimension 2 like we just did for flea motion planning, and we leave it as an open question. However, we show that we can do it with a space of dimension 3.

Let us consider the function $f : SE(2) \rightarrow RegOp^*(\mathbb{R}^2)$ as described in Fig. 6. With this function, the properties required by Theorem 4 are verified (we do not demonstrate it here), and so the theorem applies. This means that trying to find a sequence of jumps for this new variant of the flea motion planning problem is equivalent to trying to find a continuous collision-free path in $SE(2)$. The same greedy method as for flea motion planning can be applied. So, we have just seen that a slight modification of $R()$ enabled us to use Theorem 4 and obtain an equivalent continuous motion planning problem in a configuration space with 3 dimensions rather than 5 with Theorem 1. This can be interesting, but depending on the problem, such a simplification might not make sense, especially if like here it increases the motion capabilities. In general, it is better to look for simplifications that reduce the motion capabilities, leading to conservative approaches. Another remark of importance is that short paths in the continuous space are not necessarily converted into short sequences of jumps, even with the greedy approach. In practice, it seems more likely to be true with the standard flea motion planning problem. It would be interesting to investigate under which circumstances it is always possible to convert short continuous paths into short finite sequences, or more precisely to try to transfer bounds of sub-optimality from the continuous paths to the resulting finite sequences, but this is out of the scope of this paper.

Fig. 6 $f(x, y, \theta)$ is the union of two open disk of radius $\frac{l_{\max} - l_{\min}}{2}$, the first one of center $(x, y) + \frac{l_{\max} - l_{\min}}{4}(\cos \theta, \sin \theta)$, and the second one of center $(x, y) - \frac{l_{\max} - l_{\min}}{4}(\cos \theta, \sin \theta)$.



6 Applications

The technique used in [22] and [21] for footstep planning for humanoid robots can be seen as an application of Theorem 4. In the present paper, we show that it is easy to extend the method in order to plan the walking motion of a hexapod robot (cf. Fig. 7). Our objective is to make the hexapod walk on uneven terrain with non-gaited locomotion planning (which is typically computationally costly). The uneven terrain is described by a heightmap, i.e. a function $z = F(x, y)$. The heightmap can be used to set the height of the contact positions, and we ignore the contact orientations, so we use $(\mathbb{R}^2)^6$ as the configuration space (it is easy to define a heuristic that sets a unique whole-body configuration from the 6 contact positions; in particular, we require the robot main body to remain horizontal). Here is how we define the relation $C()$: from the heightmap we infer what locations are allowed for individual contacts,

and for a configuration in $(\mathbb{R}^2)^6$, we require our heuristic to lead to a valid whole-body configuration that does not collide with the heightmap.

We try to apply Theorem 4 with the configuration space $\Omega = SE(2)$. To do so, we simplify the walking capabilities of the hexapod. For a configuration (x, y, θ) , we define $f(x, y, \theta)$ as the set of configurations in $(\mathbb{R}^2)^6$ such that each contact belongs to an open disk, as shown in Fig. 7 (it is important that the disks are disjoints). With this assumption, we first replace $C()$ by heuristic checks: for a configuration $s \in f(x, y, \theta)$, $C(s)$ is verified if the contacts are safe (i.e. the heightmap is almost flat around their locations), if the maximum height difference between two contacts with the ground is less than some threshold, and if the maximum height of the heightmap in the “robot zone” (i.e. the convex hull of the contacts) is not much higher than the height of the contacts. Here is how we simplify the walking capabilities of the hexapod: we consider transitions where all the 6 legs are moved at the same time, and we say that a transition from $s \in (\mathbb{R}^2)^6$ to $s' \in (\mathbb{R}^2)^6$ is allowed if and only if there exists $(x, y, \theta) \in SE(2)$ such that $s \in f(x, y, \theta)$ and $s' \in f(x, y, \theta)$. With this restriction, we can verify that the conditions of Theorem 4 apply (only the fourth property is difficult to verify), and thus we can use the equivalence to convert our problem of locomotion planning into a continuous motion planning in $SE(2)$ (we use the library OMPL and the algorithm RRT-Connect [14] to perform the motion planning). Once the conversion of a continuous path is done, we obtain a finite sequence of transitions for which the 6 foot locations are changed at each transition. It is not difficult to convert it into a sequence of feasible transitions where at most 3 feet are moved at the same time (but sometimes 2, or just 1).

This original technique for legged locomotion planning is convenient and fast: in the example described in Fig. 7 where the hexapod must go across an uneven and challenging terrain, the whole planning (continuous planning *and* two-stage conversion into a discrete sequence of steps) was done in 57ms on an Intel(R) Core(TM) i7 1.60GHz CPU. We cannot readily use this method to solve planning problems as complex as the ones considered in [8], but it is a good compromise between gaited methods and more complex approaches such as [8]. It would be interesting to try to make an advantageous use of our method in advanced software architectures for multi-contact motion planning, such as the ones presented in [10] or [24].

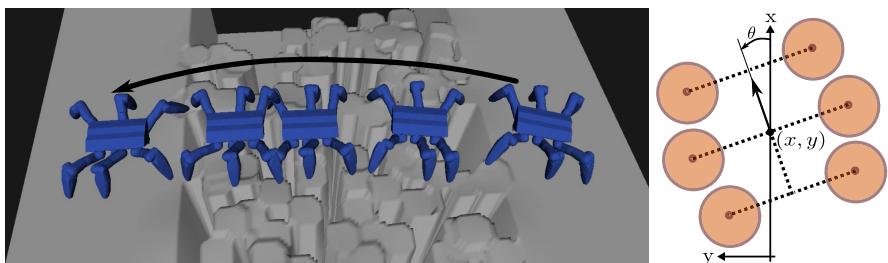


Fig. 7 On the left: the motion of the hexapod across this challenging terrain was planned in 57ms. On the right: the six open disks that constrain the configurations and steps of the hexapod (each leg must have its contact with the ground within its assigned disk).

7 Conclusion

In this paper we have proved two new equivalence results between discrete and continuous motion planning. They can be used to convert discrete problems into continuous ones that are similar to the piano mover's problem, and thus enable the application of standard motion planning algorithms to a new class of problems. We have shown that it leads to original and efficient techniques for legged locomotion planning, and expect various other types of applications such as for example regrasp planning. In future work, it would be interesting to study more precisely the complexity of the algorithms made possible by our approach, and in particular the complexity of the new collision checks, an issue that we did not address in the present paper. Another question of importance is whether a similar equivalence could be obtained with motion planning problems with kinodynamic constraints, which is not obvious at all at this stage.

References

1. Alami, R., Laumond, J.-P., Siméon, T.: Two manipulation planning algorithms. In: 1st Workshop on the Algorithmic Foundations of Robotics, WAFR 1994 (1994)
2. Bouyarmane, K., Kheddar, A.: Multi-contact stances planning for multiple agents. In: IEEE Int. Conf. on Robotics and Automation (ICRA 2011), pp. 5246–5253 (2011)
3. Bullo, F., Leonard, N.E., Lewis, A.D.: Controllability and motion algorithms for underactuated lagrangian systems on lie groups. *IEEE Transactions on Automatic Control* 45(8), 1437–1454 (2000)
4. Chestnutt, J., Lau, M., Cheung, G., Kuffner, J.J., Hodgins, J., Kanade, T.: Footstep planning for the honda asimo humanoid. In: IEEE Int. Conf. on Robotics and Automation (ICRA 2005), pp. 631–636 (2005)
5. Dalibard, S., El Khoury, A., Lamiraux, F., Taix, M., Laumond, J.-P.: Small-space controllability of a walking humanoid robot. In: IEEE/RAS Int. Conf. on Humanoid Robots, Humanoids 2011 (2011)
6. Ferré, E., Laumond, J.-P.: An iterative diffusion algorithm for part disassembly. In: IEEE Int. Conf. on Robotics and Automation, ICRA 2004 (2004)
7. Geraerts, R., Overmars, M.H.: Creating high-quality paths for motion planning. *I. J. Robotic Res.* 26, 845–863 (2007)
8. Hauser, K., Bretl, T., Latombe, J.-C., Wilcox, B.: Motion Planning for a Six-Legged Lunar Robot. In: Akella, S., Amato, N.M., Huang, W.H., Mishra, B. (eds.) *Algorithmic Foundations of Robotics VII*. STAR, vol. 47, pp. 301–316. Springer, Heidelberg (2008)
9. Hauser, K., Latombe, J.-C.: Multi-modal motion planning in non-expansive spaces. *I. J. Robotic Res.* 29(7), 897–915 (2010)
10. Kalakrishnan, M., Buchli, J., Pastor, P., Mistry, M., Schaal, S.: Learning, planning, and control for quadruped locomotion over challenging terrain. *I. J. Robotic Res.* 30(2) (2011)
11. Kanoun, O., Yoshida, E., Laumond, J.-P.: An optimization formulation for footsteps planning. In: IEEE/RAS Int. Conf. on Humanoid Robots, Humanoids 2009 (2009)
12. Karaman, S., Frazzoli, E.: Incremental sampling-based algorithms for optimal motion planning. In: *Robotics Science and Systems VI* (2010)

13. Kavraki, L.E., Svestka, P., Latombe, J.-C., Overmars, M.H.: Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. on Robotics and Automation* 12, 566–580 (1996)
14. Kuffner, J.J., LaValle, S.M.: RRT-Connect: An efficient approach to single-query path planning. In: *IEEE Int. Conf. on Robotics and Automation (ICRA 2000)*, pp. 995–1001 (2000)
15. Kuffner, J.J., Nishiwaki, K., Kagami, S., Inaba, M., Inoue, H.: Footstep planning among obstacles for biped robots. In: *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS 2001)*, pp. 500–505 (2001)
16. Lafferriere, G., Sussmann, H.J.: Motion planning for controllable systems without drift. In: *IEEE Int. Conf. on Robotics and Automation (ICRA 1991)*, pp. 1148–1153 (1991)
17. LaValle, S.M., Kuffner, J.J.: Rapidly-exploring random trees: Progress and prospects. In: *4th Workshop on the Algorithmic Foundations of Robotics (WAFR 2000)*, pp. 293–308 (2000)
18. The Open Motion Planning Library (2010), <http://omp1.kavrakilab.org>
19. Pan, J., Lauterbach, C., Manocha, D.: g-Planner: Real-time motion planning and global navigation using GPUs. In: *24th AAAI Conf. on Artificial Intelligence* (2010)
20. Pan, J., Liangjun, Z., Manocha, D.: Collision-free and curvature-continuous path smoothing in cluttered environments. In: *Robotics: Science and Systems, RSS 2011* (2011)
21. Perrin, N., Stasse, O., Lamiraux, F., Kim, Y.J., Manocha, D.: Real-time footstep planning for humanoid robots among 3d obstacles using a hybrid bounding box. In: *IEEE Int. Conf. on Robotics and Automation, ICRA 2012* (2012)
22. Perrin, N., Stasse, O., Lamiraux, F., Yoshida, E.: Weakly collision-free paths for continuous humanoid footstep planning. In: *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS 2011)*, pp. 4408–4413 (2011)
23. Serra, J.: *Image Analysis and Mathematical Morphology*. Academic Press, Inc. (1983)
24. Zucker, M., Andrew, J., Christopher, B., Atkeson, G., Kuffner, J.J.: An optimization approach to rough terrain locomotion. In: *IEEE Int. Conf. on Robotics and Automation, ICRA 2010* (2010)

Ray-Shooting Algorithms for Robotics

Yu Zheng and Katsu Yamane

Abstract. Ray shooting is a well-studied problem in computer graphics. It also occurs in robotics as a collision detection problem in 3-D object space or a contact force optimization problem in 6-D wrench space. However, the ray-shooting algorithms derived in computer graphics are limited to 3-D polyhedra and not suited for general convex sets in high-dimensional space. This paper discusses several general ray-shooting algorithms and their applications to these problems in robotics.

1 Introduction

In computational geometry and computer graphics, the ray-shooting problem deals with computing the first intersection point on the surface of given objects by a query ray and has been well studied for over four decades [1, 8, 13, 14]. In these efforts, most of the practical algorithms are limited to 2-D or 3-D polytopes.

It has been discovered that several fundamental problems in robotics are equivalent to a ray-shooting problem. Ong and Gilbert [10] proposed a unified distance measure, called the growth distance, for both separated and penetrated objects and applied it to collision-free path planning [11]. The growth distance computation can be reduced to a ray-shooting problem. Liu et al. [6, 2, 7] cast several problems in grasping, such as grasping force optimization and force-closure grasp test and synthesis, into a ray-shooting problem. However, these ray-shooting problems deal with the intersection of a ray with a set, which is specified by nonlinear parametric functions combined with complex operations on sets, such as the Minkowski sum, rather than a 3-D polytope with given vertices or facets. Some of them are also in higher-dimensional space than 3-D. Therefore, the ray-shooting techniques in computational geometry and computer graphics are not suited here. The previous solutions to these problems relied on general-purpose optimization techniques

Yu Zheng · Katsu Yamane

Disney Research Pittsburgh, USA

e-mail: {yu.zheng, kyamane}@disneyresearch.com

[10, 6] and suffered the low computational efficiency. Recently, two procedures were proposed to more quickly solve such a ray-shooting problem [16, 18].

In this paper, we extensively discuss the algorithms for a general ray-shooting problem and their applications in robotics. The original contributions include:

- A better stopping criterion for the algorithm [16] to enhance its accuracy.
- Generalization of the algorithm [18] to any case with guaranteed convergence.
- Discussion on a new algorithm and hybrid uses of these algorithms to gain higher computational efficiency.
- Application of these ray-shooting algorithms to growth distance computation, in addition to contact force optimization in the previous work [16, 18].

The rest of this paper is organized as follows. Sect. 2 defines the ray-shooting problem and summarizes algorithms to be used in developing ray-shooting algorithms in Sect. 3. Sects. 4 and 5 show their applications with numerical examples. Conclusions and future work are included in Sect. 6. In the following discussion, we will use many convex geometry concepts, for which we refer readers to [4].

2 Problem Statement and Preliminaries

In this section, we give a mathematical definition of the ray-shooting problem and summarize two existing algorithms for minimum distances [3, 15].

2.1 Definition of the Ray-Shooting Problem

Let A be a compact convex set with nonempty interior in \mathbb{R}^n , \mathbf{r} a nonzero vector in \mathbb{R}^n , and $R(\mathbf{r})$ the ray emanating from the origin $\mathbf{0}$ of \mathbb{R}^n in the direction \mathbf{r} , i.e.,

$$R(\mathbf{r}) \triangleq \{\lambda \mathbf{r} \in \mathbb{R}^n \mid \lambda \geq 0\}. \quad (1)$$

The ray-shooting problem first needs to determine

1. whether A and $R(\mathbf{r})$ intersect.

If A and $R(\mathbf{r})$ intersect, then it is also aimed at computing

2. the farthest intersection point $\mathbf{z}_A(\mathbf{r})$ of A with $R(\mathbf{r})$ from the origin $\mathbf{0}$;
3. a set of affinely independent points in A , denoted by $Z_A(\mathbf{r})$, such that $\mathbf{z}_A(\mathbf{r})$ can be written as a convex combination of $Z_A(\mathbf{r})$;
4. the normal \mathbf{n} of the hyperplane that passes through $\mathbf{z}_A(\mathbf{r})$ and supports A .

In the algorithms discussed later to solve the ray-shooting problem, the support function h_A and the support mapping s_A of A are often used, which are defined by

$$h_A(\mathbf{u}) \triangleq \max_{\mathbf{a} \in A} \mathbf{u}^T \mathbf{a}, \quad s_A(\mathbf{u}) \triangleq \arg \max_{\mathbf{a} \in A} \mathbf{u}^T \mathbf{a}. \quad (2)$$

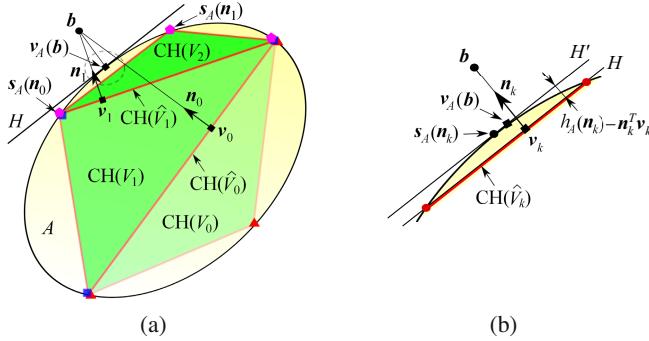


Fig. 1 Illustration of the GJK algorithm in 2-D space. (a) Iteration that leads \mathbf{v}_k to $\mathbf{v}_A(\mathbf{b})$. (b) Stopping criterion $h_A(\mathbf{n}_k) - \mathbf{n}_k^T \mathbf{v}_k < \varepsilon_{\text{GJK}}$. H and H' are hyperplanes with normal \mathbf{n}_k passing \mathbf{v}_k and $\mathbf{s}_A(\mathbf{n}_k)$, respectively, and bounds $\mathbf{v}_A(\mathbf{b})$ in the gap between them with the width of $h_A(\mathbf{n}_k) - \mathbf{n}_k^T \mathbf{v}_k$.

2.2 Summary of the GJK Algorithm [3]

Let \mathbf{b} be an arbitrary point in \mathbb{R}^n . The purpose of this algorithm is to compute the distance $d_A(\mathbf{b})$ between \mathbf{b} and A and the closest point $\mathbf{v}_A(\mathbf{b})$ in A to \mathbf{b} , defined by

$$d_A(\mathbf{b}) \triangleq \min_{\mathbf{a} \in A} \|\mathbf{a} - \mathbf{b}\|, \quad \mathbf{v}_A(\mathbf{b}) \triangleq \arg \min_{\mathbf{a} \in A} \|\mathbf{a} - \mathbf{b}\|. \quad (3)$$

Since A is compact and convex, $\mathbf{v}_A(\mathbf{b})$ exists and is unique. If $\mathbf{b} \notin A$, then $d_A(\mathbf{b}) > 0$ and the hyperplane H with normal $\mathbf{b} - \mathbf{v}_A(\mathbf{b})$ passing through $\mathbf{v}_A(\mathbf{b})$ supports A at $\mathbf{v}_A(\mathbf{b})$, as depicted in Fig. 1a; otherwise $d_A(\mathbf{b}) = 0$ and $\mathbf{v}_A(\mathbf{b}) = \mathbf{b}$.

Both $d_A(\mathbf{b})$ and $\mathbf{v}_A(\mathbf{b})$ can be computed by the GJK algorithm, as illustrated in Fig. 1. It starts with any affinely independent set V_0 in A and iterates by $V_{k+1} = \hat{V}_k \cup \{\mathbf{s}_A(\mathbf{n}_k)\}$, where \hat{V}_k is a minimal subset of V_k to represent \mathbf{v}_k (the closest point in $\text{CH}(V_k)$ to \mathbf{b}) as its convex combination and \mathbf{n}_k is the unit vector from \mathbf{v}_k to \mathbf{b} . If $h_A(\mathbf{n}_k) > \mathbf{n}_k^T \mathbf{v}_k$, then $d_{\text{CH}(V_{k+1})}(\mathbf{b}) < d_{\text{CH}(V_k)}(\mathbf{b})$. Hence, $d_{\text{CH}(V_k)}(\mathbf{b})$ is strictly decreasing with the iteration and converges to $d_A(\mathbf{b})$ while $h_A(\mathbf{n}_k) - \mathbf{n}_k^T \mathbf{v}_k < \varepsilon_{\text{GJK}}$, where ε_{GJK} is the termination tolerance. Then, \mathbf{v}_k converges to $\mathbf{v}_A(\mathbf{b})$ and \hat{V}_k gives an affinely independent set in A , denoted by $V_A(\mathbf{b})$, to represent $\mathbf{v}_A(\mathbf{b})$ as its convex combination. If $\mathbf{b} \in A$, then $\mathbf{v}_A(\mathbf{b}) = \mathbf{b}$ and \mathbf{b} is a convex combination of $V_A(\mathbf{b})$.

2.3 Summary of the ZC Algorithm [15]

Let $\text{CO}(A)$ denote the convex cone of A [4], which consists of all nonnegative combinations of A . This algorithm computes the distance $d_{\text{CO}(A)}(\mathbf{b})$ and the closest point $\mathbf{v}_{\text{CO}(A)}(\mathbf{b})$ between \mathbf{b} and $\text{CO}(A)$, defined similarly to (3) with A replaced by $\text{CO}(A)$, as illustrated in Fig. 2. It starts with a linearly independent set V_0 in A and iterates by $V_{k+1} = \hat{V}_k \cup \{\mathbf{s}_A(\mathbf{n}_k)\}$, where \hat{V}_k is a minimal subset of V_k to represent \mathbf{v}_k (the

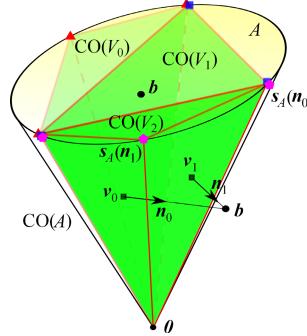


Fig. 2 Illustration of the ZC algorithm in 3-D space. If $\mathbf{b} \in \text{CO}(A)$, then a linearly independent subset of A , namely V_1 here, is obtained to contain \mathbf{b} in its convex cone.

closest point in $\text{CO}(V_k)$ to \mathbf{b}) as its positive combination and \mathbf{n}_k is the unit vector from \mathbf{v}_k to \mathbf{b} . This iteration leads $d_{\text{CO}(V_k)}(\mathbf{b})$ to $d_{\text{CO}(A)}(\mathbf{b})$ and \mathbf{v}_k to $\mathbf{v}_{\text{CO}(A)}(\mathbf{b})$ while $h_A(\mathbf{n}_k) < \varepsilon_{\text{ZC}}$, where ε_{ZC} is the termination tolerance. The final \hat{V}_k provides a linearly independent set in A , denoted by $V_{\text{CO}(A)}(\mathbf{b})$, to represent $\mathbf{v}_{\text{CO}(A)}(\mathbf{b})$ as its positive combination. If $\mathbf{b} \in \text{CO}(A)$, then $d_{\text{CO}(A)}(\mathbf{b}) = 0$ and $\mathbf{v}_{\text{CO}(A)}(\mathbf{b}) = \mathbf{b}$, and \mathbf{b} is a positive combination of $V_{\text{CO}(A)}(\mathbf{b})$.

3 Ray-Shooting Algorithms

In this section, we discuss procedures to solve the ray-shooting problem and their hybrid uses for higher efficiency. In general, each procedure generates a sequence of points on $R(\mathbf{r})$ approaching $\mathbf{z}_A(\mathbf{r})$. Hence, the convergence of these procedures can be easily proved by using the monotone-convergence principle [12].

3.1 A GJK-Based Procedure

A procedure based on the GJK algorithm to solve the ray-shooting problem was proposed in [14, 16, 17]. It computes $\mathbf{z}_A(\mathbf{r})$ by iterating a point \mathbf{b}_k on $R(\mathbf{r})$ outside of A , as depicted in Fig. 3a. Initially, \mathbf{b}_0 can be taken to be $h_A(\mathbf{r})\mathbf{r}/\mathbf{r}^T\mathbf{r}$, which is the intersection point of the supporting hyperplane $H_0 = H(\mathbf{r}, \mathbf{s}_A(\mathbf{r}))$ of A with $R(\mathbf{r})$ and not in the interior of A . Then, we update the point \mathbf{b}_k by

$$\mathbf{b}_{k+1} = \mathbf{b}_k - \frac{d_A(\mathbf{b}_k)}{\mathbf{r}^T \mathbf{n}_k} \mathbf{r} \quad (4)$$

where $d_A(\mathbf{b}_k)$ and $\mathbf{v}_A(\mathbf{b}_k)$ are the minimum distance and the closest point from A to \mathbf{b}_k , respectively, which are computed by the GJK algorithm, and \mathbf{n}_k is the unit vector from $\mathbf{v}_A(\mathbf{b}_k)$ to \mathbf{b}_k . The iteration (4) can be terminated with the conclusion that $R(\mathbf{r})$

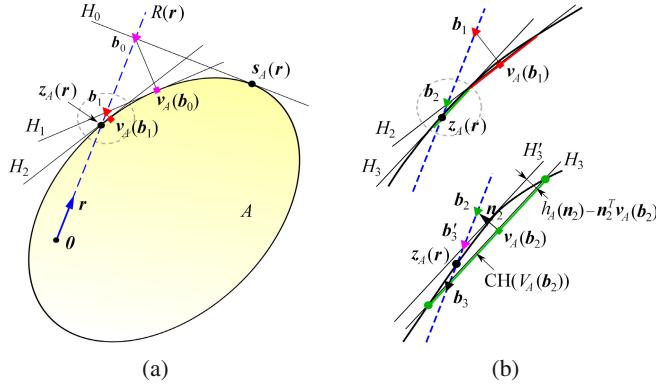


Fig. 3 Illustration of the GJK-based ray-shooting algorithm in 2-D space. (a) Iteration. \mathbf{b}_0 is the intersection point of H_0 with $R(\mathbf{r})$. H_1 is the hyperplane with normal $\mathbf{b}_0 - \mathbf{v}_A(\mathbf{b}_0)$ passing through $\mathbf{v}_A(\mathbf{b}_0)$ and intersects $R(\mathbf{r})$ at \mathbf{b}_1 , which is closer to $\mathbf{z}_A(\mathbf{r})$ than \mathbf{b}_0 . (b) Stopping criterion. Upper: Although $d_A(\mathbf{b}_1)$ is small, this does not guarantee that \mathbf{b}_1 is close enough to and can be taken to be $\mathbf{z}_A(\mathbf{r})$. Lower: A better stopping criterion is that $R(\mathbf{r})$ intersects $\text{CH}(V_A(\mathbf{b}_2))$ (the green line segment). Then, their intersection point \mathbf{b}_3 can be adopted as $\mathbf{z}_A(\mathbf{r})$ and $V_A(\mathbf{b}_2)$ as $Z_A(\mathbf{r})$.

does not intersect A if $\mathbf{r}^T \mathbf{b}_k < \max\{-h_A(-\mathbf{r}), 0\}$, or $\mathbf{r}^T \mathbf{n}_k = 0$ while $d_A(\mathbf{b}_k) \neq 0$, or $\mathbf{r}^T \mathbf{n}_k < 0$ [17]. Or else, $d_A(\mathbf{b}_k)$ will decrease to zero and \mathbf{b}_k will approach $\mathbf{z}_A(\mathbf{r})$.

Instead of the stopping criterion $d_A(\mathbf{b}_k) < \varepsilon$ used in [14, 16, 17], we verify whether $R(\mathbf{r})$ intersects $\text{CH}(V_A(\mathbf{b}_k))$ or \mathbf{r} is a nonnegative combination of $V_A(\mathbf{b}_k)$, as explained in Fig. 3b. If so, we can compute their intersection point, which is equal to \mathbf{b}_{k+1} from (4) and contained in A , since $\text{CH}(V_A(\mathbf{b}_k))$ is contained in the hyperplane $H_{k+1} = H(\mathbf{n}_k, \mathbf{v}_A(\mathbf{b}_k))$. The hyperplane $H'_{k+1} = H(\mathbf{n}_k, \mathbf{s}_A(\mathbf{n}_k))$ supports A and intersects $R(\mathbf{r})$ at the point $\mathbf{b}'_{k+1} = h_A(\mathbf{n}_k)\mathbf{r}/\mathbf{r}^T \mathbf{n}_k$, which is outside of A . It is evident that $\mathbf{z}_A(\mathbf{r})$ is between \mathbf{b}'_{k+1} and \mathbf{b}_{k+1} . The GJK algorithm stops iterating when $h_A(\mathbf{n}_k) - \mathbf{n}_k^T \mathbf{v}_A(\mathbf{b}_k) < \varepsilon_{\text{GJK}}$, which implies $\mathbf{n}_k^T (\mathbf{b}'_{k+1} - \mathbf{b}_{k+1}) < \varepsilon_{\text{GJK}}$. From this we can derive $\|\mathbf{z}_A(\mathbf{r}) - \mathbf{b}_{k+1}\| \leq \|\mathbf{b}'_{k+1} - \mathbf{b}_{k+1}\| < \varepsilon_{\text{GJK}} \|\mathbf{r}\|/\mathbf{r}^T \mathbf{n}_k$. Therefore, we take $\mathbf{z}_A(\mathbf{r}) = \mathbf{b}_{k+1}$ and $Z_A(\mathbf{r}) = V_A(\mathbf{b}_k)$, as depicted in the lower figure of Fig. 3b.

3.2 An Internal Expanding (IE) Procedure

A preliminary version of this procedure was proposed in [18]. However, its convergence was unclear in some rare singular situations. Here we solve this issue and present a generalized version with guaranteed convergence.

Assume that we have a facet F_0 in A such that $R(\mathbf{r})$ passes through its interior. Let \mathfrak{F}_k be a collection of facets in A and initially $\mathfrak{F}_0 = \{F_0\}$. In the following iteration, \mathfrak{F}_k may comprise more than one facet, but $R(\mathbf{r})$ intersects every facet in \mathfrak{F}_k at the same point, which is on the face $\text{CH}(S_k)$ shared by those facets and denoted by \mathbf{b}_k , where S_k is the set of vertices of the face. Let \mathbf{n}_i be the unit normal to each facet

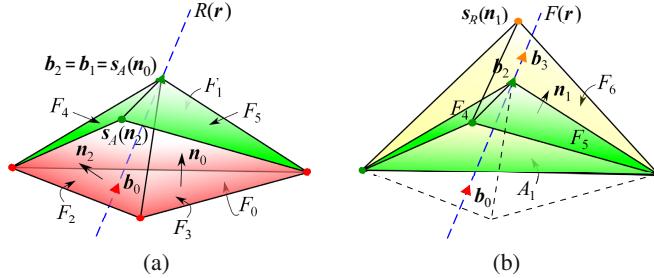


Fig. 4 Illustration of the IE ray-shooting algorithm in 3-D space. (a) $\mathfrak{F}_0 = \{F_0\}$ and $R(\mathbf{r})$ passes through an interior point \mathbf{b}_0 of F_0 . Then, $\mathfrak{F}_0^+ = \mathfrak{F}_0$. Assume that $R(\mathbf{r})$ happens to pass $s_A(\mathbf{n}_0)$. Then, $\mathfrak{F}_1 = \{F_1, F_2, F_3\}$. Suppose that F_2 is the facet in \mathfrak{F}_1 closest to the origin and $s_A(\mathbf{n}_2)$ lies on the different side of F_2 or F_3 from the origin. Then, $\mathfrak{F}_1^+ = \{F_2, F_3\}$ and F_4 and F_5 are generated to replace F_2 and F_3 . As a result, $\mathfrak{F}_2 = \{F_1, F_4, F_5\}$. (b) Suppose that F_1 is the facet in \mathfrak{F}_2 closest to the origin and $s_A(\mathbf{n}_1)$ lies on the different side of any facet in \mathfrak{F}_2 from the origin. Then, $R(\mathbf{r})$ intersects a new face, i.e., F_6 , at a point \mathbf{b}_3 closer to $\mathbf{z}_A(\mathbf{r})$ than \mathbf{b}_2 . As \mathbf{b}_3 is in the interior of F_6 , $\mathfrak{F}_3 = \{F_6\}$.

in \mathfrak{F}_k such that $\mathbf{n}_i^T \mathbf{r} > 0$. Then $\mathbf{n}_i^T \mathbf{b}_k$ is the distance from the origin to facet i in \mathfrak{F}_k . We find facet i^* in \mathfrak{F}_k whose distance from the origin is the minimum and compute $h_A(\mathbf{n}_{i^*})$ and $s_A(\mathbf{n}_{i^*})$. If $h_A(\mathbf{n}_{i^*}) - \mathbf{n}_{i^*}^T \mathbf{b}_k > \varepsilon$, we extract a sub-collection \mathfrak{F}_k^+ from \mathfrak{F}_k such that $\mathbf{n}_i^T s_A(\mathbf{n}_{i^*}) > \mathbf{n}_{i^*}^T \mathbf{b}_k$ for any facet in \mathfrak{F}_k^+ , which implies that $s_A(\mathbf{n}_{i^*})$ lies on the different side of the facet from the origin. Let \mathfrak{R} be the collection of facets of all facets in \mathfrak{F}_k^+ , which are ridges in \mathbb{R}^n , and \mathfrak{R}_i the collection of facets of a facet in \mathfrak{F}_k^+ that contain $\text{CH}(S_k)$. Then, we will meet two situations, as depicted in Fig. 4a:

1. \mathfrak{F}_k^+ is equal to \mathfrak{F}_k : In this case, $s_A(\mathbf{n}_{i^*})$ lies on the different side of every facet in \mathfrak{F}_k from the origin. Then, $R(\mathbf{r})$ intersects one of the facets formed by each ridge in $\mathfrak{R} \setminus \cup_i \mathfrak{R}_i$ with the point $s_A(\mathbf{n}_{i^*})$ and the intersection point, which is assigned to \mathbf{b}_{k+1} , is farther from the origin than \mathbf{b}_k . Hence, we reconstruct the facet collection \mathfrak{F}_{k+1} with the facets intersected by $R(\mathbf{r})$ and the ridge collections \mathfrak{R}_i for each facet in \mathfrak{F}_{k+1} accordingly. Note that $R(\mathbf{r})$ may pass through a common face of some of the newly formed facets. In that case, \mathfrak{F}_{k+1} consists of all such facets and S_{k+1} consists of vertices of the face.
2. \mathfrak{F}_k^+ is a proper sub-collection of \mathfrak{F}_k : In this case, we construct \mathfrak{F}_{k+1} as follows. First, \mathfrak{F}_{k+1} contains $\mathfrak{F}_k \setminus \mathfrak{F}_k^+$. Besides, for each ridge in \mathfrak{R}_i that is not contained in other \mathfrak{R}_i or shared by other facets in \mathfrak{F}_k^+ , we add the facet formed by the ridge with the point $s_A(\mathbf{n}_{i^*})$ to \mathfrak{F}_{k+1} . After doing this for every ridge collection \mathfrak{R}_i , we obtain a new facet collection \mathfrak{F}_{k+1} . However, \mathbf{b}_{k+1} and S_{k+1} remain the same as \mathbf{b}_k and S_k , respectively.

By the iteration in case 2, though \mathbf{b}_k and S_k do not change, the minimum distance from the origin to the facets in \mathfrak{F}_k is increased. On the other hand, the minimum distance is bounded above by the distance from the origin to the common face shared by the facets in \mathfrak{F}_k . Hence, by repeating this iteration, case 1 will become true, and \mathbf{b}_k will be updated with a new point closer to $\mathbf{z}_A(\mathbf{r})$ (see Fig. 4b). As \mathbf{b}_k approaches

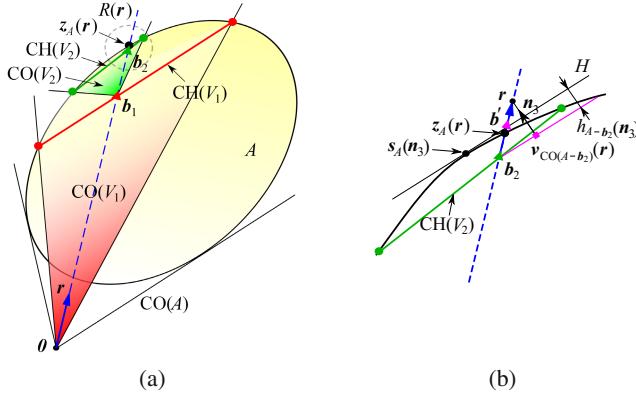


Fig. 5 Illustration of the ZC-based algorithm for ray-shooting in 2-D space. (a) It computes a subset V_k of A in every iteration such that \mathbf{r} is a positive combination of $V_k - \mathbf{b}_{k-1}$. Then \mathbf{b}_k is the intersection point of $\text{CH}(V_k)$ with $R(\mathbf{r})$ and approaches $\mathbf{z}_A(\mathbf{r})$ as the iteration proceeds. (b) The iteration can be stopped by $h_{A-\mathbf{b}_k}(\mathbf{n}_{k+1}) < \varepsilon_{\text{ZC}}$, where \mathbf{n}_{k+1} is the unit vector from $\mathbf{v}_{\text{CO}(A-\mathbf{b}_k)}(\mathbf{r})$ to \mathbf{r} . Then the point \mathbf{b}_k and the hyperplane H with normal \mathbf{n}_{k+1} passing through $\mathbf{s}_A(\mathbf{n}_{k+1})$ bounds $\mathbf{z}_A(\mathbf{r})$ between them, which implies that \mathbf{b}_k is close enough to $\mathbf{z}_A(\mathbf{r})$.

$\mathbf{z}_A(\mathbf{r})$, the stopping condition $h_A(\mathbf{n}_{i^*}) - \mathbf{n}_{i^*}^T \mathbf{b}_k < \varepsilon$ will reach, where i^* indicates the facet in \mathfrak{F}_k closest to the origin. Then, we can derive $\|\mathbf{z}_A(\mathbf{r}) - \mathbf{b}_k\| < \varepsilon \|\mathbf{r}\| / \mathbf{n}_{i^*}^T \mathbf{r}$, and the hyperplane $H(\mathbf{n}_{i^*}, \mathbf{b}_k)$ supports A at $\mathbf{z}_A(\mathbf{r}) = \mathbf{b}_k$. This procedure can handle the situation that $R(\mathbf{r})$ passes through a face of dimension lower than $n-1$ with ensured convergence and generalizes the algorithm proposed in [18].

3.3 A ZC-Based Procedure

We propose a novel procedure based on the ZC algorithm, which iterates a point in A towards $\mathbf{z}_A(\mathbf{r})$. We first let $\mathbf{b}_0 = \mathbf{0}$ and compute $d_{\text{CO}(A)}(\mathbf{r})$ using the ZC algorithm. Then, $R(\mathbf{r})$ intersects A if and only if $\mathbf{r} \in \text{CO}(A)$ or equivalently $d_{\text{CO}(A)}(\mathbf{r}) = 0$.

If $R(\mathbf{r})$ intersects A , the ZC algorithm gives $d_{\text{CO}(A-\mathbf{b}_k)}(\mathbf{r}) = 0$ and a set of linearly independent points in $A - \mathbf{b}_k$, denoted by $\mathbf{a}_1 - \mathbf{b}_k, \mathbf{a}_2 - \mathbf{b}_k, \dots, \mathbf{a}_L - \mathbf{b}_k$, such that

$$\mathbf{r} = \sum_{l=1}^L c_l (\mathbf{a}_l - \mathbf{b}_k) \quad (5)$$

where $\mathbf{a}_l \in A$ and $c_l > 0$ for all l . Let $V_k = \{\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_L\}$, $\sigma = \sum_{l=1}^L c_l$, and

$$\mathbf{b}_{k+1} = \mathbf{b}_k + \frac{1}{\sigma} \mathbf{r} = \sum_{l=1}^L \frac{c_l}{\sigma} \mathbf{a}_l. \quad (6)$$

From (6) it follows that \mathbf{b}_{k+1} is a point on $R(\mathbf{r})$ and it is also a convex combination of V_k , which implies that $\mathbf{b}_{k+1} \in A \cap R(\mathbf{r})$, as depicted in Fig. 5a. Since $\sigma > 0$, \mathbf{b}_{k+1}

is strictly farther from the origin and strictly closer to $\mathbf{z}_A(\mathbf{r})$ than \mathbf{b}_k . By repeating this iteration, \mathbf{b}_{k+1} will converge to $\mathbf{z}_A(\mathbf{r})$ and V_k will become $Z_A(\mathbf{r})$.

As \mathbf{b}_k approaches $\mathbf{z}_A(\mathbf{r})$ and the boundary of A , the ZC algorithm in a certain iteration will terminate with $h_{A-\mathbf{b}_k}(\mathbf{n}_{k+1}) < \varepsilon_{\text{ZC}}$ and $d_{\text{CO}(A-\mathbf{b}_k)}(\mathbf{r}) > 0$, where \mathbf{n}_{k+1} is the unit vector from $\mathbf{v}_{\text{CO}(A-\mathbf{b}_k)}(\mathbf{r})$ to \mathbf{r} , as depicted in Fig. 5b. The hyperplane $H(\mathbf{n}_{k+1}, \mathbf{s}_A(\mathbf{n}_{k+1}))$ supports A and intersects $R(\mathbf{r})$ at $\mathbf{b}' = h_A(\mathbf{n}_{k+1})\mathbf{r}/\mathbf{r}^T\mathbf{n}_{k+1}$. Then, \mathbf{b}' and \mathbf{b}_k bound $\mathbf{z}_A(\mathbf{r})$ between them and $\mathbf{n}_{k+1}^T(\mathbf{b}' - \mathbf{b}_k) = h_{A-\mathbf{b}_k}(\mathbf{n}_{k+1}) < \varepsilon_{\text{ZC}}$. Therefore, we can derive $\|\mathbf{z}_A(\mathbf{r}) - \mathbf{b}_k\| \leq \|\mathbf{b}' - \mathbf{b}_k\| < \varepsilon_{\text{ZC}}\|\mathbf{r}\|/\mathbf{r}^T\mathbf{n}_{k+1}$. Furthermore, $H(\mathbf{n}_{k+1}, \mathbf{b}_k)$ can be regarded as a hyperplane of support to A at $\mathbf{z}_A(\mathbf{r}) = \mathbf{b}_k$.

3.4 Hybrid Uses

3.4.1 A Bi-GJK Procedure

From Sect. 3.3, we know that whether $R(\mathbf{r})$ intersects A can be easily determined by the ZC algorithm and, if so, a point in $A \cap R(\mathbf{r})$ can be obtained, as depicted by \mathbf{b}_1 in Fig. 5a, which we denote by $\hat{\mathbf{b}}$ here. On the other hand, $\check{\mathbf{b}} = h_A(\mathbf{r})\mathbf{r}/\mathbf{r}^T\mathbf{r}$ is a point on $R(\mathbf{r})$ outside the interior of A , as depicted by \mathbf{b}_0 in Fig. 3a. Apparently, $\mathbf{z}_A(\mathbf{r})$ lies on the line segment between $\hat{\mathbf{b}}$ and $\check{\mathbf{b}}$, which allows us to use the bisection method to compute $\mathbf{z}_A(\mathbf{r})$. We call the GJK algorithm to compute the minimum distance $d_A(\mathbf{b})$ between the midpoint $\mathbf{b} = (\check{\mathbf{b}} + \hat{\mathbf{b}})/2$ and A . If $d_A(\mathbf{b}) > 0$, i.e., $\mathbf{b} \notin A$, then we can update $\check{\mathbf{b}}$ with the point from (4) by setting $\mathbf{b}_k = \mathbf{b}$ and $\mathbf{n}_k = (\mathbf{b} - \mathbf{v}_A(\mathbf{b}))/\|\mathbf{b} - \mathbf{v}_A(\mathbf{b})\|$, which is a point on $R(\mathbf{r})$ closer to $\mathbf{z}_A(\mathbf{r})$ than $\check{\mathbf{b}}$ but still outside of A . Otherwise, we can replace $\hat{\mathbf{b}}$ by \mathbf{b} . This procedure can stop as the GJK-based procedure in Sect. 3.1, as shown in Fig. 3b, or once $\|\check{\mathbf{b}} - \hat{\mathbf{b}}\| < \varepsilon$.

3.4.2 A ZC-IE Procedure

The IE procedure can be much faster than the other two procedures if $R(\mathbf{r})$ always passes through the interior of a new facet in every iteration. In that case, the only computation in an iteration is to determine the facet intersected by $R(\mathbf{r})$ among n new facets and record its vertices. By contrast, the GJK-based or ZC-based procedure needs to run the GJK or ZC algorithm in every iteration, whose computation cost is much higher. However, there is no guarantee that $R(\mathbf{r})$ can always pass through the interior of a facet. In case that $R(\mathbf{r})$ happens to pass through only a low-dimensional face shared by several facets, we have to compute and maintain a facet collection and a ridge collection for each facet as discussed in Sect. 3.2, which increases the computation cost of every iteration. To keep a relatively lower computation cost, instead we can call the ZC-based iteration as discussed in Sect. 3.3. If V_k resulting from the ZC algorithm consists of n points, which implies that $\text{CH}(V_k)$ is a facet in A and $R(\mathbf{r})$ passes through its interior, then we can switch back to the IE procedure; otherwise, we continue the ZC-based iteration.

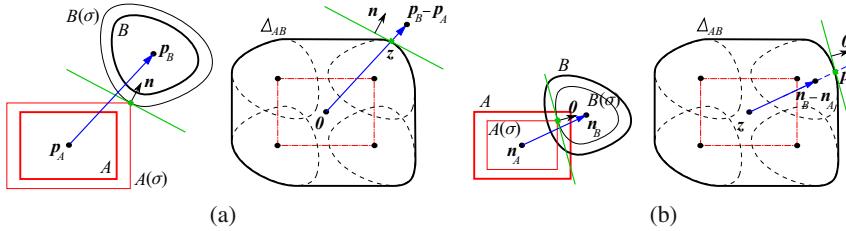


Fig. 6 Illustration of reducing the growth distance between two (a) separated or (b) penetrated convex sets A and B to a ray-shooting problem in 2-D space

4 Application to Collision Detection

Determining the status (i.e., separation, contact, or penetration) between two objects is a fundamental problem in robotics and other fields, such as computer animation and haptics. To do this, a natural way is to compute the minimum Euclidean distance between them [3, 5], but the distance computation for penetrated objects is difficult due to the existence of many local minima. To overcome this trouble and have a unified distance measure for both separated and penetrated objects, the growth distance was proposed [10]. So far, however, there is no efficient algorithm to compute it, which significantly impeded its application. In this section, we show that the computation of growth distance can be reduced to a ray-shooting problem and present numerical results obtained using the aforementioned algorithms.

4.1 Computation of the Growth Distance

Assume that A is a compact convex set with nonempty interior in \mathbb{R}^n , which represents an object and can be specified by triangles meshes or parametric functions, as long as its support function and mapping can be computed. The object A in the global coordinate frame can be expressed as

$$A = \mathbf{p}_A + \mathbf{R}_A(A_0) \quad (7)$$

where $\mathbf{p}_A \in \mathbb{R}^n$ is an interior point of A indicating its position with respect to the global coordinate frame and $\mathbf{R}_A \in SO(n)$ denotes the orientation of A , and A_0 is the description of the object in the local coordinate frame attached at \mathbf{p}_A .

As depicted in Fig. 6, the growth model of A is defined by

$$A(\sigma) \triangleq \mathbf{p}_A + \sigma \mathbf{R}_A(A_0) \quad (8)$$

where $\sigma \geq 0$. Let B be another object and $B(\sigma)$ its growth model defined in the same way as $A(\sigma)$. The growth function of the object pair (A, B) is defined by [10]

$$g(A, B) \triangleq \sigma^* = \min_{A(\sigma) \cap B(\sigma) \neq \emptyset, \sigma \geq 0} \sigma. \quad (9)$$

The growth function (9) computes the minimum scale factor σ^* such that $A(\sigma^*)$ and $B(\sigma^*)$ are not strictly separated from each other, which implies that $A(\sigma^*)$ just contacts $B(\sigma^*)$, as shown in Fig. 6. Then, we can deduce that A and B separate if $\sigma^* > 1$, A and B contact if $\sigma^* = 1$, and A and B penetrate if $\sigma^* < 1$. Since $A(\sigma) \cap B(\sigma) \neq \emptyset$ is equivalent to $\mathbf{0} \in A(\sigma) - B(\sigma)$, from (8) we can rewrite (9) as

$$g(A, B) = \min_{\frac{1}{\sigma}(\mathbf{p}_B - \mathbf{p}_A) \in \Delta_{AB}, \sigma \geq 0} \sigma \quad (10)$$

where $\Delta_{AB} = \mathbf{R}_A(A_0) - \mathbf{R}_B(B_0)$. Equation (10) implies that the calculation of $g(A, B)$ is a ray-shooting problem between Δ_{AB} and $R(\mathbf{p}_B - \mathbf{p}_A)$, as depicted in Fig. 6. It can be deduced that $g(A, B) = \|\mathbf{p}_B - \mathbf{p}_A\| / \|\mathbf{z}_{\Delta_{AB}}(\mathbf{p}_B - \mathbf{p}_A)\|$. Furthermore, from the set $Z_{\Delta_{AB}}(\mathbf{p}_B - \mathbf{p}_A)$ and the normal \mathbf{n} of the supporting hyperplane of Δ_{AB} at $\mathbf{z}_{\Delta_{AB}}(\mathbf{p}_B - \mathbf{p}_A)$, we can derive the contact point between $A(\sigma^*)$ and $B(\sigma^*)$ and the normal at the contact. Due to the page limit, we omit this derivation here.

4.2 Numerical Examples

We implement the aforementioned ray-shooting algorithms in MATLAB on a laptop with an Intel Core i7 2.67GHz CPU and 3GB RAM and apply them to computing $g(A, B)$ between an ellipsoid and a truncated cone, as depicted in Fig. 7. The surface of an ellipsoid or truncated cone is specified by parametric functions, but their sizes and relative positions and orientations are randomly generated. First, we set the two objects to be in contact with each other (Fig. 7a) and shrink (Fig. 7b) or enlarge (Fig. 7c) them by 2 times such that they become separated or penetrated. Then, the true values of $g(A, B)$ are 1, 2, and 0.5 in the three cases (contact, separate, and penetrate), respectively. We generate 3000 such pairs of ellipsoids and truncated cones for each case and report the average absolute error between the computed $g(A, B)$ and the true values and the average CPU running time of each algorithm, as exhibited in Tables 1 and 2, respectively. As a comparison with the existing approach, we also write $g(A, B)$ as a convex programming problem as in [10] and solve it using the active-set algorithm provided by the Optimization Toolbox of MATLAB with default settings. Fig. 8a shows the average error in $g(A, B)$ computed by the ray-shooting algorithms versus their termination tolerance for the three cases. Finally, we randomly generate 100 ellipsoids and truncated cones and compute $g(A, B)$ between any two of them. Then, the average running time of each algorithm is listed in the last row of Table 2. Fig. 8b plots the running time versus the termination tolerance. Table 2 also displays the number of iterations of each algorithm in the above tests.

From the numerical results, we can see that the ray-shooting algorithms are as accurate as the active-set algorithm in MATLAB and their running times are one order of magnitude shorter. It is no surprise that the result accuracy can be improved by simply reducing the termination tolerance. It should be noted that the computation costs for one iteration of these algorithms are different. Although the number of

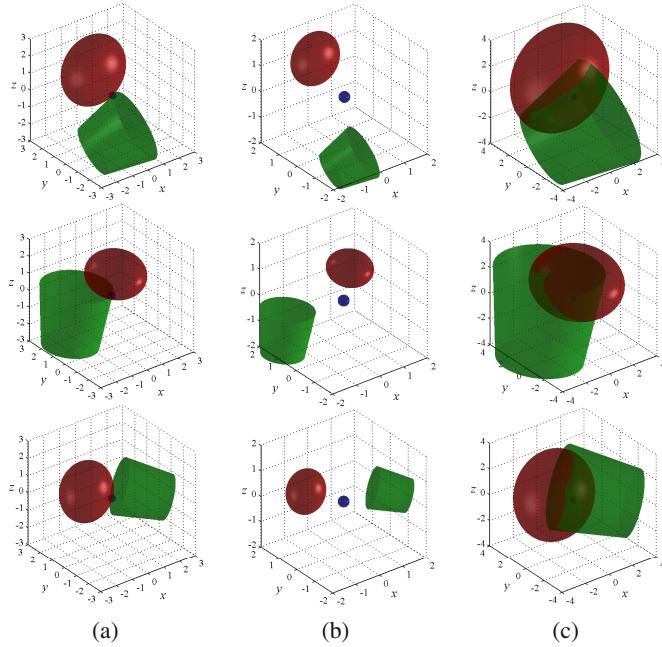


Fig. 7 Growth distance computation between an ellipsoid and a truncated cone. (a) An ellipsoid contacting the side surface (upper), one edge (middle), or one base (lower) of a truncated cone. Then $g(A, B) = 1$. (b),(c) An ellipsoid and a truncated cone obtained by shrinking or enlarging those in (a) by 2 times about their centroids. Then $g(A, B) = 2$ or $g(A, B) = 0.5$.

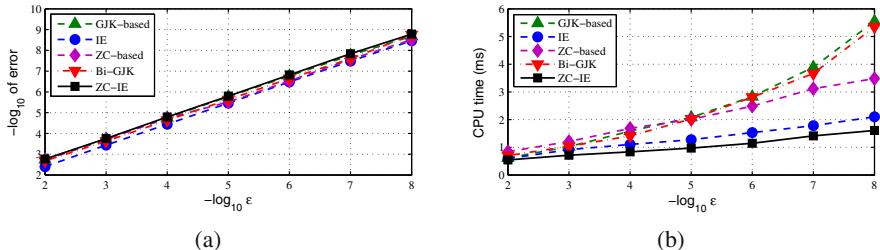


Fig. 8 Absolute error and CPU time versus the termination tolerance ϵ ($\epsilon_{GJK} = \epsilon_{ZC} = \epsilon$). (a) Absolute error in the computed $g(A, B)$. (b) CPU running time of each ray-shooting algorithm.

iterations of the IE or ZC-IE procedure is bigger than or close to those of the other algorithms, they are still faster because the computation in each of their iteration is much simpler. Moreover, the running time of the IE or ZC-IE procedure increases much slower as the termination tolerance decreases.

Table 1 Absolute error of each algorithm from the ground truth $g(A, B)$. ($\varepsilon_{\text{GJK}} = \varepsilon_{\text{ZC}} = \varepsilon = 10^{-5}$)

| | GJK-based | IE | ZC-based | Bi-GJK | ZC-IE | Active-Set |
|-----------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|
| contact | 9.62×10^{-7} | 2.91×10^{-6} | 1.24×10^{-6} | 1.68×10^{-7} | 9.10×10^{-7} | 5.14×10^{-7} |
| separate | 3.92×10^{-6} | 5.92×10^{-6} | 5.10×10^{-6} | 4.60×10^{-6} | 3.52×10^{-6} | 5.42×10^{-6} |
| penetrate | 2.42×10^{-7} | 1.51×10^{-6} | 4.82×10^{-7} | 7.04×10^{-7} | 2.18×10^{-7} | 4.85×10^{-7} |

Table 2 Average CPU running time (in milliseconds) and number of iterations (rounded up to the next highest integer in parentheses) of each algorithm to compute $g(A, B)$. ($\varepsilon_{\text{GJK}} = \varepsilon_{\text{ZC}} = \varepsilon = 10^{-5}$)

| | GJK-based | IE | ZC-based | Bi-GJK | ZC-IE | Active-Set |
|-----------|-----------|-----------|-----------|----------|-----------|------------|
| contact | 2.05 (2) | 1.48 (15) | 2.51 (20) | 1.95 (5) | 1.16 (17) | 12.77 (9) |
| separate | 1.85 (2) | 1.43 (15) | 2.36 (19) | 1.60 (5) | 1.08 (16) | 13.23 (10) |
| penetrate | 2.23 (2) | 1.44 (15) | 2.61 (20) | 2.14 (5) | 1.17 (18) | 13.23 (10) |
| random | 2.08 (2) | 1.28 (17) | 2.01 (21) | 2.02 (6) | 0.97 (18) | 20.05 (16) |

5 Application to Contact Force Optimization

5.1 Problem Description

Consider a robot system making m contacts with the environment, such as a multi-fingered robot hand grasping an object or a legged robot standing on the ground. To maintain the whole system in equilibrium, the resultant wrench \mathbf{w}_{res} from all contact forces must counterbalance the external wrench \mathbf{w}_{ext} (sum of the other wrenches) [9], which can be formulated with respect to the global coordinate frame as

$$\mathbf{w}_{\text{res}} = \sum_{i=1}^m \mathbf{G}_i \mathbf{f}_i = -\mathbf{w}_{\text{ext}} \quad (11)$$

where $\mathbf{G}_i = [\begin{matrix} \mathbf{n}_i & \mathbf{o}_i & \mathbf{t}_i & \mathbf{0} \\ \mathbf{p}_i \times \mathbf{n}_i & \mathbf{p}_i \times \mathbf{o}_i & \mathbf{p}_i \times \mathbf{t}_i & \mathbf{n}_i \end{matrix}] \in \mathbb{R}^{6 \times 4}$, \mathbf{p}_i is the position of contact i , and \mathbf{n}_i , \mathbf{o}_i and \mathbf{t}_i are the unit normal and tangent vectors at contact i in the global coordinate frame and satisfy $\mathbf{n}_i = \mathbf{o}_i \times \mathbf{t}_i$. The contact force \mathbf{f}_i has four components, i.e., three pure force components f_{i1}, f_{i2}, f_{i3} along $\mathbf{n}_i, \mathbf{o}_i, \mathbf{t}_i$, respectively, and a spin moment f_{i4} about \mathbf{n}_i . To maintain a stable contact, \mathbf{f}_i must stay within the friction cone [9]

$$F_i \triangleq \left\{ \mathbf{f}_i \in \mathbb{R}^4 \mid f_{i1} \geq 0, \sqrt{\frac{f_{i2}^2 + f_{i3}^2}{\infty_i^2} + \frac{f_{i4}^2}{\infty_{si}^2}} \leq f_{i1} \right\} \quad (12)$$

where ∞_i and ∞_{si} are the tangential and torsional friction coefficients, respectively.

A major problem in the research of multi-contact robotic systems is to determine whether there exist feasible contact forces $\mathbf{f}_i \in F_i$, $i = 1, 2, \dots, m$ to resist a given

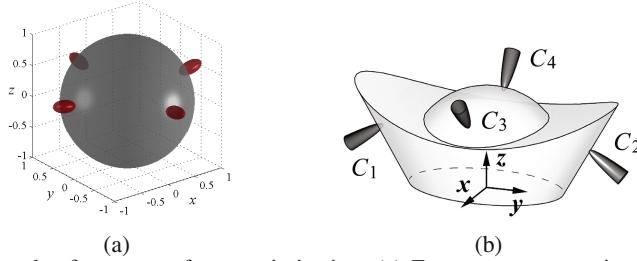


Fig. 9 Examples for contact force optimization. (a) Four contacts grasping a sphere. (b) Four contacts grasping an ingot (also used in [16, 18]).

external wrench as (11) and compute the minimum contact forces if so. The overall contact force magnitude is often measured by

$$\sigma_{L_1} \triangleq \sum_{i=1}^m f_{i1} \quad \text{or} \quad \sigma_{L_\infty} \triangleq \max_{i=1,2,\dots,m} f_{i1}. \quad (13)$$

As described in [6, 16, 18], this problem can be reduced to a ray-shooting problem between the ray $R(-\mathbf{w}_{\text{ext}})$ and the set

$$W_{L_1} \triangleq \text{CH}\left(\bigcup_{i=1}^m W_i\right) \quad \text{or} \quad W_{L_\infty} \triangleq \text{CH}\left(\bigoplus_{i=1}^m W_i\right) \quad (14)$$

where $W_i = \mathbf{G}_i(U_i)$ and $U_i \triangleq \{\mathbf{f}_i \in \mathbb{R}^4 \mid f_{i1} = 1, (f_{i2}^2 + f_{i3}^2)/\infty_i^2 + f_{i4}^2/\infty_{si}^2 = 1\}$. It can be derived that the minimum values of σ_{L_1} and σ_{L_∞} equal $\|\mathbf{w}_{\text{ext}}\|/\|\mathbf{z}_{W_{L_1}}(-\mathbf{w}_{\text{ext}})\|$ and $\|\mathbf{w}_{\text{ext}}\|/\|\mathbf{z}_{W_{L_\infty}}(-\mathbf{w}_{\text{ext}})\|$, respectively. Also, the corresponding contact forces can be computed through the computation of $Z_{W_{L_1}}(-\mathbf{w}_{\text{ext}})$ or $Z_{W_{L_\infty}}(-\mathbf{w}_{\text{ext}})$. The detailed derivation can be found in [6, 16, 18] and is omitted here due to the page limit.

As an extension to the previous work [6, 16, 18], we would like to point out here that the ray-shooting algorithms also work in the case that the overall contact force magnitude is defined and needs to be minimized as

$$\bar{\sigma}_{L_1} \triangleq \sum_{i=1}^m \|\mathbf{f}_i\| \quad \text{or} \quad \bar{\sigma}_{L_\infty} \triangleq \max_{i=1,2,\dots,m} \|\mathbf{f}_i\| \quad (15)$$

where $\|\mathbf{f}_i\| \triangleq \sqrt{f_{i1}^2 + f_{i2}^2 + f_{i3}^2 + (\infty_i^2/\infty_{si}^2)f_{i4}^2}$ is the magnitude of \mathbf{f}_i rather than its normal component f_{i1} used in (13). To do this, we only need to redefine $U_i \triangleq \{\mathbf{f}_i \in F_i \mid f_{i1}^2 + f_{i2}^2 + f_{i3}^2 + (\infty_i^2/\infty_{si}^2)f_{i4}^2 = 1\}$. The derivation and the test of the ray-shooting algorithms to compute minimum contact forces in terms of the above different measures will be included in a complete version of this paper.

5.2 Numerical Examples

We first verify the accuracy of the ray-shooting algorithms applied to the contact force optimization with the example shown in Fig. 9a, where four contacts are located symmetrically on a sphere and have the same elevation angle α . Then, it can be derived that the minimum normal contact force for holding the sphere is $G/4(\infty \cos \alpha - \sin \alpha)$ for each contact and no feasible contact forces exist to do so if $\alpha \geq \tan^{-1} \infty$, where G is the gravity of the sphere and ∞ is the tangential friction coefficient. Thus, the minimum values of σ_{L_1} and σ_{L_∞} are $G/(\infty \cos \alpha - \sin \alpha)$ and $G/4(\infty \cos \alpha - \sin \alpha)$, respectively. Here, we take $G = 10$ N, $\infty = 0.2$, and $\alpha = (1 - \lambda) \tan^{-1} \infty$ for $\lambda = 10^{-1}, 10^{-2}, \dots, 10^{-6}$, respectively. The termination tolerance is 10^{-5} for each ray-shooting algorithm. For comparison, we also formulate the contact force optimization as a convex programming problem and solve it using the active-set algorithm provided by MATLAB, for which the maximum number of function evaluations and the maximum number of iterations are both set to be sufficiently big such that its result can be comparably accurate. Tables 3 and 4 display the relative error in the result of each algorithm compared with the true minimum value of σ_{L_1} or σ_{L_∞} . It is shown that, in this particular case, the results of the ray-shooting algorithms except the IE procedure are highly accurate. Actually, they compute the minimum contact forces by evaluating the support mapping of W_{L_1} or W_{L_∞} only once, which can be done analytically as discussed in [15], and the error in their results is just the round-off error. As the elevation angle α of each contact approaches $\tan^{-1} \infty$, some algorithms report that the problem becomes infeasible. This is because the termination tolerance ϵ is too big for those cases. To relieve this numerical issue and obtain a more accurate result, we have to reduce ϵ .

We also verify the efficiency of these algorithms with the grasping example used in [16, 18], as shown in Fig. 9. We compute contact forces with minimum σ_{L_1} or σ_{L_∞} with respect to 10^3 random w_{ext} and report the average CPU running time and number of iterations of each algorithm, as exhibited in Table 5. It can be seen that the ray-shooting algorithms are several times faster than the active-set algorithm in MATLAB. In particular, the IE and ZC-IE procedures are more efficient than the other ray-shooting algorithms due to the much simpler computation in their iterations, though they need more iterations. Also, their running times rise slower along with the decrease of the termination tolerance, as revealed in Fig. 10. Furthermore, the ZC-IE procedure is even faster than the IE procedure because it saves the computation for maintaining the facet and ridge collections, which is only required on the rare occasion that the ray passes through a face of low dimension. In our implementation of algorithms, the IE and ZC-IE procedures have the same initialization. Then, the difference in their numbers of iterations shown in Table 5 implies that the occasion happens and causes the two procedures to follow different iteration steps.

Table 3 Relative error of each algorithm to minimize σ_{L_1}

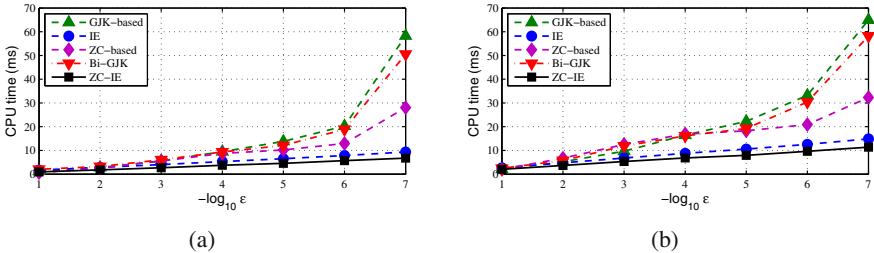
| λ | GJK-based | IE | ZC-based | Bi-GJK | ZC-IE | Active-Set |
|-----------|------------------------|-----------------------|------------------------|------------------------|------------------------|-----------------------|
| 10^{-1} | 6.86×10^{-16} | 4.93×10^{-6} | 0 | 0 | 0 | 3.63×10^{-9} |
| 10^{-2} | 1.35×10^{-14} | 1.98×10^{-6} | 0 | 1.83×10^{-16} | 0 | 1.65×10^{-6} |
| 10^{-3} | 1.38×10^{-13} | 5.38×10^{-6} | 6.88×10^{-14} | 6.91×10^{-14} | 6.88×10^{-14} | 2.88×10^{-8} |
| 10^{-4} | 0 | 2.84×10^{-6} | 6.89×10^{-13} | 6.89×10^{-13} | 6.89×10^{-13} | 3.92×10^{-3} |
| 10^{-5} | 6.89×10^{-12} | infeasible | infeasible | infeasible | infeasible | 1.09×10^{-1} |
| 10^{-6} | infeasible | infeasible | infeasible | infeasible | infeasible | infeasible |

Table 4 Relative error of each algorithm to minimize σ_{L_∞}

| λ | GJK-based | IE | ZC-based | Bi-GJK | ZC-IE | Active-Set |
|-----------|------------------------|-----------------------|------------------------|------------------------|------------------------|-----------------------|
| 10^{-1} | 0 | 6.53×10^{-6} | 0 | 0 | 0 | 7.50×10^{-9} |
| 10^{-2} | 0 | 5.49×10^{-6} | 0 | 6.96×10^{-15} | 0 | 1.18×10^{-8} |
| 10^{-3} | 1.38×10^{-13} | 5.19×10^{-6} | 1.38×10^{-13} | 1.38×10^{-13} | 1.38×10^{-13} | 5.63×10^{-6} |
| 10^{-4} | 0 | 5.39×10^{-6} | 0 | 0 | 0 | 9.65×10^{-5} |
| 10^{-5} | 1.38×10^{-11} | infeasible | infeasible | infeasible | infeasible | 3.00×10^{-4} |
| 10^{-6} | 1.50×10^{-16} | infeasible | infeasible | infeasible | infeasible | 1.70×10^{-3} |

Table 5 Average CPU running time (in milliseconds) and number of iterations (rounded up to the next highest integer in parentheses) of each algorithm for contact force optimization

| | GJK-based | IE | ZC-based | Bi-GJK | ZC-IE | Active-Set |
|------------------------------|-----------|------------|------------|------------|-----------|------------|
| minimize σ_{L_1} | 13.80 (4) | 6.50 (33) | 10.22 (26) | 12.11 (8) | 4.57 (30) | 38.94 (24) |
| minimize σ_{L_∞} | 22.23 (4) | 10.55 (54) | 18.27 (47) | 19.25 (10) | 7.93 (53) | 47.36 (31) |

**Fig. 10** CPU running time versus the termination tolerance ε ($\varepsilon_{\text{GJK}} = \varepsilon_{\text{ZC}} = \varepsilon$) of each ray-shooting algorithm to minimize (a) σ_{L_1} or (b) σ_{L_∞}

6 Conclusions and Future Work

In this paper, we discussed general ray-shooting algorithms and their applications in robotics. We first described three individual procedures and their hybrid uses to compute the ray-shooting problem. They do not use any existing optimization techniques and their implementation is very straightforward. Then, their performance

qualities, including the computational accuracy and efficiency, have been verified and compared with each other in the application to growth distance computation and contact force optimization. Numerical examples show that the ray-shooting algorithms provide better ways to solve these fundamental problems in robotics than some existing methods. Particularly, the ZC-IE procedure, which is newly proposed in this paper, is notably faster than the other ray-shooting algorithms and able to compute results at the same level of accuracy.

Because of the page limit, we could not go into the implementation details of these algorithms and leave it for future publications. As future work, we also would like to apply the algorithms to a dynamic environment, such as collision detection for moving objects or contact force optimization with respect to a time-varying external wrench. Then, by taking advantage of time coherence, they may achieve constant time complexity. We will conduct more experiments on the algorithms, especially in dynamic environments, and report their results in a complete version of this paper. By doing this, we expect to give a comprehensive evaluation of their performance and figure out how to choose an algorithm for a specific problem. In addition, we are exploring other applications of these ray-shooting algorithms.

References

1. Agarwal, P.K., Matoušek, J.: Ray shooting and parametric search. In: Proceedings of the 24th ACM Symposium on Theory of Computing, pp. 517–526 (1992)
2. Ding, D., Liu, Y.-H., Wang, Y., Wang, S.G.: Automatic selection of fixturing surfaces and fixturing points for polyhedral workpieces. *IEEE Transactions on Robotics and Automation* 17(6), 833–841 (2001)
3. Gilbert, E.G., Johnson, D.W., Keerthi, S.S.: A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Transactions on Robotics and Automation* 4(2), 193–203 (1988)
4. Lay, S.R.: Convex Sets and their Applications. John Wiley & Sons, New York (1982)
5. Lin, M., Canny, J.: A fast algorithm for incremental distance calculation. In: Proceedings of the IEEE International Conference on Robotics and Automation, Sacramento, CA, pp. 1008–1014 (1991)
6. Liu, Y.-H.: Qualitative test and force optimization of 3-D frictional form-closure grasps using linear programming. *IEEE Transactions on Robotics and Automation* 15(1), 163–173 (1999)
7. Liu, Y.-H., Lam, M.-L., Ding, D.: A complete and efficient algorithm for searching 3-D form-closure grasps in the discrete domain. *IEEE Transactions on Robotics* 20(5), 805–816 (2004)
8. Matoušek, J., Schwarzkopf, O.: On ray shooting in convex polytopes. *Discrete Computational Geometry* 10(1), 215–232 (1993)
9. Murray, R.M., Li, Z.X., Sastry, S.S.: A Mathematical Introduction to Robotic Manipulation. CRC Press, Boca Raton (1994)
10. Ong, C.J., Gilbert, E.G.: Growth distance: New measures for object separation and penetration. *IEEE Transactions on Robotics and Automation* 12(6), 888–903 (1996)
11. Ong, C.J., Gilbert, E.G.: Robot path planning with penetration growth distance. *Journal of Robotic Systems* 15(2), 57–74 (1998)
12. Rudin, W.: Principles of Mathematical Analysis, 3rd edn. McGraw-Hill, New York (1976)

13. Szirmay-Kalos, L., Havran, V., Balázs, B., Szécsi, L.: On the efficiency of ray-shooting acceleration schemes. In: Proceedings of the Spring Conference on Computer Graphics, Budmerice, Slovakia, pp. 97–106 (2002)
14. van den Bergen, G.: Ray casting against general convex objects with application to continuous collision detection (2004), <http://www.dtecta.com>
15. Zheng, Y., Chew, C.-M.: Distance between a point and a convex cone in n -dimensional space: computation and applications. IEEE Transactions on Robotics 25(6), 1397–1412 (2009)
16. Zheng, Y., Chew, C.-M.: A numerical solution to the ray-shooting problem and its applications in robotic grasping. In: Proceedings of the IEEE International Conference on Robotics and Automation, Kobe, Japan, pp. 2080–2085 (May 2009)
17. Zheng, Y., Chew, C.-M., Adiwahono, A.H.: A GJK-based approach to contact force feasibility and distribution of multi-contact robots. Robotics and Autonomous Systems 59(3-4), 194–207 (2011)
18. Zheng, Y., Lin, M.C., Manocha, D.: A fast n -dimensional ray-shooting algorithm for grasping force optimization. In: Proceedings of the IEEE International Conference on Robotics and Automation, Anchorage, Alaska, pp. 1300–1305 (May 2010)

Optimal Gap Navigation for a Disc Robot

Rigoberto Lopez-Padilla, Rafael Murrieta-Cid, and Steven M. LaValle

Abstract. This paper considers the problem of globally optimal navigation with respect to Euclidean distance for disc-shaped, differential-drive robot placed into an unknown, simply connected polygonal region. The robot is unable to build precise geometric maps of the environment. Most of the robot's information comes from a gap sensor, which indicates depth discontinuities and allows the robot to move toward them. A motion strategy is presented that optimally navigates the robot to any landmark in the region. Optimality is proved and the method is illustrated in simulation.

1 Introduction

If a point robot is placed into a given polygonal region, then computing shortest paths is straightforward. The most common approach is to compute a visibility graph that includes only bitangent edges, which is accomplished in $O(n^2 \lg n)$ time by a radial sweeping algorithm [4] (an $(n \lg n + m)$ algorithm also exists, in which m is the number of bitangents [7]). An alternative is the *continuous Dijkstra method*, which combinatorially propagates a wavefront through the region and determines the shortest path in $O(n \lg n)$ time. Numerous problem variations exist. Computing shortest paths in three-dimensional polyhedral regions is NP-hard [1]. Allowing costs to vary over regions considerably complicates the problem [14, 16]. See [6, 13] for surveys of shortest path algorithms. For recent efforts on curved obstacles, see [2].

Rigoberto Lopez-Padilla · Rafael Murrieta-Cid
Center for Mathematical Research (CIMAT),
Guanajuato, Gto. 36240, México
e-mail: {rigolpz, murrieta}@cimat.mx

Steven M. LaValle
University of Illinois, Urbana, IL 61801, USA
e-mail: lavalle@uiuc.edu

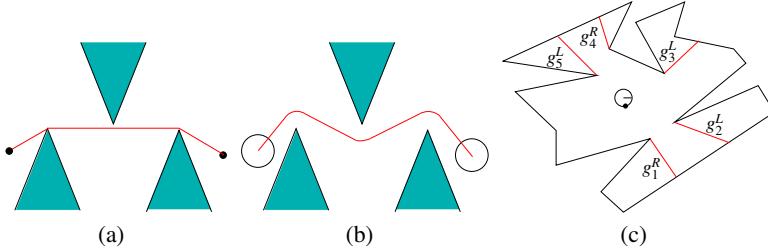


Fig. 1 a) The optimal path for a point robot, b) The optimal path for a disc robot, c) The gap sensor (attached at the small solid disc on the robot boundary) detects the sequence of gaps $G = [g_1^R, g_2^L, g_3^L, g_4^R, g_5^L]$, in which g_1^R and g_4^R are near-to-far gaps and g_2^L , g_3^L , and g_5^L are far-to-near gaps

The approaches described thus far address a point robot, which is unrealistic in most practical settings. It is therefore interesting to study the case of a disc robot, which could correspond, for example, to a Roomba platform. Various objective functions are possible; we choose to optimize the distance traveled by the center of the robot. Once the robot has nontrivial dimensions, the problem can be expressed in terms of configuration space obstacles. Solutions are presented in [3, 11].

Now suppose that the map of the environment is not given to the robot. It must use its sensors to explore and map the environment to develop navigation strategies. Given strong sensors and good odometry, standard SLAM approaches [5, 17] could be applied to obtain a map that can be used as input to the previously mentioned methods. However, we do not allow the robot to localize itself with respect to a *global* reference frame or to build a geometric map. Instead, it observes the world using mainly a *gap sensor*, introduced in [18], which allows it to determine the directions of discontinuities in depth (distance to the boundary) and move toward any one of those directions. Under this model, but for a point robot, a combinatorial filter called the Gap Navigation Tree (GNT) was introduced that encodes precisely the part of the shortest-path visibility graph that is needed for optimal navigation [18]. The learned data structure corresponds exactly to the shortest path tree [6] from the robot's location. This enables the robot to navigate to any previously seen landmark by following the distance-optimal path, even though it cannot directly measure distances. The GNT was extended and applied to exploration in [15]. The GNT was extended to point cloud models in [8]. A larger family of gap sensors is described in [10].

The case of a disc robot is important because real robots have nonzero width. Unfortunately, the problem is considerably more challenging because without additional sensing information, the robot could accidentally strike obstacles that poke into its swept region as it moves along a bitangent. See Figure 1 (a) and (b). The robot must instead execute detours from the bitangent. Sensing, characterizing, and optimally navigating around these obstructions is the main difficulty of this paper.

Before proceeding to the detailed model and motion strategy, several points are important to keep in mind:

1. The robot is placed into an environment, but it is not given the obstacle locations or its own location and orientation. Robot *observes* this information over local reference frames. The purpose is to show how optimal navigation is surprisingly possible without ordinary SLAM.
2. The robot first learns the GNT by executing a learning phase, which is described in [9, 18], and needs just minor changes to be used by disc-shaped robots. The process involves iteratively chasing “unknown” gaps, causing each to split or disappear. Eventually, only primitive gaps, which were formed by appearances of gaps (due to inflection ray crossings), and gaps formed by merging primitives remain. This corresponds to learning the entire shortest-path graph.
3. A simple navigation strategy is provided that guides the robot to any landmark placed in the environment by using the learned GNT. We give precise conditions under which the motions are optimal and prove this statement.
4. We believe that even when the optimality conditions are not met, the strategy itself is close to optimal. Therefore, it may be useful in many practical settings to efficiently navigate robots with simple sensor feedback.

Section 2 formally describes the robot model and the sensor-based motion primitives. Section 3 introduces an automaton that characterizes all possible sequences of motion primitives that could occur when executing optimal motions to a landmark. Section 4 describes how obstacle blockages are detected and handled when the robot chases a gap. This includes detours (that is, the modification of the path encoded in the GNT for a point robot) needed to achieve optimal navigation. Section 6 argues the optimality of the motion strategy. Section 7 presents an implementation in simulation, and Section 8 concludes the paper.

2 Problem Statement

The robot is modeled as a disc with radius r moving in an unknown environment, which could be any compact set $E \subset \mathbb{R}^2$ for which the interior of E is simply connected and the boundary, ∂E , of E is a polygon. Furthermore, it assumed that the collision-free subset of the robot’s configuration space C is connected. C-space obstacle corresponds to that of a translating disc, that is, the extended boundary of E which is due to the robot radius ¹.

2.1 Sensing Capabilities

1) Gap Sensor: The robot has an omnidirectional gap sensor [10, 18], which is able to detect and track two types of discontinuities in depth information: discontinuities from far to near and discontinuities from near to far (in the counterclockwise direction along ∂E). Figure 1(c) shows a robot in an environment in which the gap sensor detects some near-to-far and far-to-near gaps.

¹ Note that this is the configuration space for a translating disc rather than for a rigid body because of rotational symmetry.

Let $G = [g'_1, \dots, g'_k]$ denote the circular sequence of gaps observed by the sensor. Using this notation, t represents the discontinuity type, in which $t = R$ means a discontinuity from near to far (the hidden portion is the right) and $t = L$ means a discontinuity from far to near (the hidden portion is to the left). For example, the gap sensor in Figure 1(c) detects gaps of different types: $G = [g_1^R, g_2^L, g_3^L, g_4^R, g_5^L]$.

We place the gap sensor on the robot boundary and define motion primitives that send the robot on collision-free trajectories that possibly contact the obstacles (moving along the boundary of the free subset of the configuration space is necessary for most optimal paths). These motion primitives, described in detail in Section 2.2, allow the robot to rotate itself so that it is aligned to move the gap sensor directly toward a desired gap, move forward while chasing a gap, and follow ∂E while the sensor is aligned to a gap.

Imagine that a differential drive robot is used. It is assumed that the gap sensor can be moved to two different fixed positions on the robot boundary: The extremal left and right sides with respect to the forward wheel direction. One way to implement this is with a turret that allows the robot to move the gap sensor from its right side to its left side and vice versa. Figure 3(c) shows the sensor aligned to a near-to-far gap in which the gap sensor is on the right side of the robot. To align the sensor to a far-to-near gap, the robot moves the gap sensor to the left side of the robot.

Finally, let Λ be a static disc-shaped landmark in E with the same radius as the robot. A landmark Λ is said to be *recognized* if the landmark is visible at least partially from the location of the gap sensor. Furthermore, during the exploration phase, if required, the landmark can be reached (hence, the complete landmark would be visible from the location of the gap sensor), by traveling along optimal detours.

2) Side Sensors: To detect obstacles that obstruct the robot while it chases a gap, our algorithms need to measure distances between the extremal left and right side robot's points along the direction of the robot heading (forward) and the obstacles. Let those particular robot points be left side point lp and right side point rp . The particular direction tangent to the robot boundary at rp is called rt . The particular direction tangent to the robot boundary at lp is called lt (See Figure 2(a)). Thus, we assume that the omnidirectional sensor is able to *measure* distance. Note that based on distance the discontinuities can be detected. Let d_R be the distance between rp and the obstacles at the particular direction rt , and d_L be the distance between lp and the obstacles at the particular direction lt (see Figure 2(b)).

If the particular direction, either rt or lt , is pointing to a reflex vertex (a gap is aligned with this direction), then a discontinuity in the sensor reading at this direction occurs. Let d_R^t denotes the distance from rp to the closer point on ∂E along the discontinuity direction. Similarly, d_L^t denotes the distance from lp . See Figure 2(c). For avoiding blockages toward the vertex that generates a gap to be chased, the robot rotates. There are two types of rotation: clockwise and counterclockwise. A clockwise rotation can be executed either with respect to rp or a rotation in place (w.r.t. the robot center). Symmetrically, a robot counterclockwise rotation can be executed either with respect to lp or a rotation in place (w.r.t. the robot center).

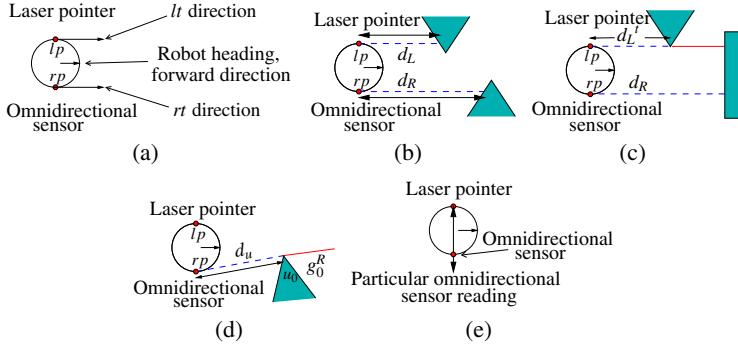


Fig. 2 Side Sensors: (a) Points rp and lp , and directions rt and lt , (b) d_L and d_R , (c) d_L^t , (d) d_u , (e) Omnidirectional sensor readings for contact detection

Finally, let d_u be the distance between the omnidirectional sensor and the vertex u_i that originated the gap g_i (in Figure 2(d) $g_i = g_0^R$).

Our motion strategy will require only comparisons of distances to determine which is larger, rather than needing precise distance measurements. Any small error in the comparison (if the distances are close) causes only a small deviation from optimality, which may be relatively harmless in practice. Our approach will furthermore require detecting whether the robot is contacting ∂E at rp or lp to enable wall-following motions.

Distance measurements between the obstacles and lp and rp in directions rt and lt (forward), and the information of whether the robot is touching ∂E at rp or lp , can be obtained with different sensor configurations. For example, it is possible to use two laser pointers and two contact sensors, each of them located at rp and lp . However, to use a smaller number of sensors and facilitate the instrumentation of the robotic system, it is possible to emulate both the contact sensors and one of the laser pointers, using the omnidirectional sensor. The omnidirectional sensor reading in the particular forward robot heading direction emulate the laser pointer reading. An omnidirectional sensor can also be used to determine whether the robot is touching ∂E at rp or lp . The sensor readings at directions perpendicular to the robot heading are used in this case. If the robot is touching ∂E at the point at which the omnidirectional sensor is located, then the sensor reading is zero. If robot is touching ∂E at the point diametrically opposed to the omnidirectional sensor, then the sensor reading will correspond to the robot diameter (see Figure 2(e)). Thus, one option is to have the robot equipped with an omnidirectional sensor and a laser pointer; they will be located at lp and rp . Recall that a turret can be used to swap the locations of the laser pointer and the omnidirectional sensor to avoid unnecessary robot rotations in place. The gaps or landmarks are always chased with the omnidirectional sensor; the laser pointer is used to detect obstacles and to correctly align the robot.

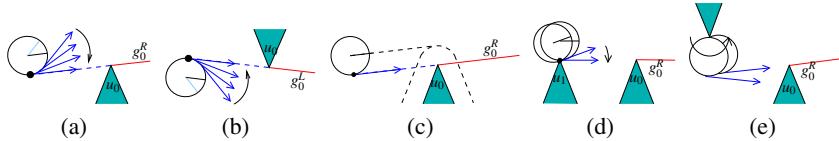


Fig. 3 The motion primitives: (a) Clockwise rotation in place, (b) Counterclockwise rotation in place, (c) Straight line motion, (d) Clockwise rotation w.r.t. to point rp , (e) Counterclockwise rotation w.r.t. to point lp .

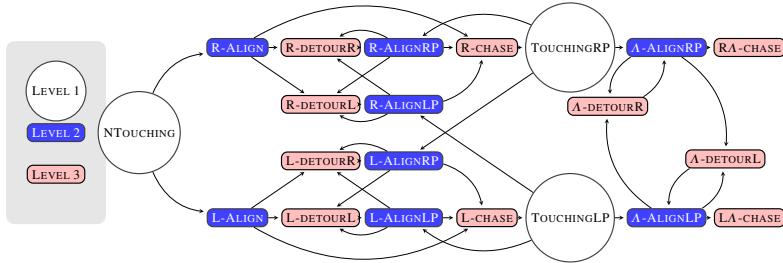


Fig. 4 The sequence of executed primitives depends on sensor feedback. The possible executions are captured by a Moore machine M in which each state applies a specific motion primitive and each transition edge is triggered by a sensing event.

2.2 Motion Primitives

The robot navigates using a sequence of motion primitives that are generated by an automaton for which state transitions are induced by sensor feedback alone. To navigate a gap (or equivalent the vertex that generates it) or a landmark is given to the robot as goal. There are five motion primitives (see Figure 3). Let the angular velocity of the right and left wheels be w_r and w_l , respectively, with $w_r, w_l \in \{-1, 0, 1\}$.

Thus, the motion primitives are generated by the following controls:

- Clockwise rotation in place: $w_r = -1, w_l = 1$.
- Counterclockwise rotation in place: $w_r = 1, w_l = -1$.
- Clockwise rotation w.r.t. to point rp : $w_r = 0, w_l = 1$.
- Counterclockwise rotation w.r.t. to point lp : $w_r = 1, w_l = 0$
- Forward straight line motion: $w_r = 1, w_l = 1$.

The rotation primitives are used to align rt or lt to a specific gap (or landmark). Once rt or lt is aligned to a gap, the robot moves in a straight line to chase the gap. If the path to the chosen gap is blocked, then the robot executes a detour by choosing a new vertex as a subgoal. More details are given in Sections 3 and 4.

3 The Movement Automaton

The algorithm for generating optimal navigation motions can be nicely captured by an automaton or (Moore) finite state machine M . See Figure 4. Every state corresponds to the selection and execution of a motion primitive on the robot or it is a decision. Each state transition is triggered by a sensor observation change. There are 21 total states, with clear right/left symmetry. The ten upper states in Figure 4 correspond to near-to-far gaps (the R cases) and the ten lower states correspond to far-to-near gaps (the L cases). The other remaining state is NTOUCHING, which is used when the robot is not touching ∂E and decides whether the gap to be chased is left or right.

The machine has three main levels (see Figure 4). The first one corresponds to decide whether the goal gap is far-to-near (left gap) or near-to-far (right gap). It also decides whether the robot is touching ∂E . If the robot is touching ∂E , then the first level determines whether the robot is touching it at lp (left side) or rp (right side). The second level is the main one, it detects blockages. According to the decisions made in the first level, the second level makes the robot execute one of the four types of rotations: 1) counterclockwise rotation in place, 2) clockwise rotation in place, 3) clockwise rotation w.r.t. rp and 4) counterclockwise rotation w.r.t. lp . The second level determines whether the path to the goal gap is blocked. According to this decision, the robot executes either a straight line motion toward the gap (the path is not blocked) or executes a detour (the robot travels in a straight line toward the subgoal vertex). The third level is in charge of executing the motion toward the gap to be chased.

In the first level no motion primitive is executed, and in this level, there are three states:

- NTOUCHING: This state happens when the robot is not touching ∂E . It decides whether the gap to be chased is left or right.
- TOUCHINGRP: This state is triggered when the robot is touching ∂E at rp and the gap being chased splits, or the robot goal is a landmark Λ (i.e., the landmark is totally visible to the omnidirectional sensor). The state decides whether the new gap to be chased is a left or right gap. If the new selected gap is a right gap (near-to-far), then the next state will be R-ALIGNRP. If the new selected gap is a left gap (far-to-near), then the next state will be L-ALIGNRP. Finally, whenever the goal is Λ the state will transit to Λ -ALIGNRP.
- TOUCHINGLP: This state is the left symmetric equivalent to TOUCHINGRP.

The second level determines whether the path to the chosen gap is blocked. There are eight states in the second level (four for the right case and four for the left). The states for the right case are:

- R-ALIGN: Right gap alignment executing clockwise rotation in place.
- R-ALIGNLP: Right gap alignment executing counterclockwise rotation w.r.t. lp .
- R-ALIGNRP: Right gap alignment executing clockwise rotation w.r.t. rp .
- Λ -ALIGNRP: Right landmark alignment executing clockwise rotation w.r.t. rp .

There are other four equivalent states when the goal gap is left g_0^L , or the landmark is chased with the omnidirectional sensor located at lp (these are L cases in M). For all of these states, there are three possible transitions. 1) If the path is not blocked, then a straight line motion is allowed. 2) The robot detects a blockage and the subgoal vertex corresponds to a right gap. 3) There is a blockage and the subgoal vertex generates a left gap.

In the third level, the robot always executes a straight line motion. Either the robot moves to the goal gap g_0 (these states are called CHASE) or toward the vertex u that represents the subgoal (these states are called DETOUR). Note that the goal vertex or the vertex that blocks the path can generate a left or right gap; for this reason the states are designed as left or right.

There are ten states in the third level (five for the right case and five for the left). The states for the right case are:

- R-CHASE: The robot moves toward the goal gap g_0^R .
- R-DETOURR: The robot moves toward a subgoal vertex u_n that generates a right gap g_n^R detour.
- R-DETOURL: The robot moves toward a subgoal vertex u_p that generates a left gap g_p^L detour.
- R Λ -CHASE: The robot moves toward Λ , and the omnidirectional sensor is located at rp .
- Λ -DETOURR: The robot moves toward a subgoal vertex u_n that generates a right gap g_n^R detour to the landmark.

There are five symmetrically equivalent states when the goal gap is left g_0^L or the landmark is chased with the omnidirectional sensor located at lp (the L cases in M). This establishes the details of the state machine M .

The next section analyzes blockage detection and gap selection. For lack of space, in this paper we briefly present an algorithm used to determine an optimal detour. This algorithm is described in detail in [12].

4 Blockage Detection and Optimal Detours

During the navigation phase, to detect a blockage, distances d_L , d_R , d_L^t and d_R^t are used. If direction rt is aligned to a vertex that generates a right gap and $d_R^t > d_L$, then a straight line robot path toward this vertex is blocked. Likewise, if direction lt is aligned to a vertex that generates a left gap and $d_L^t > d_R$ then a straight line robot path toward that vertex is blocked. In [12], we prove that during the navigation phase, these conditions are sufficient to detect blockages or declare the robot path collision free. Whenever the path is blocked, the robot executes a detour; that is, the robot travels in a straight line toward another vertex before reaching the vertex associated to the gap being chased (called goal gap). The vertex that generates the original gap to be chased, which is encoded in the GNT, is always visited. For this reason, we call the modified path a detour.

The selection of the gap (or equivalent a vertex) that corresponds to an optimal detour depends on the robot sense of rotation. We call u_p and u_n the vertices that determine an optimal detour. To determine u_p and u_n vertices, it is necessary to compute distances d_R' and d_L' . It is also necessary to compute two angles: 1) the angle that the robot needs to rotate (either counterclockwise or clockwise) to align rt to a right vertex (a vertex that generates a right gap), this angle is called θ_R , and 2) the angle that the robot needs to rotate (either counterclockwise or clockwise) to align lt to a left vertex (a vertex that generates a left gap), this angle is called θ_L . To compute these distances and angles, we locate the vertices in *local reference frames*. Either a reference frame is defined by point rp and a right goal vertex or a reference frame is defined by point lp and a left goal vertex. In [12], a detailed description of the construction of these local reference frames is provided. To find a vertex u_n or a vertex u_p , we use two orders. In one order w.r.t. distance, distance d_R' is used to consider vertices that generate right gaps. Symmetrically, distance d_L' is used to consider vertices that generate left gaps. An order from smaller to larger distances including both d_R' and d_L' is generated. The second order is an angular order also from smaller to larger; vertices are ordered by angle including both θ_R and θ_L , angle θ_R is used to consider vertices that generate right gaps and θ_L is used to consider vertices that generate left gaps. Now, we define u_n and u_p based on these two orders.

Definition 1. The next vertex u_n is the first vertex in clockwise order after the original goal vertex, which is aligned with rt . It is reachable by the robot travelling a straight line path and it corresponds to the optimal detour.

Definition 2. The previous vertex u_p is the last vertex in clockwise order before the original goal vertex, which is aligned with lt . It is reachable by the robot travelling a straight line path and it corresponds to the optimal detour. Refer to Figure 5 and Table 1.

Remark 1. There are two analogous definitions of u_n and u_p , for a counterclockwise rotation.

4.1 Algorithm to Find an Optimal Detour

Table 1 shows an example of the execution of Algorithm 1 and the determination of a u_p vertex; \uparrow indicates the subgoal vertex, \times indicates the vertices that might block the path toward the subgoal vertex, \otimes indicates the vertex selected as subgoal at each iteration, $-$ indicates that the distance to this vertex is smaller than the distance to the subgoal vertex, $+$ indicates that the distance to this vertex is larger than the distance to the subgoal vertex, \rightarrow indicates that for a left vertex, the vertex that must be selected as subgoal is the last in the angular order, and \leftarrow indicates that for a right vertices, the vertex that must be selected as subgoal is the first in the angular order. The algorithm determines that u_f is a u_p vertex corresponding to the optimal detour.

Lemma 1. *Algorithm 1 finds the optimal detour in the sense of Euclidean distance toward the goal vertex.*

Algorithm 1. Handle an optimal detour

1) The algorithm starts from the goal vertex. If the goal vertex is a left vertex, then go to Step 4.

2) Detect left vertices that block the path toward a right goal vertex.

To block the path toward right goal vertex u_{goal}^R , left vertices must have an angle θ_L larger than the angle θ_R related to the goal vertex, and a distance d_L^t smaller than distance d_R^t related to the goal right vertex u_{goal}^R . If no vertex blocks the path toward the goal vertex, then go to Step 6.

3) Selection of a left goal vertex.

The left vertex with largest θ_L , the last in the angular order is selected as a new goal vertex.

4) Detect right vertices that block the path toward a left goal vertex.

To block the path toward left goal vertex u_{goal}^L , right vertices must have an angle θ_R smaller than the angle θ_L related to the goal vertex, and a distance d_R^t smaller than distance d_L^t related to the left goal vertex u_{goal}^L . If no vertex blocks the path toward the goal vertex, then go to Step 6.

5) Selection of a right goal vertex.

The right vertex with smallest angle θ_R , the first in the angular order is selected as a new goal vertex. Go to Step 2.

6) The vertex selected as goal is not blocked.

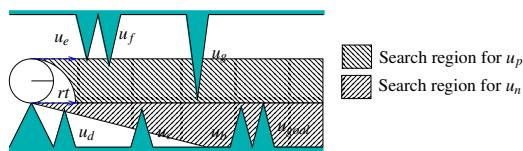


Fig. 5 A u_p vertex

Table 1 Example of orders for selecting a u_p vertex

| Angular order | | | | | | | Distance order | | | | | | | | |
|---------------|------------|----------|------------|-----------|------------|-------|----------------|-----------|-------|------------|------------|------------|------------|-------|------------|
| Index | 1 | 2 | 3 | 4 | 5 | 6 | 7 | Index | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Direction | rt | rt | lt | rt | lt | rt | lt | Direction | rt | lt | lt | rt | lt | rt | rt |
| Type | R | R | L | R | L | R | L | Type | R | L | L | R | L | R | R |
| Vertex | u_{goal} | u_b | u_e | u_c | u_f | u_d | u_g | Vertex | u_d | u_e | u_f | u_c | u_g | u_b | u_{goal} |
| \rightarrow | \uparrow | \times | \times | \otimes | | | | | $-$ | $-$ | $-$ | $-$ | \uparrow | | |
| \leftarrow | | | \otimes | \times | \uparrow | | | | $-$ | | $-$ | \uparrow | $+$ | $+$ | |
| \rightarrow | | \times | \uparrow | \otimes | | | | | $-$ | $-$ | \uparrow | $+$ | | | |
| \leftarrow | | | \uparrow | \times | | | | | $-$ | \uparrow | $+$ | $+$ | $+$ | $+$ | |

Proof. The structure of the path representing a detour is a sequence of sub-paths between vertices, hence for the global path to be optimal, each element of the sequence must be locally optimal. Since each vertex selected as subgoal lies on the boundary of the restriction then each element of the sequence is locally optimal. Therefore, the resulting detour is optimal. \square

5 The Feedback Motion Strategy

Although M represents the decision component of the system, the commands to the motors can be implemented by simple sensor feedback. Only five binary sensor observations affect the control: 1) the robot is touching ∂E with the left side (point lp); (2) it is touching ∂E with the right side (point rp); (3) the robot is aligned to a gap; (4) there is a blockage; and (5) the type of gap (left 0, right 1). Depending on the observation, one of the five different motion primitives will be executed: (1) straight line motion, (2) counterclockwise rotation in place, (3) clockwise rotation in place, (4) counterclockwise rotation with respect to point lp and (5) clockwise rotation with respect to point rp . Recall that the angular velocities of the differential-drive wheels yield one of these motion primitives. Hence, the feedback motion strategy can be established by: $\gamma: \{0, 1\}^5 \rightarrow \{-1, 0, 1\}^2$, in which the sensor observation vector is denoted as $y_i = (rp, lp, aligned, blockage, type)$, to obtain $\gamma(y_i) = (w_r, w_l)$. The set of all 32 possible observation vectors can be partitioned by letting x denote “any value” to obtain: $y_1 = (x, x, 1, 0, x)$, $y_2 = (0, 0, x, 1, 0)$, $y_3 = (0, 0, 0, x, 0)$, $y_4 = (0, 0, x, 1, 1)$, $y_5 = (0, 0, 0, x, 1)$, $y_6 = (x, 1, 0, x, x)$, $y_7 = (x, 1, x, 1, x)$, $y_8 = (1, x, 0, x, x)$, $y_9 = (1, x, x, 1, x)$.

The strategy γ can be encoded as

$$\begin{aligned}\gamma(y_1) &= (1, 1); & \gamma(y_2 \vee y_3) &= (1, -1) \\ \gamma(y_4 \vee y_5) &= (-1, 1); & \gamma(y_6 \vee y_7) &= (1, 0) \\ \gamma(y_8 \vee y_9) &= (0, 1),\end{aligned}$$

in which \vee means “or”.

6 Proof of Optimal Navigation

6.1 Non-blocked GNT-Encoded Paths

In this section we establish that the robot executes a Euclidean, distance-optimal path in the absence of blockages. The shortest path to Λ is encoded as a sequence of gaps in the GNT. Let $U = (u_n, u_{n-1}, \dots, u_0)$ be the sequence of connected intervals $u_i \subset \partial E$ that the robot contacts when the gap sensor (fixed to the robot boundary) moves from its initial position to its final position in Λ . Let $H = (g_n, g_{n-1}, \dots, g_0)$ denote the corresponding sequence of gaps that are chased, in which $g_i \in H$ is the gap that is being chased on the path to u_i or while traversing u_i .

Now consider the problem in terms of the configuration space of the robot. The obstacle region in the configuration space is obtained by growing the environment obstacles by the robot’s radius. Let C denote the projection of the obstacle region into the plane, thereby ignoring rotation. Let $V = (v_n, v_{n-1}, \dots, v_0)$ be the sequence of intervals $v_i \subset \partial C$ obtained by transforming the interval sequence U from ∂E to ∂C , element by element. The following lemma uses the definition of a generalized bitangent from [18].

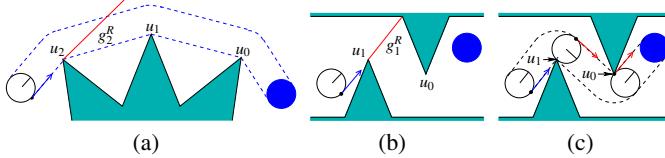


Fig. 6 a) A non-blocked GNT-encoded path that involves only near-to-far gaps, b) A non-blocked GNT-encoded path that involves both types of gaps (near-to-far and far-to-near), c) Robot path.

Lemma 2. *Chasing the sequence H of gaps produces the shortest path if and only if: 1) there is a straight collision-free path from the center of the robot to v_n , 2) there is a (generalized) bitangent line between v_{i+1} and v_i , 3) there is a straight collision-free path from v_0 to the landmark center, and 4) C is connected.*

Proof. Note that if any of the first three conditions is violated, then the robot movement is blocked by an obstacle and therefore does not execute an optimal path. For the last condition, if C is not connected then there is no solution path. \square

Figure 6(a) shows an example of how M generates an optimal path for the non-blocked case. In the figure, the GNT encodes the sequence $H = (g_2^R, g_1^R, g_0^R)$. In this example, the machine M traverses the following sequence of states while generating the appropriate motion primitives: NTOUCHING, R-ALIGN, R-CHASE, TOUCHINGRP, R-ALIGNRP, R-CHASE, TOUCHINGRP, R-ALIGNRP, R-CHASE, TOUCHINGRP, Λ -ALIGNRP, R Λ -CHASE.

Now we describe the association of the states with each gap and the landmark. First, g_2^R is chased, executing states NTOUCHING, R-ALIGN, R-CHASE. Next, g_1^R is chased, executing states TOUCHINGRP, R-ALIGNRP, R-CHASE. Next, g_0^R is chased, executing again the states TOUCHINGRP, R-ALIGNRP, R-CHASE. Finally, Λ is chased, executing states TOUCHINGRP, Λ -ALIGNLP, R Λ -CHASE.

In the previous example all of the gaps in H were of the same type. Using the example illustrated in Figure 6(b), we explain the operation of M when there are different types of gaps. To reach Λ , the robot chases the sequence $H = (g_1^R, g_0^L)$. The resulting sequence is NTOUCHING, R-ALIGN, R-CHASE (from chasing g_1^R), then TOUCHINGRP, L-ALIGNRP, L-CHASE, (from chasing g_0^L), and finally TOUCHINGLP, Λ -ALIGNLP, L Λ -CHASE (from chasing Λ).

6.2 Blocked GNT-Encoded Paths

We now consider the cases for which either of the first three conditions of Lemma 2 is violated, meaning that the robot would become blocked when applying the GNT in the usual way. For these cases, various forms of “detours” are required. The GNT-encoded path is based on the bitangent lines between intervals in E . However, in the configuration space, some bitangent lines disappear. Bitangent lines in the workspace that remain in the configuration space are displaced by a distance r or

are rotated by some fixed angle. The GNT-encoded path cannot be executed by the robot when there is a blockage to chasing $g_i \in H$ (or Λ). If this happens it means that: 1) there is no bitangent line between v_{i+1} and v_i in C , 2) the robot is in a zone in which it cannot detect the crossing of a bitangent line in C , 3) there is no clear path to chase Λ when the robot sees Λ , or 4) C is disconnected. These are the Lemma 2 conditions. We present a solution to deal with the first three cases presented above. However, we do not handle the disconnection of C because there is no path to Λ .

If the robot detects a blocked path, then it performs a detour to avoid the obstacles that blocks the GNT-encoded path. We cannot re-plan the entire path to Λ because the path depends on the gap $g_i \in H$ (or Λ) that is in the gap sensor field of view. For this reason the detour to avoid obstacles is done when the robot detects a blocked path while chasing g_i or Λ . In Section 4, we have presented a proof (see Lemma 1) showing that the detour is optimal. Now, we present the theorem that ensure globally optimal navigation when using M .

Theorem 1. *The path that the robot center follows when commanded by the automaton M , using the information encoded in the GNT and making detours when the path to chase $g_i \in H$ is blocked or when the path to chase Λ is blocked, is optimal in the sense of Euclidean distance.*

Proof. The GNT-encoded path is the shortest path for a point in the workspace and it is in the same homotopy class that the shortest path in C because E and C are simply connected. We have shown that the sequence of connected intervals in E that the robot traverses is only changed when the conditions of Lemmas 2 are not satisfied; therefore, the shortest path for a disc contains the intervals in U . Since each detour is made between consecutive intervals of U , and they are locally optimal (as proved in Lemma 1). Hence, the resulting global path is optimal. \square

Constructing a Complete GNT: In [18], it has been proved that the construction of the GNT for a point robot will terminate (Lemmas 2 and 3 in [18]). Incompleteness of the GNT is caused by any non-primitive leaves (these that correspond to the portions of the environment that have not been perceived by the robot). The key observation to prove the completeness of the GNT for a point robot is that any time that a new gap appears, it must be primitive. If the gap is chased, it cannot split. Therefore, the only gaps that contribute to the incompleteness of the GNT are ones that either appeared in at the beginning of the construction or were formed by a sequence of splits of these gaps. Now, we prove that the construction GNT for a disc robot must also terminate. As mentioned above, compared with the GNT for a point robot, a path that the robot travels to chase a gap will be of two types: non-blocked paths and blocked paths.

Lemma 3. *The learning (construction) phase of the GNT for a disc robot always terminates and produces the same GNT as in [18].*

Proof. To chase a gap, a disc robot first aligns lt to a left vertex, or rt to a right vertex. If $d_R^t > d_L$ then a straight line robot path toward this vertex is blocked. Likewise, if $d_L^t > d_R$ then a straight line robot path toward that vertex is blocked.

For blocked paths, it has been shown in Lemma 1 that the detour is correct and locally optimal. Furthermore, while the robot traverses the detours, it will always have the information about the identity of the original goal gap to be chased g_i , since it is codified in the GNT. For a non-blocked path, there are two sub-cases: 1) From the aligned robot configuration, vertices that can be touched by the robot with point rp (a right vertex) or point lp (a left vertex) traveling a straight line path from the initial robot configuration are equivalent to chase a gap for a point robot; hence the gap can be marked as a primitive gap. 2) From the aligned robot configuration, vertices that cannot be touched by the robot with point rp (a right vertex) or point lp (a left vertex) traveling a straight line path from the initial robot configuration cannot occlude the landmark. Refer to Figures 7(a) and 7(b). Hence, during the exploration phase the related gaps are marked as primitive gaps (explored gaps) at moment that robot touch ∂E with a point different to lp or rp . Therefore, the construction of the GNT for a disc robot must terminate if and only if it would terminate for a point robot. \square

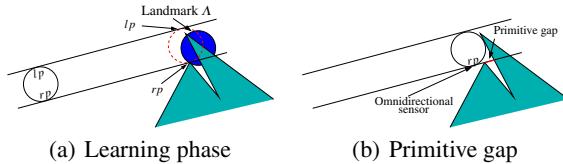


Fig. 7 Gaps marked as primitive gaps during the exploration phase

7 Implementation

We have implemented the method to further verify its correctness. Figure 8 shows a simulation of the optimal gap navigation for a disc robot. The figure shows snapshots of the simulation program. Figures 8(a) and 8(c) show the robot at different times while following a sequence of gaps to reach the landmark. To the right of each figure is shown the complete GNT with the representation used in [18]. The landmark to be chased is marked as a blue triangle in the workspace and in the GNT as a leaf

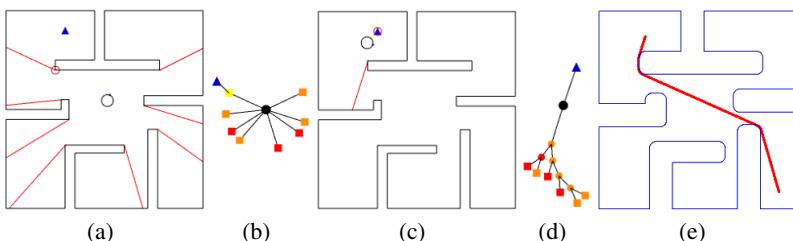


Fig. 8 A simulation of optimal gap navigation for a disc robot. Part (e) shows the path in the projected configuration space that the robot traverses to go to the landmark.

triangle node. Figure 8(a) shows the robot after the first gap split and the robot is chasing the gap that occludes the landmark. Finally, Figure 8(c) shows the robot chasing the landmark. Figure 8(e) shows the shortest path in the configuration space that the robot traverses to navigate to the landmark. This path was computed based in the information obtained by the robot sensors and using the automaton M from Section 3.

8 Conclusions

In this paper we have extended the GNT approach in [18] to a disc-shaped differential-drive robot. The robot is equipped with simple sensors and it is unable to build precise geometric maps or localize itself in a global Euclidean frame. This problem is considerably more challenging than in the case of a point robot because visibility information does not correspond exactly to collision free paths in the configurations space. Consequently, the robot must execute detours from the bitangents in the workspace. Indeed, critical information from the workspace is obtained from the robot’s sensors, to infer the optimal robot paths in the configuration space. To solve this problem we developed a motion strategy based on simple sensor feedback and then proved that the motion strategy yields globally optimal motions in the sense of Euclidean distance by characterizing all possible trajectories in terms of sequences of states visited in a finite state machine. Even if precise distance comparisons are not possible, the motion strategy is simple and effective in a broader setting. Important directions for future work include multiply connected environments, disconnected configuration spaces, and bounds with respect to optimality for the cases in which all sensing conditions are not met.

Acknowledgments. 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. Canny, J., Reif, J.: New lower bound techniques for robot motion planning problems. In: Proceedings IEEE Symposium on Foundations of Computer Science, pp. 49–60 (1987)
2. Chen, D.Z., Wang, H.: Paths among curved obstacles in the plane. In: Proceedings of Computing Research Repository (2011)
3. Chew, L.P.: Planning the shortest path for a disc in $O(n^2 \log n)$ time. In: Proceedings ACM Symposium on Computational Geometry (1985)
4. de Berg, M., van Kreveld, M., Overmars, M., Schwarzkopf, O.: Computational Geometry: Algorithms and Applications, 2nd edn. Springer, Berlin (2000)
5. Durrant-Whyte, H., Bailey, T.: Simultaneous localization and mapping: Part I. IEEE Robotics and Automation Magazine 13(2), 99–110 (2006)
6. Ghosh, S.K.: Visibility Algorithms in the Plane. Cambridge University Press, Cambridge (2007)

7. Ghosh, S.K., Mount, D.M.: An output sensitive algorithm for computing visibility graphs. In: Proceedings IEEE Symposium on Foundations of Computer Science, pp. 11–19 (1987)
8. Landa, Y., Tsai, R.: Visibility of point clouds and exploratory path planning in unknown environments. *Communications in Mathematical Sciences* 6(4), 881–913 (2008)
9. LaValle, S.M.: Planning Algorithms. Cambridge University Press, Cambridge (2006), <http://planning.cs.uiuc.edu/>
10. LaValle, S.M.: Sensing and filtering: A fresh perspective based on preimages and information spaces. *Foundations and Trends in Robotics Series*. Now Publishers, Delft, The Netherlands (2012)
11. Liu, Y.-H., Arimoto, S.: Finding the shortest path of a disc among polygonal obstacles using a radius-independent graph. *IEEE Trans. on Robotics and Automation* 11(5), 682–691 (1995)
12. Lopez-Padilla, R., Murrieta-Cid, R., LaValle, S.M.: Detours for optimal navigation with a disc robot, pp. 1–21 (February 2012),
<http://www.cimat.mx/%7Emurrieta/Papersonline/AppendixWAFR12.pdf>
13. Mitchell, J.S.B.: Shortest paths and networks. In: Goodman, J.E., O'Rourke, J. (eds.) *Handbook of Discrete and Computational Geometry*, 2nd edn., pp. 607–641. Chapman and Hall/CRC Press, New York (2004)
14. Mitchell, J.S.B., Papadimitriou, C.H.: The weighted region problem. *Journal of the ACM* 38, 18–73 (1991)
15. Murphy, L., Newman, P.: Using incomplete online metric maps for topological exploration with the gap navigation tree. In: Proc. IEEE Int. Conf. on Robotics & Automation (2008)
16. Reif, J.H., Sun, Z.: An efficient approximation algorithm for weighted region shortest path problem. In: Donald, B.R., Lynch, K.M., Rus, D. (eds.) *Algorithmic and Computational Robotics: New Directions*, pp. 191–203. A.K. Peters, Wellesley (2001)
17. Thrun, S., Burgard, W., Fox, D.: Probabilistic Robotics. MIT Press, Cambridge (2005)
18. Tovar, B., Murrieta, R., LaValle, S.M.: Distance-optimal navigation in an unknown environment without sensing distances. *IEEE Trans. on Robotics* 23(3), 506–518 (2007)

Min-Max Latency Walks: Approximation Algorithms for Monitoring Vertex-Weighted Graphs

Soroush Alamdari, Elaheh Fata, and Stephen L. Smith

Abstract. In this paper, we consider the problem of planning a path for a robot to monitor a known set of features of interest in an environment. We represent the environment as a vertex- and edge-weighted graph, where vertices represent features or regions of interest. The edge weights give travel times between regions, and the vertex weights give the importance of each region. If the robot repeatedly performs a closed walk on the graph, then we can define the latency of a vertex to be the maximum time between visits to that vertex, weighted by the importance (vertex weight) of that vertex. Our goal in this paper is to find the closed walk that minimizes the maximum weighted latency of any vertex. We show that there does not always exist an optimal walk of polynomial size. We then prove that for any graph there exist a constant approximation walk of size $O(n^2)$, where n is the number of vertices. We provide two approximation algorithms; an $O(\log n)$ -approximation and an $O(\log \rho)$ -approximation, where ρ is the ratio between the maximum and minimum vertex weight. We provide simulation results which demonstrate that our algorithms can be applied to problems consisting of thousands of vertices.

1 Introduction

An emerging application area for robotics is in performing long-term monitoring tasks. Some example problems in monitoring include 1) environmental monitoring tasks such as ocean sampling [15], where autonomous underwater vehicles sense the

Soroush Alamdari
Cheriton School of Computer Science,
University of Waterloo, Waterloo ON N2L 3G1, Canada
e-mail: s26hosse@uwaterloo.ca

Elaheh Fata · Stephen L. Smith
Department of Electrical and Computer Engineering,
University of Waterloo, Waterloo ON N2L 3G1, Canada
e-mail: {efata, stephen.smith}@uwaterloo.ca

ocean to detect the onset of algae blooms; 2) surveillance tasks [12], where robots repeatedly visit vantage points in order to detect events or threats, and; 3) infrastructure inspection tasks such as power-line or manhole cover inspection [18], where spatially distributed infrastructure must be repeatedly inspected for the presence of failures. For such tasks, a key problem is to plan robot paths that visit different parts of the environment so as to efficiently perform the monitoring task. Since some parts of the environment may be more important than others (e.g., in ocean sampling, some regions are more likely to experience an algae bloom than others), the planned path should visit regions with a frequency proportional to their importance.

In this paper, we cast such long-term monitoring tasks as an optimization problem on a vertex- and edge-weighted graph: the *min-max latency walk problem*. The vertices represent features or regions of interest. The edge weights give travel times between regions, and the vertex weights give the importance of each region. Given a robot walk on the graph, the *latency* of a vertex is the maximum time between visits to that vertex, weighted by the importance (vertex weight) of that vertex. We then seek to find a walk that minimizes the maximum latency over all vertices. In an ocean sampling task, this would be akin to minimizing the expected number of algae blooms that occur in any region prior to a robot visit.

Prior work: The min-max latency walk problem generalizes our earlier work [16], where we considered the problem for features distributed in a Euclidean space according to a known probability distribution. Under this setup, constant factor approximation algorithms were developed for the limiting case of large numbers of vertices. However, the algorithms have no performance guarantees for general input graphs that may have non-Euclidean edge weights and smaller numbers of vertices.

In [18], the authors consider a preventative maintenance problem in which the input is the same as in the min-max latency walk problem, but the output is a walk which visits each vertex exactly once. More important vertices (i.e., those that are more likely to fail) should be visited earlier in the path. The authors find a path by solving a mixed-integer program. The min-max latency walk problem can be thought of as a generalization of this problem, where the maintenance and inspection should continually be performed.

The problem considered in this paper is also a more general version of sweep coverage [5], where a robot must move through the environment so as to cover the entire region with its sensor. Variants of this problem include on-line coverage [9], where the robot has no *a priori* information about the environment, and dynamic coverage [10], where each point in the environment requires a pre-specified “amount” of coverage. In [17], a dynamic coverage problem is considered where sensor continually surveys regions of interest by moving according to a Markov Chain. In [3] a similar approach to continuous coverage is taken and a Markov chain is used to achieve a desired visit-frequency distribution over a set of features.

Another related problem is patrolling [4, 8, 14], a region must be continually surveyed by a group of robots. Existing work has considered the case of minimizing the time between visits to each point in space. A variant of patrolling is considered in [2] for continual target surveillance. The persistent monitoring problem

considered in this paper extends the work on patrolling in that different points change at different rates, and the change between visits must be minimized.

Finally, the min-max latency walk problem is related to vehicle routing and dynamic vehicle routing (DVR) problems [1]. One example is the period routing problem [6], where each customer must be visited a specified number of times per week. A solution consists of an assignment of customers to days of the week, and a set of routes for the vehicles on each day.

Contributions: The contribution of this paper are threefold. First, we introduce the general min-max latency walk problem and show that it is well-posed and that it is APX-hard. Second, we provide results on the existence of optimal and approximation algorithms for the problem. We showed that in general, the optimal walk can be very long—it’s length can be non-polynomial in the size of the input graph, and thus there cannot exist a polynomial time algorithm for the problem. We then show that there always exists a constant factor approximation solution that consists of a walk of length $O(n^2)$, where n is the number of vertices in the input graph. Third, and finally, we provide two approximation algorithms for the problem. Defining ρ_G to be the ratio between the maximum and minimum vertex weight in the input graph G , we give a $O(\log \rho_G)$ approximation algorithm. Thus, when ρ_G is independent of n , we have a constant factor approximation. We also provide an $O(\log n)$ approximation which is independent of the value of ρ . The algorithms rely on relaxing the vertex weights to be powers of 2, and then planning paths through “batches” of vertices with the same relaxed weights.

Organization: This paper is organized as follows. In Section 2 we give some background on graphs and formalize the min-max latency walk problem. In Section 3 we present a relaxation of graph weights which allows for the design of approximation algorithms. In Section 4 we present results on the existence of constant factor approximations and some negative results on the required length of the walk. In Section 5 we present two approximation algorithms for the problem. In Section 6 we present large scale simulation data for standard TSP test-cases and in Section 7 we present conclusions and future directions.

2 Background and Problem Statement

In this section, we review graph terminology and define the problem considered in this paper.

2.1 Background on Graphs

The vertex set and edge set of a graph G are denoted by $V(G)$ and $E(G)$ respectively, where $E(G)$ consists of two element subsets of $V(G)$. We write an edge in $E(G)$ as $\{v_i, v_j\}$ or $v_i v_j$. An edge-weighted graph G associates a weight $w(e) \geq 0$ to each edge $e \in E(G)$. A vertex-weighted graph G associates a weight $\phi(v) \in [0, 1]$ to each vertex $v \in V(G)$. Throughout this paper, all referenced graphs are both

vertex-weighted and edge-weighted and therefore we omit the explicit reference. Also, without loss of generality, we assume that there is at least one vertex in $V(G)$ with weight 1, as in our applications weights can be scaled so that this is true. We define ρ_G to be the ratio between the maximum and minimum vertex weight: $\rho_G := \max_{v_i, v_j \in V(G)} \{\phi(v_i)/\phi(v_j)\}$. Given a graph G and a set $V' \subseteq V(G)$, the graph $G[V']$ is the graph obtained from G by removing the vertices of G that are not in V' and all edges incident to a vertex in $V(G) \setminus V'$.

A walk of *length* k in a graph G is a sequence of vertices, $(v_1, v_2, \dots, v_{k+1})$, such that there exists an edge $v_i v_{i+1} \in E(G)$ for $1 \leq i \leq k$. The *weight* of a walk W , denoted by $\text{weight}(W)$, is the sum of the weights of edges of that walk. A walk is closed if its beginning and end are the same vertex. Given a walk $W = (v_1, \dots, v_k)$, and integers $i \leq j \leq k$, the *sub-walk* $W(i, j)$ is defined as the subsequence of W given by $W(i, j) = (v_i, v_{i+1}, \dots, v_j)$. Given the walks W_1, W_2, \dots, W_k , the walk $W = [W_1, W_2, \dots, W_k]$ is the result of concatenation of W_1 through W_k , while preserving order.

An *infinite walk* is a sequence of vertices, (v_1, v_2, \dots) , such that there exists an edge $v_i v_{i+1} \in E(G)$ for $i \in \mathbb{N}$. We say that a closed walk W expands to an infinite walk $\Delta(W)$, when $\Delta(W)$ is constructed by an infinite number of copies of W appended together: $\Delta(W) = [W, W, \dots]$. It can be seen that for any closed walk, there exists a unique expansion to an infinite walk. The *kernel* of an infinite walk W , denoted by $\delta(W)$, is the shortest closed walk such that W is the *expansion* of $\delta(W)$. It is easy to observe that there are infinite walks for which a kernel does not exist. For such an infinite walk W , we define $\delta(W)$ to be W itself.

2.2 The Min-Max Latency Walk Problem

Let G be a weighted graph and W be an infinite walk in G . We define the *latency* of vertex v on walk W , denoted by $L(W, v)$, as the maximum weight of the sub-walk between any two consecutive visits to v on W . Then, we can define the cost of a vertex $v \in V(G)$ on the walk W to be

$$c(W, v) := \phi(v)L(W, v).$$

The cost of an infinite walk W , denoted by $c(W)$ is

$$c(W) = \max_{v \in V(G)} c(W, v).$$

Then, the min-max latency walk problem can be stated as follows.

The min-max latency walk problem. Find an infinite walk W that minimizes the cost $c(W)$.

2.3 Well-Posedness of the Problem

Finding an infinite walk is computationally infeasible. Instead, we will try to find the kernel of the minimum cost infinite walk. The first question, however, is whether or not there always exists a minimum cost walk.

Lemma 1. *For any graph G , there exists a walk of minimum cost.*

Proof. Let W be a walk in G that covers $V(G)$. Let c be the (necessarily finite) cost of W . There are finite walks in G with cost less than c . The reason is that for any vertex $v \in V(G)$, there are finite closed walks beginning and ending in v with weight less than $c/\phi(v)$. Hence there are finite possible costs so that v can induce to a walk of cost less than $c(W)$. In other words, there are finite numbers $c' < c$ that can be the cost of some walk in G . \square

We define OPT_G to be the minimum cost among all infinite walks on G . By Lemma 1, such a number always exists. Let S be the set of kernels of all infinite walks of cost OPT_G in G . We define $\tau(G)$ to be the length of the shortest kernels in S . Next we will show that the problem of min-max latency is APX-hard, implying that there is no polynomial-time approximation scheme (PTAS) for it, unless P=NP.

Theorem 1. *The min-max latency problem is APX-hard.*

Proof. The reduction is from the metric Traveling Salesman Problem (TSP). TSP is the problem of finding the smallest closed walk that visits all vertices exactly once. Such walk is referred to as the TSP tour. It is known that finding the TSP tour is APX-hard in metric graphs [13], and it is approximable within a factor of 1.5. We show a reduction that preserves the hardness of approximation.

Let G be the input of the metric TSP. Assign weight 1 to all vertices of G . Let W be a minimum cost infinite walk in G . Let c be the cost of W and v be the vertex with $c(W, v) = c$. Let i and j be the indices of two consecutive instances of v with $\text{weight}(W(i, j)) = c$. It is easy to see that all vertices of G appear in $W(i, j)$, otherwise, there is another vertex u , with $c(W, v) < c(W, u)$. Let M be a closed walk that is an optimal solution for TSP in G , we prove $\text{weight}(M) = c$. Let $\text{weight}(M) = c'$. It is easy to observe that the cost of $\Delta(M)$ is also c' . Therefore, c' cannot be less than c , because this would contradict the fact that W has minimum cost. Also, c' can not be greater than c , since in that case, the spanning closed walk $W(i, j)$ with cost $c < c'$ would imply existence of a better solution for TSP than M . It is well known that in metric graphs with a closed walk T , there is a cycle T' with $\text{weight}(T) \geq \text{weight}(T')$ that visits the same set of vertices and each vertex exactly once by shortcircuiting the repetitive vertices in T . Note that we showed that the size of the solution for the two problems are equal, hence the reduction is gap preserving and the APX-hardness carries over. \square

We focus on solving the min-max latency problem only for complete metric graphs. The reason is that for any graph G and any $u, v \in V(G)$ we can create a graph G' with the same set of vertices such that the uv edge in G' has weight equal to the shortest-path distance from u to v in G , $d(u, v)$. Then to construct a walk for G based on a

walk in G' , we can replace each uv edge with the shortest path connecting u and v in G . Since $\text{OPT}_G = \text{OPT}_{G'}$ and any walk in G' corresponds to a walk of lower or equal cost in G , any approximation in G' carries over to G . In the literature, the graph G' is referred to as the *metric closure* of G .

3 Relaxations and Simple Bounds

In this section, we present a relaxation of the problem and two simple bounds based on the weights of the edges of the input graph.

3.1 Relaxation of Vertex Weights

Here, we define a relaxation of the problem so that all weights are of the form $1/2^x$, where x is an integer.

Definition 1 (Weight Relaxation). We say weights of vertices are relaxed, if for any vertex $v \in V(G)$, we update its weight to $\phi'(v)$, where $\phi'(v) = \frac{1}{2^x}$ with the property that x is the smallest integer so that $\frac{1}{2^x} \leq \phi(v)$ holds.

Lemma 2 (Relaxed vertex weights). *Let G' be obtained by relaxing the weights of G . Let W be a walk with cost c in G and cost c' in G' . Then $c' \leq c < 2c'$. Consequently, $\text{OPT}_{G'} \leq \text{OPT}_G < 2\text{OPT}_{G'}$.*

Proof. The weight of each vertex in G' is less than or equal to the weight of that vertex in G . Therefore, $c' \leq c$. Also, since the weight of each vertex in G' is more than half of the weight of the same vertex in G , we have that $c < 2c'$. \square

3.2 Simple Bounds on Optimal Cost

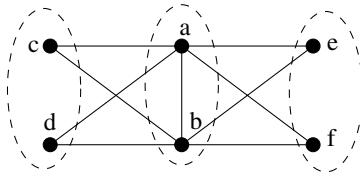
It is easy to observe that no vertex can be too far away from a vertex with weight one, as this distance will bound the cost of the optimal solution.

Lemma 3. *Let G be a graph with $\text{OPT}_G = c$. For any vertex $v \in V(G)$ with weight 1 and any $u \in V(G)$, $d(u, v) \leq c/2$.*

Proof. For the sake of contradiction, assume that $d(u, v) > c/2$ for some $v \in V(G)$ with $\phi(v) = 1$ and $u \in V(G)$. Let u_i be an occurrence of u in W . Let v_j and v_k be the two consecutive occurrences of v on W with $j < i < k$. It is obvious that the sub-walk of W that lies between the two visits to v has length more than c . Since $\phi(v) = 1$, this contradicts the assumption that W has cost c . \square

Corollary 1. *Let G be a metric graph with $\text{OPT}_G = c$. Then the maximum edge weight in G is at most c .*

Fig. 1 The graph G as in proof of Lemma 4 with $n = 6$, $s = 2$, $V_1 = \{a, b\}$, $V_2 = \{c, d\}$ and $V_3 = \{e, f\}$. The walk that Algorithm 1 constructs would be $[[[a, b], c, [a, b], d], [a, b], e, [[a, b], c, [a, b], d], [a, b], f]$.



4 Properties of Min-Max Latency Walks

In this section, we characterize the optimal and approximate solutions of the min-max latency problem.

4.1 Bounds on the Length of Kernel of an Optimal Walk

Here, we first show that the optimal solution for the min-max latency problem can be very large with respect to the size of the input graph.

Lemma 4. *There are infinitely many graphs for which any optimal walk has a kernel that is non-polynomial in the size of G .*

Proof. For any constant integer k and any multiple of it $n = sk$, we construct a graph G with unit weight edges and $|V(G)| = n$ and prove $\tau(G)$ to be in $\Omega(n^{k-1})$. Let V_1, \dots, V_k be a partition of $V(G)$ into k sets each having size s . Let there be a unit weight uv edge for any $u \in V_i$ and $v \in V_i$, where $i \in \{1, 2, \dots, k\}$. For each $v \in V_i$ where $1 \leq i \leq k$, let $\phi(v) = \frac{1}{(s+1)^i}$. We first prove $\text{OPT}_G \leq 1$. Let W be a walk constructed by Algorithm 1. It is easy to see that cost of $\Delta(W)$ is 1. The reason is that each vertex in V_i for $i \in \{1, 2, \dots, k-1\}$ has weight $\frac{1}{(s+1)^i}$ and is visited in $\Delta(W)$ every other $(s+1)^i$ steps. $i \in \{1, 2, \dots, k\}$. Also the vertices in V_k have weight $\frac{1}{(s+1)^k}$ and are visited every other $(s+1)^k - (s+1)^{k-1}$ steps. Therefore $c(\Delta(W), v)$ for any vertex v is 1.

We have proved $\text{OPT}_G \leq 1$. It remains to prove any infinite walk M in G with cost less or equal to 1 has a kernel of size $\Omega(n^{k-1})$. Let M_1 be a sub-walk of length s of M . Then all vertices of V_1 appear in M_1 , otherwise there is a vertex v in V_1 that does not appear in M_1 (note that both cost of M and $|V_1|$ are equal to s). Therefore, $c(M, v) \geq (s+2) \times \frac{1}{s+1} > 1$. This means that after each visit to a member of V_i with $i > 1$, the next s vertices that are visited in M all belong to V_1 .

Now we need to show that at most a single instance of vertices in $\bigcup_{j>i} V_j$ appears in any sub-walk of M of length $(s+1)^{i-1} - 1$. To prove this we use induction on i . Let M' be a sub-walk of M with length $(s+1)^{i-1} - 1$. We can partition the elements of M' into $s+1$ disjoint sub-walks of length $(s+1)^{i-2} - 1$. By the induction hypothesis, we know that each part of this partition has at most a single instance of vertices in $\bigcup_{j>i-1} V_j$. Also, we know that all vertices of V_i appear in M' , or else

Algorithm 1. WalkMaker($\{V_1, \dots, V_{i-1}, V_i\}$)

```

1: if  $i < 1$  then
2:   return  $\emptyset$ 
3: else
4:    $W \leftarrow \emptyset$ 
5:   for  $j = 1 \rightarrow |V_i|$  do
6:      $W \leftarrow [W, \text{WalkMaker}(\{V_1, \dots, V_{i-1}\})]$ ,
7:      $W \leftarrow [W, \text{WalkMaker}(\{V_1, \dots, V_{i-2}\})]$ ,
8:      $W \leftarrow [W, v]$ ; where  $v$  is the  $j$ -th element in  $V_i$ 
9:   end for
10:  return  $W$ 
11: end if

```

the vertex $v \in V_i$ that is not visited in M' would have cost $c(M, v) > 1$. Since there are s vertices in V_i and $s+1$ visits to vertices of $\bigcup_{j>i-1} V_j$ in M' , there is at most a single visit to a vertex in $\bigcup_{j>i} V_j$ in M' . Since all vertices in V_k appear in the kernel of M , this means that the kernel of M has length at least $(s+1)^{k-1} - 1$ which is in $\Omega(n^{k-1})$ since k is a constant. \square

Corollary 2. *There is no polynomial time algorithm for the min-max latency problem.*

Corollary 2 does not show exactly how hard the problem is. In fact, any algorithm that checks all possible walks to find the optimal solution will have complexity $\Omega(c^{e(|V(G)|)})$, where $c > 1$ and e grows faster than any polynomial function.

4.2 Binary Walks

We showed that any exact algorithm is not scalable with respect to the size of the input graph. Therefore, we turn our attention to finding walks that approximate OPT. We show there always exists a walk that has polynomial size and has a cost within a constant factor of the optimal walk. To obtain this result, we first need to define a special structure. Here we define a class of walks, and show that there are walks in this class that provide constant factor approximations.

Definition 2 (Binary Walks and Decompositions). Let G' be a relaxed graph and V_i be the set of vertices with weight $1/2^i$ in G' . Let S be the walk $[S_1, S_2, \dots, S_t]$, where $t = 2^{\log_2 \rho_{G'} + 1}$. We say S is a *binary walk* if for any $v \in V_i$ and $0 \leq j < t/2^i$, v appears exactly once in $S_{j2^i+1}, S_{j2^i+2}, \dots, S_{(j+1)2^i}$, i.e., in each 2^i consecutive S_l 's starting from S_{j2^i+1} , v appears exactly once. Also, we say that the tuple of walks (S_1, S_2, \dots, S_t) is a *binary decomposition* of S .

It is easy to see that $t \leq 2\rho_{G'}$. Also, each vertex appears in each S_i at most once, therefore length of each S_i is bounded by n . This means that S has length bounded by $2n\rho_{G'}$.

Lemma 5. Let G' be a graph with relaxed weights. There is a binary walk W in G' of cost less than or equal to $2.5 \times \text{OPT}_{G'}$. Moreover, since this walk is binary, it has length bounded by $2np_{G'}$.

Proof. Let $M' = (m'_1, m'_2, \dots)$ be an infinite walk of cost c in G' . Note that for any infinite walk, we can remove any prefix of it without increasing the cost of it. Let $M = (m_1, m_2, \dots)$ be an infinite walk of cost c obtained by removing some prefix of M' such that $\phi(m_1) = 1$. Based on M , we construct a binary walk W , such that the cost of $\Delta(W)$ is at most $2c$ as follows: Let a_0 be 0 and S_i be the sub-walk $M(a_i + 1, a_{i+1})$ such that a_{i+1} is the maximal index satisfying $\text{weight}(M(1, a_{i+1})) \leq ic$. Each S_i is a walk of weight at most c , such that the union of S_i 's partitions M .

Now we modify the walks S_1, S_2, \dots by omitting some of the instances of vertices in them. Let V_i be the set of vertices with weight $1/2^i$ in G' . Let $t = 2^{\lfloor \log_2 \rho_{G'} \rfloor + 1}$ as in definition of binary walks. For any vertex $u \in V_i$ and any number $0 \leq j < t/2^i$, omit all but one of the instances of u that appear in $S_{j2^i+1}, S_{j2^i+2}, \dots, S_{(j+1)2^i}$. There exists at least one such instance, otherwise a vertex u with weight $1/2^i$ exists that is not visited in an interval of weight larger than $c \times 2^i$, implying $c(M, u) > c$.

Let S'_1, S'_2, \dots be the result of this modification, note that $\text{weight}(S_i) \geq \text{weight}(S'_i)$. Let S be $[S'_1, S'_2, \dots, S'_t]$. We claim that $\Delta(S)$ has cost at most $2c$. Let $u \in V_i$ be a vertex of G' . Then we know that u appears exactly once in $[S_{j2^i+1}, S_{j2^i+2}, \dots, S_{(j+1)2^i}]$, for any $0 < j$. Also, by the construction we have that for any j, k with $0 < j \leq k$, $[S'_j, S'_{j+1}, \dots, S'_k]$ has weight at most $c(k - j + 1)$. Also since $\phi(m_1) = 1$, by Lemma 3 we know that for any $0 \leq k \leq j$, $[S'_j, S'_{j+1}, \dots, S'_t, S'_1, S'_2, \dots, S'_k]$ has length at most $c((t - j + 1) + 0.5 + k)$. This means $\text{weight}(\Delta(S)(a, b)) < 2^{i+1}c + 0.5c \leq (2.5)c2^i$, for any a and b that are the indices of two consecutive visits to u in $\Delta(S)$. Consequently, the cost $c(\Delta(S), u) \leq 2.5c$. \square

Theorem 2. In any graph G , there exists a closed walk W of length $O(n^2)$, where the cost of $\Delta(W)$ is less or equal to $6 \times \text{OPT}_G$.

Proof. Let G' be the relaxation of G and $U = \{u_1, u_2, \dots, u_{|U|}\}$ be the set of vertices in $V(G')$ with weights less than $1/2^{\lfloor \log n \rfloor + 2}$. Graph G'' is obtained by removing vertices in U from G' . Note that G'' is also a complete metric graph. Therefore $\rho_{G''} \leq 2^{\lfloor \log n \rfloor + 2} \leq 4n$. Let S be a binary walk in G'' , with cost at most $2.5\text{OPT}_{G''}$ as described in Lemma 5. Since $\rho_{G''} \leq 4n$, length of S is bounded by $2\rho_{G''}n \leq 8n^2$.

Now, we add the vertices in U to S in order to obtain a walk W that covers all vertices of G' . Let $v \in V(G')$ be a vertex with $\phi(v) = 1$. Let (S_1, S_2, \dots, S_t) where $t = 2^{\lfloor \log n \rfloor + 2}$ be the binary decomposition of S . Let v_i be the i -th instance of v in S . Note that $t > 2n$, and thus v appears in S at least $2n$ times. For each $1 \leq i \leq |U|$ modify S by duplicating v_{2i} and inserting an instance of u_i between the two copies of v_{2i} . Let W be the resulting walk. We claim that the cost of $\Delta(W)$ is at most $2\text{OPT}_{G'} + \text{OPT}_G$.

Let $\phi(u)$ be $1/2^i \geq 1/2^{\lfloor \log n \rfloor + 2}$ in G' . Let a and b be the indices of two consecutive visits to u in $\Delta(W)$. Then there are at most 2^i instances of vertices in U in $W(a, b)$. This follows from the proof of Lemma 5, where we showed that $W(a, b)$ intersects at most 2^{i+1} members of (S_1, S_2, \dots, S_t) . Of these 2^{i+1} walks, at least half

of them were not changed in W . Therefore, at most 2^i vertices of U lie between indices a and b of W . Also, we inserted the visits to the vertices of U at visits to v with $\phi(v) = 1$. Therefore, by Lemma 3 each of these new detours made to visit a member of U has weight at most $2(\text{OPT}_G/2) = \text{OPT}_G$. Also, by Lemma 5 we already know that $c(\Delta(S), u) \leq 2.5\text{OPT}_{G'}$. Therefore, we already have that:

$$c(\Delta(W), u) < 2.5\text{OPT}_{G'} + \text{OPT}_G \quad (1)$$

Note that the extra 0.5 factor in Lemma 5 is due to the distance of the last vertex of S_t to the first element of S_1 . However, this extra cost can be treated as one of the detours to vertices of U , as we avoided adding one of these detours to S_1 and S_t . This means that we have already accounted for this extra cost in the second part of the righthand side of the inequality 1. Consequently, we have $c(\Delta(W), u) < 2\text{OPT}_{G'} + \text{OPT}_G$. By Lemma 2 we have $\text{OPT}_{G'} \leq \text{OPT}_G$, therefore $c(\Delta(W), u) < 3\text{OPT}_G$. Also by Lemma 2 we know that the cost of W in G would be less than 6OPT_G , concluding the proof. \square

Lemma 6. *Any algorithm with guaranteed output size $O(n^2/k)$ has approximation factor $\Omega(k)$.*

Proof. Let ε be a very small positive number. Let the graph G be as follows:

- There are $n/2$ vertices of weight 1, called heavy vertices,
- There are $n/2$ vertices of weight ε , called light vertices,
- The heavy vertices are in a clique with edges of weight ε ,
- There is an edge of weight 1 connecting any light vertex to any heavy vertex.

In G , any minimum cost infinite walk visits all heavy vertices between visits to two light vertices. This means that each heavy vertex is repeated $n/2$ times in any walk that expands into a minimum cost infinite walk. So far we have shown that any optimum solution has size $\Omega(n^2)$. Note that to reduce the size of the output by a factor k , we would need to visit at least k light vertices between two consecutive visits to some heavy vertex v . This means that a walk of length smaller than $\frac{n^2}{4k}$, has cost at least k , which is $k/2$ times the optimal solution $2 + (\varepsilon \times O(n^2))$. Therefore, any solution for the min-max latency in G with size $O(n^2/k)$ has approximation factor of $\Omega(k)$. This concludes the lemma. \square

Lemma 6 directly gives that there is no constant factor approximation algorithm with guaranteed output size of $o(n^2)$. Note that this implies that Theorem 2 is tight in the sense that the size of the constructed kernel, can not be reduced except for a constant factor, maybe.

5 Approximation Algorithms for the Min-Max Latency Walk

In this section, we present two polynomial time approximation algorithms for the min-max latency problem. The approximation factor in the first algorithm is a function of the ratio of the maximum weight to the minimum weight among vertices.

The approximation ratio of the second algorithm however, solely relies on the number of vertices in the graph.

5.1 An $O(\log \rho_G)$ -Approximation Algorithm

A crucial requirement for our algorithms is a useful property regarding binary walks.

Lemma 7. (Binary property) *Let G' be a graph with relaxed weights. Let S be a binary walk in G' with the binary decomposition (S_1, S_2, \dots, S_t) . Assume we know that $\max_{1 \leq i \leq t} (\text{weight}(S_i)) \leq c$ and for some vertex v , each S_i begins and ends in v . Then the cost of S is at most $2c$.*

Proof. Let S_j be $S_{(j \bmod t)}$. Let V_i be the set of vertices $u \in V(G')$ of weight $1/2^i$. Let $u \in V_i$ be a vertex of G' . Then we know that u appears exactly once in $[S_{j2^i+1}, S_{j2^i+1}, \dots, S_{(j+1)2^i}]$ for any $0 < j$. Also, by the construction and Corollary 1, we have that for any $0 < j \leq k$, $[S_j, S_{j+1}, \dots, S_k]$ has weight at most $c(k - j + 1)$. This means $\text{weight}(\Delta(S)(a, b)) \leq 2c2^i$, for any a and b that are the indices of two consecutive visits to u in $\Delta(S)$. Consequently, the cost $c(\Delta(S), u) < 2c$. \square

Here we define a useful tool. Let the function $\text{Partition}(W, k)$ be a function that gets a walk W and an integer k as input and returns a set of k walks $\{W_1, W_2, \dots, W_k\}$ such that these walks partition vertices of W and also $\text{weight}(W_i) \leq \text{weight}(W)/k$ for all $1 \leq i \leq k$. It is easy to see this is always doable in linear time.

Given a graph G , our first algorithm is guaranteed to find a solution within a factor of $O(\log(1/\varepsilon))$ of the optimal solution, where ε is the smallest weight among the vertices.

Algorithm 2. BrutePartitionAlg(G)

```

1: Let  $V_i$  be the set of vertices of weight  $\frac{1}{2^{i+1}} < w(u) \leq \frac{1}{2^i}$  for  $0 \leq i \leq \log_2 \rho_G$ 
2: Let  $t$  be  $2^{\lfloor \log_2 \rho_G \rfloor + 1}$ 
3:  $S_1, S_2, \dots, S_t \leftarrow \emptyset$ 
4: for  $i = 0 \rightarrow \lfloor \log_2 \rho_G \rfloor$  do
5:    $\{W_1, \dots, W_{2^i}\} \leftarrow \text{Partition}(\text{TSP}(G[V_i]), 2^i)$ 
6:   for  $j = 1 \rightarrow t$  do
7:      $S_j \leftarrow [S_j, W_x]$ ; where  $x$  is  $j \bmod 2^i$ ,
8:   end for
9: end for
10:  $S \leftarrow \emptyset$ 
11: for  $i = 1 \rightarrow t$  do
12:    $S \leftarrow [S, S_i]$ 
13: end for
14: return  $S$ 

```

Theorem 3. *Given a graph G , Algorithm 2 constructs a walk of length $O(n\rho_G)$ that is within $O(\log \rho_G)$ factor of the OPT_G .*

Proof. Let G' be the result of relaxing the weights of G . Let v be a vertex in G' with $\phi(v) = 1$. Let V_i be the set of vertices $u \in V(G')$ of weight $\frac{1}{2^i}$. Let t be the smallest power of two that is larger than ρ_G . Algorithm 2 constructs a binary walk $S = [S_1, S_1, \dots, S_t]$ such that all S_i begin and end in v and $\max_{1 \leq i \leq t} \text{weight}(S_i) < 2(\lceil \log \rho_G \rceil) \text{OPT}_G$.

Assume addition and subtraction on the index of S_i is modulo t (e.g., S_{t+4} is the same as S_4). Here, in addition to the constraints defining a binary walk, we will be trying to satisfy another constraint: Each vertex in V_i appears in S_j through S_{j+2^i-1} exactly once. This condition will force a better behavior of S as it guarantees vertices to be visited more uniformly.

For minimizing the maximum weight S_j , we look at each V_i separately and try to minimize the maximum contribution of V_i to each S_j . Since there are at most $\log \rho_G$ sets V_i , this will give us an overhead approximation factor of $\log \rho_G$.

Let us look at V_i . We will construct 2^i closed walks beginning and ending in v , such that they cover V_i . Let W be the TSP tour of V_i . We showed that the best solution for min-max latency problem in graphs with uniform weight is the same as the TSP tour. Therefore $\text{weight}(W)/2^i \leq \text{OPT}_{G'}$. Let W_1, W_2, \dots, W_{2^i} be a set of 2^i paths partitioning W such that the maximum of them is smaller than $\text{OPT}_{G'}$. Construct W'_j by adding v to the both ends of W_j . By Lemma 3, this increases the weight of each W'_j by at most $2(\text{OPT}_{G'}/2) = \text{OPT}_{G'}$. Therefore, each W'_j has weight at most $2\text{OPT}_{G'}$. Appending each W'_j to S_{2^i+j} for all $0 \leq i \leq t$ will construct our desired solution. Note that since all W'_j end in v , we do not need to worry about concatenation of these walks. In the end, there will be $\lceil \log \rho_G \rceil$ closed walks in S_j each of weight at most $2\text{OPT}_{G'}$. Therefore $\max_{1 \leq j \leq t} (\text{weight}(S_j)) \leq 2\lceil \log \rho_G \rceil \text{OPT}_{G'}$. By Lemma 7 this means that S has cost $4\lceil \log \rho_G \rceil \text{OPT}_{G'}$. Hence by Lemma 2 S has cost within $8\lceil \log \rho_G \rceil$ factor of the optimal solution for G . \square

5.2 An $O(\log n)$ -Approximation Algorithm

In many applications, the value ρ_G is independent of n . For example, in a monitoring scenario, there may be only a finite number of importance levels that can be assigned to a point of interest. In this case we have a constant factor algorithm. However, the ratio between largest and smallest weights ρ_G does not directly depend on the size of the input graph. For even a small graph, ρ_G can be very large. Therefore, in such cases we need an algorithm with an approximation guarantee that is bounded by a function of the size of the graph. Next we present an approximation algorithm for min-max latency problem that is guaranteed to find a solution within logarithmic factor of the optimal solution.

Theorem 4. *Given a graph G , Algorithm 3 constructs a walk of length $O(n^2)$ that is within $O(\log n)$ factor of the OPT_G .*

Proof. The idea is to remove the vertices of small weight so that we can use Algorithm 2 as a subroutine. Let G' be the result of relaxing the weights of G and U be the set of vertices of G' with weight at most $1/2^{\lfloor \log n \rfloor + 1}$. Let G'' be the result of removing vertices in U from G' . Assume $S = [S_1, S_1, \dots, S_t]$ is the result of running Algorithm 2 on G'' with $\rho_{G''} = 2^{\lfloor \log n \rfloor + 2} < 4n$. Add the i -th vertex of U at the end of S_{2i} . Note that since $|U| < n - 1$ and $2n < t$, this is possible. Let $S' = [S'_1, S'_1, \dots, S'_t]$ be the result of this modifications. Each walk S_i begins and ends in v , where $\phi(v) = 1$. Therefore, by Lemma 3 each detour to a vertex in U has weight bounded by OPT_G . Also, by the proof of Theorem 3, each S_i has weight at most $(2 \log n + 2)\text{OPT}_{G'}$. This means that each S'_i has weight at most $(2 \log n + 3)\text{OPT}_G$. By Lemmas 2 and 7, this means that the cost of $\Delta(S')$ in G is bounded by $(8 \log n + 12)\text{OPT}_G$. \square

Algorithm 3. SmartPartitionAlg(G)

- 1: Let V_i be the set of vertices of weight $\frac{1}{2^{i+1}} < w(u) \leq \frac{1}{2^i}$ for $0 \leq i \leq \log \rho_G$
 - 2: Let v be an element with weight 1
 - 3: $U \leftarrow \cup_{i>\lfloor \log n \rfloor} V_i$
 - 4: $S \leftarrow \text{BrutePartitionAlg}(G[V \setminus U])$
 - 5: $i \leftarrow 0$
 - 6: **for all** $u \in V_k$ where $k > \lfloor \log n \rfloor$ **do**
 - 7: Insert u after the $(2i)$ -th instance of v in S
 - 8: Increment i
 - 9: **end for**
 - 10: **return** S
-

6 Simulations

In this section, we present simulation results for the two approximation algorithms presented in Section 5. As Algorithm 3 always performs better than Algorithm 2 both in runtime and approximation factor, we will be studying the performance of Algorithm 3.

For the simulations, we use test data that are standard benchmarks for testing performance of heuristics for calculating TSP tour. The data sets used here are taken from [7]. Each data set represents a set of locations in a country. We construct a graph by placing a vertex for each locations and letting the distance of any pair of vertices be the Euclidean distance of the corresponding points. Unfortunately, no information regarding each individual location was available. Such information could be used to assign weights of the vertices of the graph. For example, if the population of each city was also available, it would have made a meaningful measure for the weights of the vertices.

In many applications of the min-max latency problem—such as monitoring or inspection—the likelihood of a vertex with very high weight is low. In other words, majority of vertices have low priority, while few vertices need to be visited more frequently. To simulate this behavior, we use a distribution that has the following exponential property:

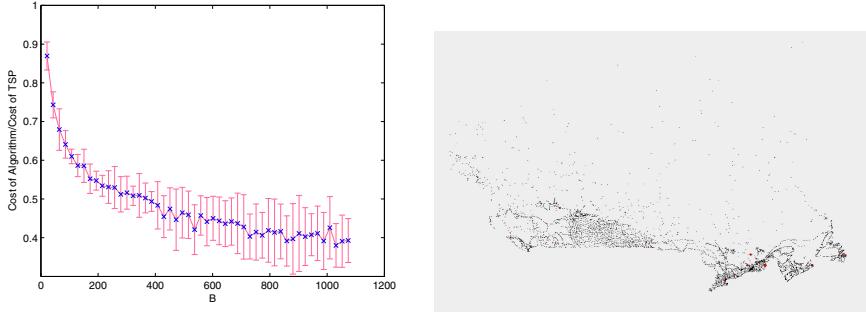


Fig. 2 (Left) The ratio of the cost of the walk produced by Algorithm 3 to the cost of the TSP for different values of B . (Right) The 4663 vertex graph used for all tests corresponding to all cities in Canada [7].

$$P[(1/2)^{k+1} < \phi(v) \leq (1/2)^k] = 1/B \quad (2)$$

for $k < B$ where B is a fixed integer. If we assign to a vertex v the weight $(1/2)^B \leq \phi(v) \leq 1$ with probability $f(\phi(v)) = (\phi(v)B\ln 2)^{-1}$ the exponential property holds.

Here, we compare our algorithms to the simple algorithm of following a TSP tour through all vertices in G . For finding an approximate solution for TSP we used an implementation of the Lin-Kernighan algorithm [11] available at [7]. Relative to other heuristics we test for the min-max latency problem, the cost of TSP tour is low when the weights are distributed uniformly. One of the reasons for this is that when weights are uniform, ρ_G grows proportional to $\log n$. This means that by rounding all weights to ϵ and calculating the TSP we can obtain a solution of expected approximation factor of $\log n$. However, it is easy to construct a graph G in which the cost of the TSP tour of G can be $\Omega(n) \times \text{OPT}_G$.

6.1 Performance with Respect to Vertex Weight Distribution

An important aspect of an environment is the ratio of weight of the elements, therefore it is natural to test our algorithm with respect to ρ_G . Note that $\rho_G > (1/2)^B$. Therefore, we consider different values of B to assess the performance of the algorithm for different ranges of weights on the same graph (see Figure 2). It is easy to see that if $B = 1$, then Algorithm 3 returns the TSP tour of the graph. Also, if $B < \log n$, then Algorithms 3 and 2 behave the same. Figure 2 depicts the behavior of Algorithm 3 on a graph induced by 4663 cities in Canada with different values for B . It can be seen that for larger B our algorithm outperforms the TSP tour by a greater factor.

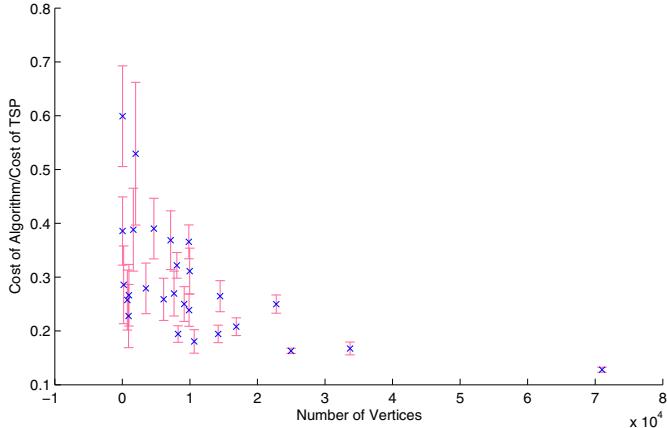


Fig. 3 The ratio of the cost of Algorithm 3 to the cost of the TSP on the 27 test graphs in [7]

6.2 Performance with Respect to Input Graph Size

Here, we use graphs of different sizes to evaluate the performance of our algorithms. Again, the cost is compared to that of a simple TSP tour that visits each vertex in the graph once. Figure 3 depicts the ratio of the cost of the walk constructed by Algorithm 3 to the cost of the TSP tour, on 27 different graphs each corresponding to a set of locations in a different country. Here B is fixed. It can be seen that the ratio of the cost of the TSP tour to the cost of the walk produced by our algorithm increases as the size increases.

Also, the time complexity of the algorithm is $O(n^2 + \beta(n))$ where $\beta(n)$ is the running time of the algorithm used for finding the TSP tour. For the test data corresponding to 71009 locations in China, our Java implementation of Algorithm 3 constructs an approximate solution in 20 seconds using a regular laptop with a 2.50 GHz CPU and 3 GB RAM.

7 Conclusions and Future Work

In this paper, we considered the problem of planning a path for a robot to monitor a known set of features of interest in an environment. We represent the environment as a vertex- and edge-weighted graph and we addressed the problem of finding a closed walk that minimizes the maximum weighted latency of any vertex. We showed several results on the existence and non-existence of optimal and constant factor approximation solutions. We then provided two approximation algorithms; an $O(\log n)$ -approximation and an $O(\log \rho_G)$ -approximation, where ρ_G is the ratio between the maximum and minimum vertex weights. We also showed via simulation that our algorithms scale to very large problems consisting of thousands of vertices.

For future work there are several directions. We continue to seek a constant factor approximation algorithm, independent of ρ_G . We also believe that by adding some heuristic optimizations to the walks produced by our algorithms, we could significantly improve their performance in practice. Finally, we are currently looking at ways to extend our results to multiple robots. One approach we are pursuing is to equitably partition the graph such that the single robot solution can be utilized for each partition.

Acknowledgements. This work was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

1. Bullo, F., Frazzoli, E., Pavone, M., Savla, K., Smith, S.L.: Dynamic vehicle routing for robotic systems. *Proceedings of the IEEE* 99(9), 1482–1504 (2011)
2. Caffarelli, L., Crespi, V., Cybenko, G., Gamba, I., Rus, D.: Stochastic distributed algorithms for target surveillance. In: *Intelligent Systems and Design Applications*, Tulsa, OK, pp. 137–148 (2003)
3. Cannata, G., Sgorbissa, A.: A minimalist algorithm for multirobot continuous coverage. *IEEE Transactions on Robotics* 27(2), 297–312 (2011)
4. Chevaleyre, Y.: Theoretical analysis of the multi-agent patrolling problem. In: *IEEE/WIC/ACM Int. Conf. Intelligent Agent Technology*, Beijing, China, pp. 302–308 (2004)
5. Choset, H.: Coverage for robotics – A survey of recent results. *Annals of Mathematics and Artificial Intelligence* 31(1-4), 113–126 (2001)
6. Christofides, N., Beasley, J.E.: The period routing problem. *Networks* 14(2), 237–256 (1984)
7. Cook, W.: National travelling salesman problem (2009), <http://www.tsp.gatech.edu/index.html>
8. Elmaliach, Y., Agmon, N., Kaminka, G.A.: Multi-robot area patrol under frequency constraints. In: *IEEE Int. Conf. on Robotics and Automation*, Roma, Italy, pp. 385–390 (2007)
9. Gabriely, Y., Rimon, E.: Competitive on-line coverage of grid environments by a mobile robot. *Computational Geometry: Theory and Applications* 24(3), 197–224 (2003)
10. Hussein, I.I., Stipanović, D.M.: Effective coverage control for mobile sensor networks with guaranteed collision avoidance. *IEEE Transactions on Control Systems Technology* 15(4), 642–657 (2007)
11. Lin, S., Kernighan, B.: Effective heuristic algorithm for the traveling salesman problem. *Operations Research* 21, 498–516 (1973)
12. Michael, N., Stump, E., Mohta, K.: Persistent surveillance with a team of mavs. In: *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, San Francisco, CA, pp. 2708–2714 (2011)
13. Papadimitriou, C.H., Yannakakis, M.: The traveling salesman problem with distances one and two. *Math. Oper. Res.* 18, 1–11 (1993)
14. Pasqualetti, F., Franchi, A., Bullo, F.: On optimal cooperative patrolling. In: *IEEE Conf. on Decision and Control*, Atlanta, GA, USA, pp. 7153–7158 (2010)
15. Smith, R.N., Schwager, M., Smith, S.L., Rus, D., Sukhatme, G.S.: Persistent ocean monitoring with underwater gliders: Adapting sampling resolution. *Journal of Field Robotics* 28(5), 714–741 (2011)

16. Smith, S.L., Rus, D.: Multi-robot monitoring in dynamic environments with guaranteed currency of observations. In: IEEE Conf. on Decision and Control, Atlanta, GA, pp. 514–521 (2010)
17. Tiwari, A., Jun, M., Jeffcoat, D.E., Murray, R.M.: Analysis of dynamic sensor coverage problem using Kalman filters for estimation. In: IFAC World Congress, Prague, Czech Republic (2005)
18. Tulabandhula, T., Rudin, C., Jaillet, P.: Machine learning and the traveling repairman (2011), <http://arxiv.org/abs/1104.5061>

Multi-agent Path Planning and Network Flow

Jingjin Yu and Steven M. LaValle

Abstract. This paper connects multi-agent path planning on graphs (roadmaps) to network flow problems, showing that the former can be reduced to the latter, therefore enabling the application of combinatorial network flow algorithms, as well as general linear program techniques, to multi-agent path planning problems on graphs. Exploiting this connection, we show that when the goals are permutation invariant, the problem always has a feasible solution path set with a longest finish time of no more than $n + V - 1$ steps, in which n is the number of agents and V is the number of vertices of the underlying graph. We then give a complete algorithm that finds such a solution in $O(nVE)$ time, with E being the number of edges of the graph. Taking a further step, we study time and distance optimality of the feasible solutions, show that they have a pairwise Pareto optimal structure, and again provide efficient algorithms for optimizing two of these practical objectives.

1 Introduction

Consider the problem illustrated in Fig. 1, which inspired the authors to pursue this research. As an exercise (26-1 in [7]), the *escape problem* is to determine, given $m \leq n^2$ evaders placed on m different points of an $n \times n$ grid, whether there are m vertex disjoint paths from these m locations to m different points on the boundary of the grid. Intended as a demonstration of applications of *maximum flow* algorithms (Ch. 26 of [7]), it undoubtedly mimics multi-agent path planning problems on

Jingjin Yu
Department of Electrical and Computer Engineering,
University of Illinois, Urbana-Champaign
e-mail: jyu18@uiuc.edu

Steven M. LaValle
Department of Computer Science,
University of Illinois, Urbana-Champaign
e-mail: lavalle@uiuc.edu



Fig. 1 Examples of the escape problem on a 6×6 grid. The black discs are the initial evader locations. The goal is to plan disjoint paths for the evaders to reach different vertices on the boundary of the grid. a) An instance with solution given as the bold edges. b) An instance without a solution.

graphs. Intrigued by the elegant network flow based solution to the escape problem, we wonder: How tightly are these two classes of problems intertwined and how we may take advantage of the relationship?

In this paper, we explore and exploit the connection between multi-agent path planning on *collision-free unit-distance graphs* (or CUGs, see Section 2 for the definition) and network flow. We begin by showing that multi-agent path planning on CUGs is closely related to a class of problems called *dynamic network flow* or *network flow over time*. We then focus on the permutation invariant multi-agent path planning problem on CUGs (by permutation invariant, we mean that goals are not pre-assigned to agents. Instead, we only require that each goal is reached by a unique agent), establishing that such problems always have solutions. To solve the problem algorithmically, an adapted maximum flow algorithm is provided which plans collision free paths for all agents with worst time complexity $O(nVE)$, in which n is the number of agents, V is the number of vertices of the CUG and E is the number of edges of the CUG. Moreover, we guarantee that the last agent takes time no more than $n + V - 1$ to reach its goal, assuming that agents travel at unit speed. Next, we construct efficient algorithms for obtaining temporally and spatially optimal solutions. For example, our algorithm for shortest overall time has running time $O(nVE \log V)$. We also show that these temporal and spatial objectives cannot be optimized simultaneously (i.e., they have a *Pareto optimal* structure).

As a universal subroutine in multi-agent systems, collision-free path planning for multiple agents finds applications in tasks spanning assembly [18, 31], evacuation [4, 37], formation control [2, 36, 39, 41, 43], localization [14], object transportation [29, 38], search and rescue [20], and so on. Given its importance, path planning for multi-agent systems has remained as a subject of intense study for many decades. Due to the vast size of the available literature, we only mention a most related subset of the research in this field and refer the readers to [5, 24, 26] and the references therein for a more comprehensive review of the subject.

When all agents are treated as a single agent with a high dimensional configuration space, the problem can be solved using cylindrical algebraic decomposition [6] or Canny's roadmap algorithm [3], in theory. Such *coupled* approaches suffer from

the curse of dimensionality; even when sampling based methods [22, 25] are used, instances involving only a small number of agents can be computationally challenging. This difficulty prompts the study of methods that seek to explore local features whenever possible to avoid working with too many agents at a time. Among these, *decoupled* planning is the most popular, which generally performs coordination of robot motion after deciding a path for each robot [17, 21, 32, 35, 40, 47]. In contrast, priority based methods force an order on agents to significantly reduce the search space [10, 45]. Some more recent works using decoupling heuristics include applying optimal decoupling techniques to exploit problem instances with low degrees of coupling [46], using *push-and-swap* primitives to avoid unnecessary exploration of search space [28], and heuristics aimed at performance guarantees (completeness is lost) [48].

Our algorithmic efforts in this paper focus on the *permutation invariant* multi-agent path planning problem on CUGs. Such formulations, in both discrete and continuous forms, are extensively studied as formation control problems [2, 36, 39, 41, 43], among others. On research that appears mostly related to this aspect of our paper, a discrete grid abstraction model for formation control was studied in [30]. To plan the paths, a three-step process was used in [30]: 1) Target assignment, 2) Path allocation, 3) Trajectory scheduling. Although it was shown that the process always terminates, no characterization of solution complexity was offered. In contrast, we provide very efficient algorithms that solve a strictly more general class of problems with optimality assurance. On the continuous side, a novel *formation space* approach was employed to represent the entire formation of robot teams with a single polynomial of which the roots correspond to the unassigned configurations for the robots in the formation [23].

We delay the literature review on network flow, from which we devise our time expansion construction for multi-agent path planning, to Section 3. The basic idea of applying time expansion to robotics problem is far from new [10, 34]. To the best of our knowledge, however, the research presented here is an original attempt at proposing a general time expansion technique, connecting it to network flow, and making full use of the benefits that come with this approach. We also note that our exact and complete algorithms all come with low constants in their respective worst case time complexity because they are derived from well studied fully combinatorial algorithms¹. Our simulation result, which we omit due to the length limit, confirms this assertion.

There are three main contributions. First, we formally establish the link between multi-agent path planning on graphs and network flow, showing how multi-agent path planning can be reduced to network flow problems, thereby enabling the potential application of powerful tools from combinatorial optimization to path planning for multiple agents in a principled way. Second, for the planning problem in which agents do not have pre-specified goals, we give fast and complete algorithms for finding collision free path sets that deliver every agent to a different goal. Third, we

¹ A *fully combinatorial algorithm* is an algorithm that only adds, subtracts, and compares values; no multiplication and division operations are allowed (i.e., ordered *group* operations versus ordered *field* operations) [16].

study time and distance optimality of the feasible solutions to the aforementioned problem, show that they have a pairwise Pareto optimal structure, and again provide efficient algorithms for optimizing two of these practical objectives.

The rest of the paper is organized as follows. In Section 2, we define two multi-agent path planning problems on CUGs. Section 3 starts with a quick review of network flow problems and then proceeds to show the reduction from multi-agent path planning on CUGs to network flow. Concentrating our efforts on the permutation invariant multi-agent path planning problem, Section 4 begins with a key construction that allows us to tightly bound the time steps required for a time-expanded network to have a feasible solution, which in turn enables efficient algorithms. Section 5 takes a further step and studies solution optimality on three natural objectives, showing the objectives have a Pareto optimal structure. We conclude in Section 6. Proof sketches of key theorems are provided; full proofs are omitted due to length².

2 Multi-agent Path Planning Problems on Collision-Free Unit-Distance Graphs

Let $G = (V, E)$ be a connected, undirected, simple graph (i.e., no multi-edges), in which $V = \{v_i\}$ is its vertex set and $E = \{(v_i, v_j)\}$ is its edge set. Let $A = \{a_1, \dots, a_n\}$ be a set of agents with initial and goal locations on G given by the injective maps $x_I : A \rightarrow V$ and $x_G : A \rightarrow V$, respectively. Note that A is essentially an index set; $x_I(A)$ and $x_G(A)$ are the set of initial and goal locations, respectively. We require that $x_I(A)$ and $x_G(A)$ be disjoint. For convenience, we let $n = |A|$ and use V, E to denote the cardinality of the sets V, E , respectively, since the meaning is usually clear from the context. Let σ be a bijection³ that acts on x_G , a *feasible path* for a single agent a_i is a map $p_i : \mathbb{Z}^+ \rightarrow V$ with the following properties⁴: 1. $p_i(0) = x_I(a_i)$. 2. For each i , there exists a smallest $k_{\min} \in \mathbb{Z}^+$ such that $p_i(k_{\min}) = (\sigma \circ x_G)(a_i)$ for some fixed σ . That is, the end point of the path p_i is some goal vertex. 3. For any $k \geq k_{\min}$, $p_i(k) \equiv (\sigma \circ x_G)(a_i)$. 4. For any $0 \leq k < k_{\min}$, $(p_i(k), p_i(k+1)) \in E$ or $p_i(k) = p_i(k+1)$. Intuitively, think of the domain of the paths as discrete time steps. We say that two paths p_i, p_j are in *collision* if there exists $k \in \mathbb{Z}^+$ such that $p_i(k) = p_j(k)$ (meet) or $(p_i(k), p_i(k+1)) = (p_j(k+1), p_j(k))$ (head-on). If $p(k) = p(k+1)$, the agent stays at vertex $p(k)$ during the time interval $[k, k+1]$.

As mentioned, in this paper, we work with a specific type of graph called the collision-free unit-distance graph (CUG): A CUG is a connected, undirected graph G satisfying the following: 1. Every edge is of unit length; 2. Given any two distinct edges (u_1, v_1) and (u_2, v_2) of G with $u_1 \neq u_2, v_1 \neq v_2$, two disc shapes (or spherical for 3D or more) agents of radius less than $\sqrt{2}/4$ traveling at unit speed through these edges (starting simultaneously at u_1, u_2 , respectively) will never collide. A

² A comprehensive extended version of this paper, including all proofs and extensions, is permanently available at <http://arxiv.org/abs/1204.5717>.

³ σ is introduced to unify the problem formulations; its use will become clear shortly. For now, the reader may think of it simply as the identity map.

⁴ $\mathbb{Z}^+ := \mathbb{N} \cup \{0\}$.

radius of $\sqrt{2}/4$ is the largest possible for two adjacent agents to travel along an “L” shaped path. One can easily verify that any graph with unit edge length and no acute angles between adjacent edges is a CUG. Hence, a connected 2D grid with holes is a CUG. Since subgraphs of 2D grids are easy to draw and visualize, we generally use subgraphs of 2D grids when we create examples in this paper. With the above setup, the *multi-agent path planning on CUGs* problem is defined as follows.

Problem 1. [Multi-agent Path Planning on CUGs] Given a 4-tuple (G, A, x_I, x_G) in which G is a CUG, find a set of paths $P = \{p_1, \dots, p_n\}$ such that p_i ’s are feasible paths for respective agents a_i ’s with σ being the identity map and no two paths p_i, p_j are in collision.

We require the graph to be a CUG so that it is suitable for multi-agent path planning. We formalize the rationale in the following observation.

Observation 1. Let p_i, p_j be two paths that are not in collision (as a partial solution to Problem 1). Then two disc shaped agents⁵ of radius less than $\sqrt{2}/4$, starting at the same time and moving along these respective paths with unit speed, will never collide.

Observation 1 shows that a solution to Problem 1 provides a path set for disc agents with radius $\sqrt{2}/4$ in A to reach their respective goals without a collision. It is easy to see that not all instances of this problem are solvable. Moreover, the decision version of Problem 1 (i.e., is there a solution that takes the agents to goals within K time steps) is NP-complete⁶. If we remove the assumption that all agents must reach their respective goals and allow permutation invariant paths (i.e., as long as each goal gets occupied by a unique agent in the end), Problem 1 becomes the *permutation invariant multi-agent path planning on CUGs* problem.

Problem 2. [Permutation Invariant Multi-agent Path Planning on CUGs] Given a 4-tuple (G, A, x_I, x_G) in which G is a CUG, find a set of paths $P = \{p_1, \dots, p_n\}$ such that p_i ’s are feasible paths for respective agents a_i ’s for an arbitrary (but fixed) permutation σ and no two paths p_i, p_j are in collision.

Problem 2 models the problem in which multiple identical or indistinguishable agents need to be deployed for serving requests at different locations (for example, formation control). This problem always has a solution: We simply plan and execute one path at a time and use more “remote” goal vertices earlier to avoid possible blocking of later paths; a formal result on the existence of such a choice of paths is given in Section 4.

⁵ Or spherical agents with radius less than $\sqrt{2}/4$, for dimensions higher than 2.

⁶ The lengthy proof is out of scope for the current paper. For curious readers, the NP-hardness proof is similar to that from [42]; the NP membership proof, which is non trivial, leads to efficient complete algorithms for solving Problem 1.

3 Multi-agent Path Planning on CUGs and Network Flow

3.1 Network Flow

In this subsection we give a brief review of network flow problems and algorithms pertinent to our problems. For surveys on network flow, see [1, 13]. We start with the classic static network flow problems.

Static Network Flow. A *network* $\mathcal{N} = (G, u, c, S)$ consists of a directed graph $G = (V, E)$ with $u, c : E \rightarrow \mathbb{Z}^+$ as the maps defining the capacities and costs on edges, respectively, and $S \subset V$ as the set of sources and sinks. We let $S = S^+ \cup S^-$, with S^+ denoting the set of sources and S^- denoting the set of sink vertices. Following conventions, for a vertex $v \in V$, let $\delta^+(v)$ (resp. $\delta^-(v)$) denote the set of edges of G going to (resp. leaving) v . A feasible (static) S^+, S^- -flow on this network \mathcal{N} is a map $f : E \rightarrow \mathbb{Z}^+$ that satisfies edge capacity constraints,

$$\forall e \in E, \quad f(e) \leq u(e), \quad (1)$$

the flow conservation constraints at non terminal vertices,

$$\forall v \in V \setminus S, \quad \sum_{e \in \delta^+(v)} f(e) - \sum_{e \in \delta^-(v)} f(e) = 0, \quad (2)$$

and the flow conservation constraints at terminal vertices,

$$\sum_{v \in S^+} \left(\sum_{e \in \delta^-(v)} f(e) - \sum_{e \in \delta^+(v)} f(e) \right) = \sum_{v \in S^-} \left(\sum_{e \in \delta^+(v)} f(e) - \sum_{e \in \delta^-(v)} f(e) \right). \quad (3)$$

The quantity on either side of (3) is called the *value* of the flow.

The classic (single-commodity) *maximum flow* problem asks the question: Given a network \mathcal{N} , what is the maximum value of flow that can be pushed through the network (i.e., seeking to maximize the value of the flow)? The *minimum cost maximum flow* problem further requires the flow to have minimum total cost among all maximum flows. That is, we want to find the flow among all maximum flows such that the quantity

$$\sum_{e \in E} c(e) \cdot f(e) \quad (4)$$

is minimized. Given integer inputs, integer maximum flow always exists, and many polynomial time algorithms exist for finding such a solution [9, 15]. The minimum cost maximum flow problem is equivalent to the *minimum cost circulation* problem, which is also solvable in polynomial time [44].

When additional structure is put on S , additional questions arise. If we limit the supply (resp. demand) of the source (resp. sink) vertices, we obtain a type of the flow problem called the *transshipment* problem. To formalize this, let $d : V \rightarrow \mathbb{Z}$ be the supplies on the vertices of G . Given a vertex $v \in V$, a positive $d(v)$ suggests that

the vertex has positive supply ($v \in S^+$) and a negative one suggests that the vertex has positive demand ($v \in S^-$). For all other vertices v , $d(v) = 0$. The basic version of the transshipment problem asks for a feasible flow through the network that also respects the supply/demand requirements

$$\begin{aligned} \forall v \in S^+, \sum_{e \in \delta^-(v)} f(e) - \sum_{e \in \delta^+(v)} f(e) &= d(v), \\ \forall v \in S^-, \sum_{e \in \delta^+(v)} f(e) - \sum_{e \in \delta^-(v)} f(e) &= d(v). \end{aligned} \quad (5)$$

The transshipment problem becomes the *evacuation* problem when $|S^-| = 1$ and the demand of the single sink vertex is equal to the total supply of the source vertices. The transshipment problem and the evacuation problem, as special cases of the maximum flow problem, can be solved with maximum flow algorithms mentioned above. If we instead require that vertices of S^+, S^- are paired up as $(s_1, s'_1), \dots, (s_k, s'_k)$ and that commodity of type i can be injected only into s_i and taken out at s'_i , we get the *multi-commodity flow* problem. Optimality questions as these from the single-commodity case can be asked here as well. Unlike in the single commodity case, finding integer maximum flow for multi-commodity problems is NP-hard in general and MAX SNP-hard (NP-hard to approximate below a certain multiple of optimal flow value) even for some simple restrictions [8].

Dynamic Network Flow. If we consider that flowing commodities through edges takes some time to complete, the problem becomes a *dynamic network flow* problem, which sometimes is also called *network flow over time*. There are two common variations of the dynamic network flow model: *Discrete time* and *continuous time*. In a *discrete time* model, flows enter and exit from vertices at integer time steps $t = 0, 1, \dots, T$. For a given edge $e = (u, v) \in E$, we may view the cost $c(e)$ as the time that is required to pass an amount of flow (not exceeding the capacity) from the tail u to the head v of the edge e . Therefore, we may interpret a (static) flow network \mathcal{N} as a dynamic one without any change of notations. In the closely related *continuous time* model, which we do not use in this paper, a *flow rate* is assigned to each edge, designating how fast a unit of flow can pass through the edge. The constraints imposed in the static network flow model generally apply to dynamic network flow models, except that dynamic network flow further requires that at any time, the flow passing through any edge cannot exceed the edge capacity.

Given a dynamic flow network, a question similar to the single-commodity maximum flow problem is the following: Starting at $t = 0$, what is the maximum units of flow the can reach the sinks on or before time $t = T$? It turns out that this problem can be solved using static flow algorithms such as Edmonds-Karp [9] over a *time-expanded network*. For example, given the dynamic flow network in Fig. 2(a), its time-expanded network with $T = 4$ is given in Fig. 2(b). To compute a flow over the time expanded network, we first add a *super source* and connect it using

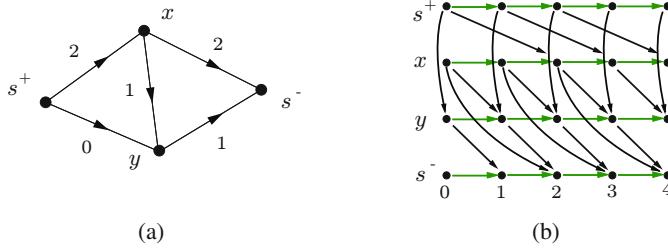


Fig. 2 a) The (static) flow network with source s^+ and sink s^- . The numbers on the edges are the costs/time delay for passing through these edges. We may assume that the capacities are all unit capacities. b) The time-expanded network with 5 copies of the original vertices ($T = 4$). All edges have unit capacity. There is a forward edge between two vertices u and v at time steps t and t' , respectively (e.g. x at $t = 0$ and y at $t' = 1$), if one of the following is true: 1. $e = (u, v)$ is an edge of the static network with $c(e) = t' - t$ (the black edges, which retain the costs as $c(e)$'s); 2. u, v are the same vertex of the static network and $t' - t = 1$ (the green edges, which have unit costs). The green edges are also called *holdover* edges since traveling through a green edge is the same as the agent not actually moving.

outgoing edges to all copies of source vertices at $t = 0$, and add a *super sink* and connect all copies of sink vertices for all t to it using outgoing edges (the super source, super sink, and additional edges are not shown in Fig. 2(b)). With this construction, a static flow on the time-expanded network corresponds to a dynamic flow on the dynamic flow network.

Lemma 2. *For a sufficiently large T , a flow for a dynamic flow network \mathcal{N} is feasible if and only if the corresponding static flow on the time-expanded network of \mathcal{N} is feasible.*

A proof of Lemma 2 can be found in [12]. Note that determining a minimally sufficient T required by Lemma 2, which directly affects the running time of the resulting algorithm, is non-trivial. The standard maximum flow algorithms have time complexity depending polynomially on T and are therefore *pseudopolynomial* in general. For a special class of problems, the *quickest transshipment problem*, of which the goal is finding the quickest feasible flow for a transshipment problem over a dynamic network, a strongly polynomial time algorithm⁷ exists [19]. However, the algorithm requires calling subroutines (for example, submodular function optimization routines) that are not *combinatorial* algorithms and also has with large constant terms when it comes to asymptotic time complexity.

⁷ An algorithm is a strongly polynomial algorithm if: 1. The number of operations in the arithmetic model of computation is bounded by a polynomial in the number of integers in the input instance, and 2. The space used by the algorithm is bounded by a polynomial in the size of the input [16].

3.2 Equivalence between Multi-agent Path Planning on CUGs and Maximum Network Flow

In this subsection, we establish a reduction from the problems of our interest to multi-commodity network flow. For illustration purposes, we use the simple graph G in Fig. 3(a), with initial locations $\{s_i^+\}, i = 1, 2$ and goal locations $\{s_i^-\}, i = 1, 2$. An instance of Problem 1 is given by $(G, \{a_1, a_2\}, x_I : a_i \mapsto s_i^+, x_G : a_i \mapsto s_i^-)$. To apply maximum flow algorithms, we construct from G a time-expanded directed graph G' , part of which is shown in Fig. 3(c). We construct Fig. 3(c) as follows.

Since we cannot create an infinite time-expanded network, we need to specify the required number of time steps. For now assume that this number is some sufficiently large T (that is, if a flow with value F is achievable with an arbitrarily long time expansion, then F is also achievable with only T time steps). After fixing a T , we create $2T + 1$ copies of vertices from G , with indices $0, 1, 1', \dots$, as shown in Fig. 3(c). For each vertex $v \in G$, we denote these copies $v(0) = v(0)', v(1), v(1'), v(2), \dots, v(T)'$. For each edge $(u, v) \in G$ and time steps $t, t+1, 0 \leq t < T$, we then add the gadget shown in Fig. 3(b) between $u(t)', v(t)'$ and $u(t+1), v(t+1)$ (arrows from the gadget are omitted from Fig. 3(c) since they are too small to draw). This gadget ensures that two agents cannot travel in opposite directions on an edge in the same time step. For the gadget, we assign unit capacity to all edges, unit cost to the horizontal middle edge, and zero cost to the other four edges. To finish the construction of Fig. 3(c), for each vertex $v \in G$, we add one edge between every two successive copies (i.e., we add the edges $(v(0), v(1)), (v(1), v(1)'), \dots, (v(T), v(T)')$). These correspond to the green and blue edges in Fig. 3(c). For all green edges, we assign them unit capacity and cost; for all blue edges, we assign them unit capacity and zero cost.

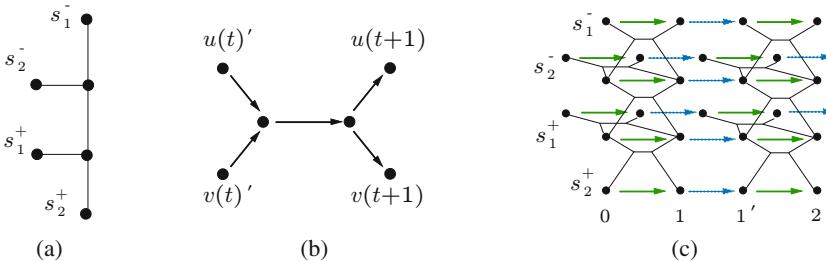


Fig. 3 a) A simple CUG G . b) A gadget for splitting an undirected edge through time steps. c) Part of the time-expanded network ($T = 2$).

The graph Fig. 3(c) is the main piece of G' , which is mostly done with the exception of the set S . We may simply let $S^+ = \{u(0) : u \in \{s_i^+\}\}$ and $S^- = \{v(T)' : v \in \{s_i^-\}\}$. That is, S^+ contains the first copies of the initial locations and S^- the last copies of the goal locations. The network $\mathcal{N}' = (G', u, c, S^+ \cup S^-)$ is now complete; we have reduced Problem 1 to an integer maximum multi-commodity flow problem on \mathcal{N}' with each agent from A as a single type of commodity.

Theorem 3. Given an instance of Problem 1 with input parameters (G, A, x_I, x_G) , there is a bijection between its solutions (with maximum number of time steps up to T) and the integer maximum multi-commodity flow solutions of flow value n on the time-expanded network \mathcal{N}' constructed from (G, A, x_I, x_G) with T time steps.

PROOF SKETCH. Fixing a T , one can establish a bijection between solutions on the original network and solutions on the time-expanded network with T time steps. \square

Since integer maximum multi-commodity flow is NP-hard, the above construction does not directly offer an efficient solution to Problem 1. Moving to Problem 2, allowing an arbitrary permutation σ to act on x_G means that we may treat all agents as a single type of commodity. Theorem 3 then implies that Problem 2 is equivalent to the quickest transshipment problem, which is solvable in polynomial time using subroutines for optimizing submodular functions. In the next section, we show that we can do better by bounding the required time steps for finding a feasible solution to Problem 2 and then apply more standard combinatorial algorithms for network flow to solve it.

4 Efficient Combinatorial Algorithms for Permutation Invariant Multi-agent Path Planning on CUGs

If we choose to apply combinatorial network flow algorithms over the time-expanded network to find solutions to Problem 2, the first priority is to determine the required number of time steps necessary to find a solution; otherwise we cannot declare that the algorithm is complete. We now provide a tight bound on T . Let (G, A, x_I, x_G) be an instance of Problem 2. We first prove some intermediate results on path sets over G . To distinguish these paths from the solution path set, denote them as $Q = \{q_1, \dots, q_n\}$. For convenience, $head(q_i)$, $tail(q_i)$, and $len(q_i)$ denote the start vertex, end vertex, and length of q_i , respectively. With a slight abuse of notation, $V(\cdot)$, $E(\cdot)$ denote the vertex set and edge set of the input parameter, which can be either a path, q_i , or a set of paths, such as Q . To start off, we want a path set Q with the following properties:

Property 1. For all $1 \leq i \leq n$, $head(q_i) \in x_I(A)$ and $tail(q_i) \in x_G(A)$. For any two paths q_i, q_j , $head(q_i) \neq head(q_j)$ and $tail(q_i) \neq tail(q_j)$.

Property 2. Each path q_i is a shortest path between $head(q_i)$ and $tail(q_i)$ on G .

Property 3. The total length of the path set Q is minimal.

Property 4. If we orient the edges of every path $q_i \in Q$ from $head(q_i)$ to $tail(q_i)$, no two paths share a common edge oriented in different directions.

Lemma 4. There exists a set of paths $Q = \{q_1, \dots, q_n\}$ that satisfies Properties 1-4.

PROOF SKETCH. Properties 1 and 2 are merely restrictions to have the initial and goal vertices paired up using shortest paths. Since we work with a discrete problem, a path set satisfying Property 3 always exists. Property 4 is implied by Property 3:

If on the contrary that two edges from two paths are oriented differently, switching destinations on those paths will reduce the total path length by two. \square

The technique from the proof of Lemma 4 can be generalized to show that oriented paths cannot form a directed cycle.

Proposition 5. *A path set Q that satisfies Properties 1-3 induces a directed acyclic graph (DAG) structure on $E(Q)$.*

A *standalone goal vertex* is a vertex $v \in x_G(A)$ such that there is a single path $q \in Q$ containing v .

Corollary 6. *A path set Q that satisfies Properties 1-3 has a standalone goal vertex.*

PROOF SKETCH. The absence of standalone vertices implies the existence of a directed cycle, contradicting Proposition 5. \square

The existence of a standalone goal vertex allows the construction of a path set which decomposes into paths that can be sequentially scheduled without colliding into each other. We characterize such a path set as one with an additional property.

Lemma 7. *There exists a path set Q satisfying Properties 1-4 and the following additional property:*

Property 5. Let $Q_i := \{q_i, \dots, q_n\}$. For any $1 \leq i \leq n$, restricting to Q_i , among all possible paths connecting an initial location (of Q_i) to a standalone goal location (of Q_i) using oriented edges from $E(Q_i)$, q_i is one shortest such.

PROOF SKETCH. Using an arbitrary shortest path set Q_0 , one can iteratively construct a new path set Q using edges of $E(Q_0)$, starting with a “furthest” standalone vertex, that satisfies Property 5. \square

If we schedule agents using a path set Q satisfying properties 1-5, there can never be cases where two agents block each other, as a direct consequence of Lemma 4. There is still the possibility that one agent blocks another. The following theorem shows that such blocking can be minimized.

Theorem 8. *Given an instance of Problem 2 with input parameters (G, A, x_I, x_G) and let ℓ be the largest pairwise distance between a member of $x_I(A)$ and a member of $x_G(A)$,*

$$\ell = \max_{\forall u \in x_I(A), v \in x_G(A)} \text{dist}(u, v). \quad (6)$$

A time-expanded network \mathcal{N}' with $T = n + \ell - 1$ is necessary and sufficient for a feasible solution to Problem 2 to exist.

PROOF SKETCH. Starting with a path set Q satisfying Properties 1-5 and letting path q_i start at time step $t = i - 1$, one can show that no collision can occur. \square

Since ℓ cannot be larger than V , the number of vertices of G , the following corollary is immediate.

Corollary 9. *For every instance of Problem 2, a feasible solution exists.*

In particular, the construction in the proof of Lemma 4 yields a complete (may not be efficient) algorithm for Problem 2. In addition to confirming that any maximum flow algorithm over the time expanded network \mathcal{N}' with $T = n + \ell - 1$ is also complete algorithm, Theorem 8 enables us to show that such algorithms are efficient.

Theorem 10. *Problem 2 is solvable using a combinatorial algorithm in strongly polynomial time.*

Using the Ford-Fulkerson algorithm [11], the time complexity is $O(nVE)$. In practice, even better running times are possible. If G is a planar graph, we have $E \sim O(V)$ and $\ell \sim O(V^{\frac{1}{2}})$. The time complexity then becomes $O(\text{MF}(n, V(n + V^{\frac{1}{2}} - 2), V(n + V^{\frac{1}{2}} - 2))) \sim O(\text{MF}(n, V(n + V^{\frac{1}{2}}), V(n + V^{\frac{1}{2}})))$. Since in our case $n < V$, Ford-Fulkerson gives us the running time $O(nV(n + V^{\frac{1}{2}})) = O(n^2V + nV^{\frac{3}{2}})$.

5 Optimal Solutions

In this section, we present optimal solutions for the permutation invariant multi-agent path planning problem. After introducing several temporal and spatial objectives of practical importance, we apply techniques from network flow to obtain optimal solutions for two of these objectives. Since these objectives are different from the basic version of Problem 2, the time bound T may be different. Lastly, we show that these objectives possess a Pareto optimal structure and they cannot be optimized simultaneously.

5.1 Optimizing over the Feasible Solutions

Having found *feasible* solutions to Problem 2, we turn the focus to the *optimality* of these solutions for practical purposes. As mentioned in Section 2, we intend to use the formulation as a model for scenarios such as multi-robot servicing. For many applications, time optimality is a top priority. Optimizing over the feasible solutions to Problem 2 (that is, we require that all goals are reached), there are two natural criteria for measuring time optimality:

Objective 1. Minimizing the average time it takes for all agents to reach their goals.

Objective 2. Minimizing the time it takes for the last agent to reach its goal.

In terms of agents (robots or people) serving requests, Objective 1 seeks to minimize the average time before a request gets served. The sufficient condition on the time bound T for this objective is given below.

Theorem 11. *There exists an optimal solution for Objective 1 in a time-expanded network with $T = (n - 1)(n - 2)/2 + V$.*

The second objective, minimizing the time that its last goal is reached, provides a lower bound on the time that is required to reach all goals. Solutions optimizing this objective are useful in providing worst servicing time estimate or guarantee. Solutions to the quickest transshipment problem [19] yield optimal solutions to this objective. However, we can avoid using submodular function optimization routines if we have a polynomial bound on T , which is provided in the following corollary of Theorem 8.

Corollary 12. *There exists an optimal solution for Objective 2 in a time-expanded network with $T = n + \ell - 1$.*

To see that Corollary 12 is true, note that $T = n + \ell - 1$ is sufficient for finding a feasible solution, which must have completion time as large as that of a solution to Objective 2. With the bound on T , running $\log T$ rounds (via binary search) of maximum flow over time-expanded network with different time horizon then gives us an optimal solution to Objective 2. The running time is then bounded by $O(MF(n, V^2, VE) \log V)$, which is strongly polynomial. In particular, with Ford-Fulkerson, the running time becomes $O(nVE \log V)$. After time optimality, another very useful solution property to optimize is the total distance traveled by the agents, i.e., spatial optimality:

Objective 3. Minimizing the total distance traveled by the agents on G .

Because we work with a CUG, if an agent a_i actually moves along path p_i between time steps t and $t + 1$, $p_i(t)$ must be different from $p_i(t + 1)$. These correspond to the black edges in the time-expanded network (see Fig. 3(b)). Thus, to optimize this objective, we can find the shortest total distance traveled by all agents via setting the cost of the holdover edge (green edges in Fig. 3(b)) to zero and then running minimum cost maximum flow algorithm over the time-expanded network. The method is again strongly polynomial, with complexity $O(V^2E \log V)$, due to the following corollary.

Corollary 13. *There exists an optimal solution for Objective 3 in a time-expanded network with $T = n + \ell - 1$.*

5.2 Pareto Optimality between the Objectives

From the discussion in the previous subsection, we observe that each of the three objectives is of practical importance. At this point, one might be tempted to seek solutions that optimize multiples of these objectives simultaneously. We show that

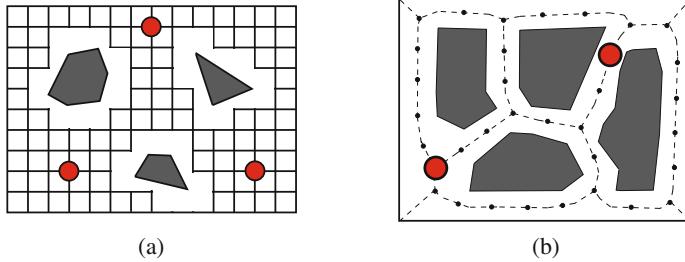


Fig. 4 a) Overlaying grid on a workspace with obstacles. b) Adapting a roadmap to obtain a graph that can be used with our multi-agent path planning algorithms.

this is not possible for each pair of these objectives. In the following theorem, we say that two objectives are *compatible* if and only if they can be optimized simultaneously. Otherwise, we say the objectives are *incompatible*.

Theorem 14. *Over the feasible solutions to Problem 2, Objectives 1-3 are pairwise incompatible.*

6 Conclusion, Future Work, and Open Problems

In this paper, we established the close link between two classes of problems: Multi-agent path planning on CUGs and network flow. Focusing on the permutation invariant versions of the multi-agent path planning problem, we proved a tight bound on the number of time steps necessary and sufficient for a feasible path set to exist in the time-expanded network, enabling efficient algorithmic solutions to these problems. We then explored optimality issues, demonstrating that the time-expansion bound generally carry over to yield strongly polynomial algorithms for optimizing two of these practical objective functions. Interestingly, each pair of these objectives cannot be optimized simultaneously.

Given our study, an immediate question or criticism is the applicability of the results to problems beyond CUGs. After all, real agents, whether robots or people, do not always live on a discrete graph. To answer this question, we have research under way that explores the idea of overlaying the CUGs on the actual workspace. That is, we may first create a roadmap over the workspace that captures the connectivity and then discretize the roadmap over which the statement of Observation 1 continues to hold (as long as the edges are close to unit length the angle between two edges is obtuse, similar version of Observation 1 can be stated) [33]. A basic solution (Fig. 4(a)) may be to put a grid on the roadmap and delete vertices inside or close to obstacles. To overcome the issue of the inherited Manhattan metric of grids, we may adapt the grid to align with the geodesics of the environment. For example, for a two dimensional workspace with polygonal obstacles, we can arrange the grid edges to follow edges of the visibility graph [27] of the environment when possible.

When clearance is tight, we may start with a maximum clearance roadmap [33] and add the vertices carefully (see Fig. 4(b)). Note that because the workspace is often two dimensional, these preparations can be computed relatively efficiently.

Many interesting open problems remain. Although finding a distance optimal solution to Problem 1 using a time-expanded network is impractical due to its intrinsic hardness, the network flow approach might still produce efficient methods that yield basic feasible solutions since the time-expanded network has a forward only structure. In addition, approximation algorithms on integer multi-commodity flow could lead to better heuristics for optimal solution search. Along this line, we only touched the most essential results in the field of network flow, which are but the tips of an iceberg. It would not be surprising that results from the vast amount of network flow literature could be readily carried over to tackle path planning problems, either along the structure we proposed in this paper or in some other forms. As an example, for Problem 2, since Objectives 1-3 are all of practical concerns but incompatible, it is desirable to seek solutions that provide performance guarantees on each of these objectives. Network flow methods, closely relate to linear programming, appear to be promising tools for such parametric optimization tasks.

Acknowledgments. The authors thank Max Katsev for double checking the proofs and the anonymous reviewers for their constructive suggestions. 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. Aronson, J.E.: A survey on dynamic network flows. *Annals of Operations Research* 20(1), 1–66 (1989)
2. Balch, T., Arkin, R.C.: Behavior-based formation control for multirobot teams. *IEEE Transaction on Robotics and Automation* 14(6), 926–939 (1998)
3. Canney, J.F.: *The Complexity of Robot Motion Planning*. MIT Press, Cambridge (1988)
4. Chalmet, L.G., Francis, R.L., Saunders, P.B.: Network models for building evacuation. *Management Science* 28(1), 86–105 (1982)
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*. MIT Press, Cambridge (2005)
6. Collins, G.E.: Quantifier Elimination for Real Closed Fields by Cylindrical Algebraic Decomposition. In: Brakhage, H. (ed.) GI-Fachtagung 1975. LNCS, vol. 33, pp. 134–183. Springer, Heidelberg (1975)
7. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: *Introduction to Algorithms*, 2nd edn. MIT Press, Cambridge (2001)
8. Costa, M.-C., Létocart, L., Roupin, F.: Minimal Multicut and Maximal Integer Multiflow: A Survey. *European Journal of Operational Research* 162, 55–69 (2005)
9. Edmonds, J., Karp, R.M.: Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM* 19(2), 248–264 (1972)
10. Erdmann, M.A., Lozano-Pérez, T.: On multiple moving objects. In: Proceedings IEEE International Conference on Robotics & Automation, pp. 1419–1424 (1986)

11. Ford, L.R., Fulkerson, D.R.: Maximal flow through a network. Research Memorandum RM-1400, The RAND Corporation (November 1954)
12. Ford, L.R., Fulkerson, D.R.: Constructing maximal dynamic flows from static flows. *Operations Research* 6, 419–433 (1958)
13. Ford, L.R., Fulkerson, D.R.: *Flows in Networks*. Princeton University Press, New Jersey (1962)
14. Fox, D., Burgard, W., Kruppa, H., Thrun, S.: A probabilistic approach to collaborative multi-robot localization. *Autom. Robots* 8(3), 325–344 (2000)
15. Goldberg, A.V., Tarjan, R.E.: A new approach to the maximum flow problem. In: STOC 1986: Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing, pp. 136–146. ACM, New York (1986)
16. Grötschel, M., Schrijver, A., Lovász, L.: Complexity, Oracles, and Numerical Computation. Springer (1988)
17. Guo, Y., Parker, L.E.: A distributed and optimal motion planning approach for multiple mobile robots. In: Proceedings IEEE International Conference on Robotics and Automation, pp. 2612–2619 (2002)
18. Halperin, D., Latombe, J.-C., Wilson, R.: A general framework for assembly planning: The motion space approach. *Algorithmica* 26(3-4), 577–601 (2000)
19. Hoppe, B., Tardos, É.: The quickest transshipment problem. *Mathematics of Operations Research* 25(1), 36–62 (2000)
20. Jennings, J.S., Whelan, G., Evans, W.F.: Cooperative search and rescue with a team of mobile robots. In: Proceedings IEEE International Conference on Robotics & Automation (1997)
21. Kant, K., Zucker, S.: Towards efficient trajectory planning: The path velocity decomposition. *International Journal of Robotics Research* 5(3), 72–89 (1986)
22. 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)
23. Kloder, S., Hutchinson, S.: Path planning for permutation-invariant multirobot formations. *IEEE Transactions on Robotics* 22(4), 650–665 (2006)
24. Latombe, J.-C.: *Robot Motion Planning*. Kluwer, Boston (1991)
25. LaValle, S.M.: Rapidly-exploring random trees: A new tool for path planning. TR 98-11, Computer Science Dept., Iowa State University (October 1998)
26. LaValle, S.M.: *Planning Algorithms*. Cambridge University Press, Cambridge (2006), <http://planning.cs.uiuc.edu/>
27. Lozano-Pérez, T., Wesley, M.A.: An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM* 22(10), 560–570 (1979)
28. Luna, R., Bekris, K.E.: Push and swap: Fast cooperative path-finding with completeness guarantees. In: Twenty-Second International Joint Conference on Artificial Intelligence, pp. 294–300 (2011)
29. Matarić, M.J., Nilsson, M., Simsarian, K.T.: Cooperative multi-robot box pushing. In: Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 556–561 (1995)
30. Miklic, D., Bogdan, S., Fierro, R., Nestic, S.: A discrete grid abstraction for formation control in the presence of obstacles. In: Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 3750–3755 (2009)
31. Nnaji, B.: *Theory of Automatic Robot Assembly and Programming*. Chapman and Hall (1992)
32. O'Donnell, P.A., Lozano-Pérez, T.: Deadlock-free and collision-free coordination of two robot manipulators. In: Proceedings IEEE International Conference on Robotics & Automation, pp. 484–489 (1989)

33. O'Dúnlaing, C., Yap, C.K.: A retraction method for planning the motion of a disc. *Journal of Algorithms* 6, 104–111 (1982)
34. Peasgood, M., Clark, C., McPhee, J.: A complete and scalable strategy for coordinating multiple robots within roadmaps. *IEEE Transactions on Robotics* 24(2), 283–292 (2008)
35. Peng, J., Akella, S.: Coordinating Multiple Robots with Kinodynamic Constraints along Specified Paths. In: Boissonnat, J.-D., Burdick, J., Goldberg, K., Hutchinson, S. (eds.) *Algorithmic Foundations of Robotics V*. STAR, vol. 7, pp. 221–237. Springer, Heidelberg (2004)
36. Poduri, S., Sukhatme, G.S.: Constrained coverage for mobile sensor networks. In: *Proceedings IEEE International Conference on Robotics & Automation* (2004)
37. Rodriguez, S., Amato, N.M.: Behavior-based evacuation planning. In: *Proceedings IEEE International Conference on Robotics and Automation*, pp. 350–355 (2010)
38. Rus, D., Donald, B., Jennings, J.: Moving furniture with teams of autonomous robots. In: *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 235–242 (1995)
39. Shucker, B., Murphrey, T., Bennett, J.K.: Switching rules for decentralized control with simple control laws. In: *American Control Conference* (July 2007)
40. Siméon, T., Leroy, S., Laumond, J.-P.: Path coordination for multiple mobile robots: A resolution complete algorithm. *IEEE Transactions on Robotics & Automation* 18(1) (February 2002)
41. Smith, B., Egerstedt, M., Howard, A.: Automatic generation of persistent formations for multi-agent networks under range constraints. *ACM/Springer Mobile Networks and Applications Journal* 14(3), 322–335 (2009)
42. Surynek, P.: An optimization variant of multi-robot path planning is intractable. In: *The Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2010)*, pp. 1261–1263 (2010)
43. Tanner, H., Pappas, G., Kumar, V.: Leader-to-formation stability. *IEEE Transactions on Robotics and Automation* 20(3), 443–455 (2004)
44. Tardos, É.: A strongly polynomial minimum cost circulation algorithm. *Combinatorica* 5(3), 247–255 (1985)
45. van den Berg, J., Overmars, M.: Prioritized motion planning for multiple robots. In: *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems* (2005)
46. van den Berg, J., Snoeyink, J., Lin, M., Manocha, D.: Centralized path planning for multiple robots: Optimal decoupling into sequential plans. In: *Proceedings Robotics: Science and Systems* (2009)
47. Švestka, P., Overmars, M.H.: Coordinated path planning for multiple robots. *Robotics and Autonomous Systems* 23, 125–152 (1998)
48. Wang, K.-H.C., Botea, A.: Tractable multi-agent path planning on grid maps. In: *Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI 2009*, pp. 1870–1875. Morgan Kaufmann Publishers Inc., San Francisco (2009)

Trajectory Planning and Assignment in Multirobot Systems

Matthew Turpin, Nathan Michael, and Vijay Kumar

Abstract. In this paper, we consider the problem of tasking large numbers of homogenous robots to move to a set of specified goal locations, addressing both the assignment and trajectory planning subproblems concurrently. This is related to the standard linear Euclidean assignment problem except that the solution to the trajectory generation subproblem must result in time-parameterized trajectories and guarantee collision avoidance. We begin with a centralized approach and derive an optimal centralized solution and study the computational complexity. The main contribution of this paper, however, is a decentralized algorithm with limited communication between neighbors that guarantees collision-avoidance and overcomes the computational challenges of the centralized method at the cost of suboptimal solutions. We demonstrate the performance of the algorithm as the number of robots is increased to tens of robots and the resulting increase in communication across neighbors required for safe execution.

1 Introduction

We consider a system with N permutation invariant robots seeking M desired goal locations in an n -dimensional Euclidean space where $N \geq M$. This paper seeks to find a computationally tractable algorithm to plan collision-free agent trajectories such that each goal location is occupied by a robot at a desired termination time. The agents' homogeneity adds an additional degree of freedom to the trajectory planning problem as compared to a system with assigned goal locations. Some applications of these homogeneous robotic systems include object manipulation [10, 3], localization [4], and satellite formation control [1]. In all these applications finding safe trajectories is

Matthew Turpin · Nathan Michael · Vijay Kumar
GRASP Laboratory, University of Pennsylvania, Philadelphia, PA
e-mail: {mturpin,nmichael,kumar}@seas.upenn.edu

critical. Indeed, for our work on aerial robots operating in close proximity [14], finding collision-free trajectories is very important since even near misses can result in aerodynamic interactions which may cause catastrophic collisions.

Since we require each goal location to be occupied by a single agent at the final time we must explicitly assign goals to robots. Fortunately, the assignment problem occurs naturally in a number of disciplines including distributed computing, operations research, as well as the problems already mentioned and there has been significant study into solutions of the assignment problem. How the cost function of this assignment is formulated determines whether it is a linear assignment or quadratic assignment problem.

For $N = M$, each possible assignment of robots to goals can be represented by an $N \times N$ permutation matrix. The space of all possible assignments for the N agent, N goal single agent to single goal is isomorphic to S^N , or the group of all permutations of every agent. To completely enumerate all possibilities requires $N!$ operations.

In general, the process of matching goals to robots is a linear assignment problem if the total cost to be minimized is the sum of individual transition costs. The usual assignment strategy for multi-robot systems is solving the linear assignment problem minimizing the sum of distance traveled [6, 13] and will be considered in detail in Sect. 3.1. The well-known Hungarian Algorithm [8] can solve the linear assignment problem in polynomial time.

Others have relaxed the linear assignment problem to find near optimal solutions. [12] uses a heuristic to minimize the sum of Euclidean distance traveled for a very simplified Euclidean linear assignment problem that grows in $\mathcal{O}(N^{\frac{5}{6}})$. [9] presents a hybrid method for the Euclidean assignment algorithm for very large numbers of robots with both a global and local approach. There has also been work in potential field algorithms [16] to solve the linear assignment problem. Other potential field methods for controlling groups of robots to goal destinations [11] or goal sets [2] have been developed. In general, these gradient descent approaches lead to unpredictable trajectories, may take very a long time to converge, and some do not guarantee all goals are satisfied. In general, decentralized auction-based algorithms can be more expensive than a centralized solution using the Hungarian algorithm when taking into consideration the cost of communications and may require a centralized auctioneer.

The quadratic assignment problem extends the notion of the linear assignment problem by adding the notion of a flow to each assignment. Exact solutions to the quadratic assignment problems are NP-hard but there are numerous suboptimal assignment algorithms for the quadratic assignment problem using Tabu search, simulated annealing, genetic algorithms, iterated locally greedy search, and many more [5].

Solutions to this problem are especially difficult when collisions have to be avoided. A centralized algorithm to create collision-free paths to solve the assignment problem for 2-dimensional robots is reduced to the numerical solution of roots of a complex polynomial in [7].

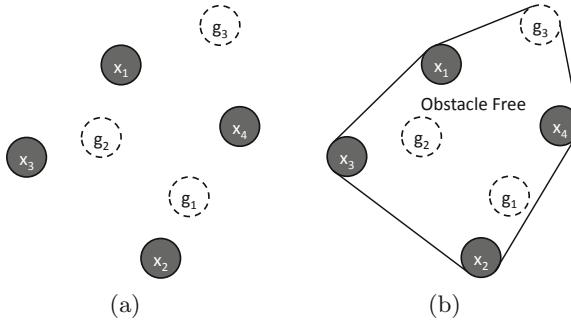


Fig. 1 For the locations of goals and agents in \mathbb{R}^2 in Fig. 1(a), the necessary obstacle free area \mathcal{K} is designated by the closed area in Fig. 1(b)

Graph search techniques like A^* or D^* that are extremely powerful for finding collision-free paths do not scale well as the dimensionality of the configuration space grows with the number of robots. The M^* algorithm [15] circumvents this difficulty by only resorting to searches in the joint space when pairs of robots are in close proximity. Our approach is similar in spirit to this work. However, we consider the continuous planning problem of interchangeable agents, and our decentralized algorithm functions online.

2 Preliminaries

We define the sets $\mathcal{N} = \{1, 2, \dots, N\}$ and $\mathcal{M} = \{1, 2, \dots, M\}$. The location of the i^{th} robot is having radius R is specified by $\mathbf{x}_i \in \mathbb{R}^n$:

$$\mathbf{x}_i(t) = [x_1(t), x_2(t), \dots, x_n(t)]^T, \forall i \in \mathcal{N}$$

Similarly, the i^{th} goal location is specified by $\mathbf{g}_i \in \mathbb{R}^n$:

$$\mathbf{g}_i = [g_1, g_2, \dots, g_n]^T, \forall i \in \mathcal{M}$$

The minimum convex operating space that must be obstacle free to solve this problem with straight line paths is defined by $\mathcal{K} \subset \mathbb{R}^n$, the Minkowski sum of the convex hull of initial locations and goal locations with ball of radius R :

$$\mathcal{K} \equiv \text{conv}(\{\mathbf{x}_i(t_0) | i \in \mathcal{N}\} \cup \{\mathbf{g}_j | j \in \mathcal{M}\}) \oplus \mathcal{B}_R \quad (1)$$

See Fig. 1 for a 2-dimensional pictorial example of \mathcal{K} .

Similar to a permutation matrix, we define a binary relation to assign goals to agents.

$$\phi : \mathcal{M} \rightarrow \mathcal{N}$$

Accordingly we let ϕ be a $N \times M$ assignment matrix so that $\phi_{i,j} = 1$ if and only if agent i is assigned to goal location j . Note that ϕ has the following properties:

$$\begin{aligned}\phi_{i,j} &\in \{0, 1\} \quad \forall i \in \mathcal{N}, j \in \mathcal{M} \\ \mathbf{1}_N^T \phi &= \mathbf{1}_M^T\end{aligned}\tag{2}$$

From this we know:

$$\phi^T \phi = I_M$$

where I_M is the $M \times M$ identity matrix. We will use the expanded assignment matrix $\Phi \equiv \phi \otimes I_n$ where \otimes signifies the Kronecker product.

We then define the Nn -dimensional state vector, $\mathbf{X} \in \mathbb{R}^{Nn}$:

$$\mathbf{X}(t) = [\mathbf{x}_1(t)^T, \mathbf{x}_2(t)^T, \dots, \mathbf{x}_N(t)^T]^T.$$

where $\mathbf{x}_i(t) \in \mathcal{K} \forall i \in \mathcal{N}$. We similarly define the stacked goal state vector $\mathbf{G} \in \mathbb{R}^{Mn}$:

$$\mathbf{G} = [\mathbf{g}_1^T, \mathbf{g}_2^T, \dots, \mathbf{g}_M^T]^T$$

We distinguish between paths and trajectories where paths are time-independent curves connecting current and final locations. This is in contrast to trajectories, which are time-parameterized planned locations of the agents. The goal of this paper is to find Nn -dimensional trajectories:

$$\gamma(t) : [t_0, t_f] \rightarrow \mathbf{X}(t),$$

where t_0 and t_f are the initial and final times respectively.

The initial position $\mathbf{x}_i(t_0) \forall i \in \mathcal{N}$ and the goal locations $\mathbf{g}_j = \sum_{i=1}^N \phi_{i,j} \mathbf{x}_i(t_f)$ have been specified so the boundary conditions are:

$$\begin{aligned}\gamma(t_0) &= \mathbf{X}(t_0) \\ \Phi^T \gamma(t_f) &= \mathbf{G}\end{aligned}\tag{3}$$

We define clearance δ as the minimum space between robots at any time during the trajectory:

$$\delta(t) = \underset{i,j \in \mathcal{N}, i \neq j}{\text{minimize}} \quad \|\mathbf{x}_i(t) - \mathbf{x}_j(t)\| - 2R$$

To ensure collision avoidance, we require the clearance to always be greater than zero:

$$\delta(t) > 0 \quad t \in [t_0, t_f]\tag{4}$$

Problem Definition

The concurrent trajectory planning and goal assignment problem involves finding a trajectory $\gamma^*(t)$ that minimizes a cost functional:

$$\begin{aligned} \gamma^*(t) = \arg \min_{\gamma(t)} & \int_{t_0}^{t_f} L(\gamma, \dot{\gamma}, t) dt \\ \text{subject to} & (2), (3), (4) \end{aligned} \quad (5)$$

Assumptions

It is useful to reiterate the assumptions (A1-A3) we have made thus far and introduce four new ones (A4-A7):

- (A1) All agents are homogeneous and interchangeable with no preference of goal destination.
- (A2) Each agent is a set of points confined to \mathcal{B}_R , a ball with radius R .
- (A3) There are no obstacles in \mathcal{K} as defined in (1).
- (A4) All agents are kinematic ($\dot{x} = u$) with no actuation error and a perfect estimate of the state.
- (A5) The initial and goal locations are spaced Δ apart. In other words, $\|x_i(t_0) - x_j(t_0)\| > \Delta$, and $\|g_i - g_j\| > \Delta$, $\forall i \neq j \in \{1, \dots, N\}$ where Δ will be defined later. Clearly $\Delta > 2R$ to ensure collision avoidance.
- (A6) All robots are capable of exchanging information about their current state and their assigned goals to other robots closer than distance $h > \Delta$. Robots with non-negligible communications time should ensure $h >> \Delta$.
- (A7) Each goal location is initially assigned to exactly one robot.

Assumptions (A1-A5) are required for the centralized algorithm that is discussed in the next section. (A6-A7) deal with the decentralized case presented in Sect. 4.

3 Centralized Algorithm

We propose two different centralized cost functions for (5) which both seek to simultaneously find an assignment matrix ϕ and collision free trajectories $\gamma(t)$ for all agents in the system such that each goal state is occupied by an agent at $t = t_f$. The centralized optimization only needs to be computed initially at $t = t_0$ as the solution for ϕ will not change over the interval $[t_0, t_f]$.

3.1 The Minimum Sum of Distances Trajectory

The first cost function we analyze is the current standard practice for the location assignment problem of minimizing the sum of distances traveled by all agents:

$$\begin{aligned} \underset{\phi, \gamma(t)}{\text{minimize}} & \sum_{i=1}^N \int_{t_0}^{t_f} \sqrt{\dot{\mathbf{x}}_i(t)^T \dot{\mathbf{x}}_i(t)} dt \\ \text{subject to} & (2), (3), (4) \end{aligned}$$

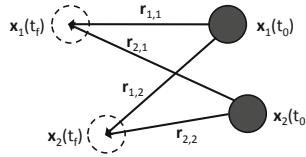


Fig. 2 For agents in n -dimensional Euclidean space ($n > 2$), if the paths collide as is the case for straight lines along $\mathbf{r}_{1,2}$ and $\mathbf{r}_{2,1}$ do, we can see that using the triangle inequality, switching assignments will always lead to collision free paths.

If we temporarily ignore clearance requirements, it is clear that the solution reduces to positive progress on straight line paths for $t \in [t_0, t_f]$. Thus, this problem reduces to:

$$\begin{aligned} \underset{\phi}{\text{minimize}} \quad & \sum_{j=1}^M \left\| \sum_{i=1}^N \phi_{i,j} \mathbf{x}_i(t_0) - \mathbf{g}_j \right\| \\ \text{subject to} \quad & (2), (3) \end{aligned} \quad (6)$$

This is the well-known transportation assignment problem. We show below in Theorem 1 that the assignment from this optimization guarantees paths that never intersect in any n -dimensional Euclidean space ($n > 2$).

Theorem 1. *The optimal assignment ϕ using the minimum sum of distance optimization in (6) results in paths that do not intersect except for the special case when a pair of agents have start and goal locations in a one dimensional space.*

Proof. Assume the paths of agents i and j intersect but do not all fall on a line. These paths necessarily exist in a plane and therefore we can reduce these two paths to an equivalent $n = 2$ problem. Since these paths intersect, we know that switching the assignment will always reduce the sum of distances by using the triangle inequality and the given ϕ is not optimal for (6). Therefore, the minimum assignment found in (6) will never result in intersecting paths. See Fig. 2 for an illustration of this proof.

Unfortunately, we can show with a simple example that agents with finite extent using the assignment from (6) are not guaranteed collision free trajectories in Fig. 3, rendering this optimization useless for real robots with physical extent.

3.2 The Minimum Velocity Trajectory

The second method we propose is to minimize the sum of the integral of velocity squared traveled by all agents:

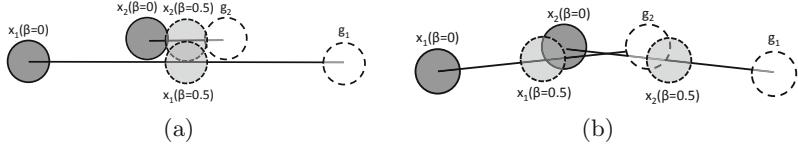


Fig. 3 For the example with two agents in Fig. 3(a) we can see that the minimum sum of distances paths (calculated by (6)) never intersect. However, having intersection-free paths does not guarantee collision-free trajectories for agents with finite size. In this case, merely switching goal assignments as shown in Fig. 3(b) does ensure collision-free trajectories.

$$\begin{aligned} & \underset{\phi, \gamma(t)}{\text{minimize}} \quad \sum_{i=1}^N \int_{t_0}^{t_f} \dot{\mathbf{x}}_i(t)^T \dot{\mathbf{x}}_i(t) dt \\ & \text{subject to} \quad (2), (3), (4) \end{aligned}$$

which is equivalent to:

$$\begin{aligned} & \underset{\phi, \gamma(t)}{\text{minimize}} \quad \int_{t_0}^{t_f} \dot{\mathbf{X}}(t)^T \dot{\mathbf{X}}(t) dt \\ & \text{subject to} \quad (2), (3), (4) \end{aligned} \quad (7)$$

The paths returned from (7) will be identical to minimizing the sum of distance traveled squared.

If it is unclear how the optimization in Sect. 3.2 differs from Sect. 3.1, consider moving a contiguous block of a number of books each with identical width to another contiguous block, but moved one book over and ignoring collisions. One solution is to move the first book to the last position, where another is to move each book one position over. Both schemes result in the same sum of distance traveled, however moving each book one unit over results in a lower sum of distances squared as a result of distance squared being a strictly convex cost function. Notice that in the many smaller moves solution, one book crossing another is unnecessary.

If we temporarily ignore the clearance requirements in (4), (7) returns an assignment matrix ϕ and trajectories:

$$\gamma^*(t) = \left(1 - \frac{t - t_0}{t_f - t_0}\right) \mathbf{X}(t_0) + \left(\frac{t - t_0}{t_f - t_0}\right) (\Phi \mathbf{G} + (I_{Nn} - \Phi \Phi^T) \mathbf{X}(t_0)) \quad (8)$$

It is clear that at $t = t_0$, (8) satisfies $\gamma^*(t_0) = \mathbf{X}(t_0)$. To verify the final boundary conditions, we can premultiply (8) by Φ^T :

$$\Phi^T \gamma^*(t) = \left(1 - \frac{t - t_0}{t_f - t_0}\right) \Phi^T \mathbf{X}(t_c) + \frac{t - t_0}{t_f - t_0} \mathbf{G}$$

to verify $\Phi^T \gamma^*(t_f) = \mathbf{G}$.

Now that we know that the trajectory will take the form in (8), we can formulate (7) as a linear assignment problem and can use an optimal assignment solving algorithm such as the Hungarian Algorithm or a suboptimal algorithm. For hundreds of robots, as shown in [5], the Hungarian Algorithm can be used to compute the exact solution of the assignment problem in under a minute on a modest computer. We now demonstrate that if we take $\Delta > 2\sqrt{2}R$, the solution to (8) will be collision free.

For notational convenience, we define:

$$\begin{aligned}\mathbf{r}_{i,j} &\equiv \mathbf{x}_j(t_f) - \mathbf{x}_i(t_0) & \mathbf{u}_{i,j} &\equiv \mathbf{x}_j(t_0) - \mathbf{x}_i(t_0) \\ \mathbf{w}_{i,j} &\equiv \mathbf{x}_j(t_f) - \mathbf{x}_i(t_f) & \beta(t) &\equiv \frac{t - t_0}{t_f - t_0} \in [0, 1]\end{aligned}$$

We first prove a lemma related to the geometry of the optimal solutions.

Lemma 1. *The optimal solutions to (7) satisfy:*

$$\mathbf{w}_{i,j}^T \mathbf{u}_{i,j} \geq 0 \quad \forall i, j \in \mathcal{N} \quad (9)$$

Proof. Since we globally minimized the sum of integrated velocity squared in (7), we know that switching goal states of agent i with agent j will not decrease the sum of distance squared, or:

$$\|\mathbf{r}_{i,i}\|^2 + \|\mathbf{r}_{j,j}\|^2 \leq \|\mathbf{r}_{i,j}\|^2 + \|\mathbf{r}_{j,i}\|^2 \quad \forall i, j \in \mathcal{N} \quad (10)$$

We then substitute:

$$\|\mathbf{r}_{i,j}\|^2 = \mathbf{r}_{i,j}^T \mathbf{r}_{i,j} = \mathbf{x}_j(t_f)^T \mathbf{x}_j(t_f) - 2\mathbf{x}_i(t_0)^T \mathbf{x}_j(t_f) + \mathbf{x}_i(t_0)^T \mathbf{x}_i(t_0)$$

into (10) and simplify:

$$(\mathbf{x}_j(t_f) - \mathbf{x}_i(t_f))^T (\mathbf{x}_j(t_0) - \mathbf{x}_i(t_0)) \geq 0 \quad \forall i, j \in \mathcal{N}$$

or

$$\mathbf{w}_{i,j}^T \mathbf{u}_{i,j} \geq 0 \quad \forall i, j \in \mathcal{N} \quad (11)$$

Theorem 2. *If $\Delta > 2\sqrt{2}R$, trajectories in (8) will satisfy (4) and be collision free.*

Proof. The analytic solution for when agents i and j ($i \neq j$) will be closest is:

$$\beta_{i,j}^* = \frac{\mathbf{u}_{i,j}^T (\mathbf{u}_{i,j} - \mathbf{w}_{i,j})}{(\mathbf{u}_{i,j} - \mathbf{w}_{i,j})^T (\mathbf{u}_{i,j} - \mathbf{w}_{i,j})}$$

If $\beta_{i,j}^*$ is outside the range $[0, 1]$, the agents are at a minimum at either the start or end of the trajectory and the agents will not collide due to (A5). If however, $\beta_{i,j}^* \in [0, 1]$, the minimum distance agent i will be from agent j is:

$$\|\mathbf{x}_i - \mathbf{x}_j\|_{\min} = \sqrt{\mathbf{u}_{i,j}^T \mathbf{u}_{i,j} - \frac{(\mathbf{u}_{i,j}^T (\mathbf{u}_{i,j} - \mathbf{w}_{i,j}))^2}{(\mathbf{u}_{i,j} - \mathbf{w}_{i,j})^T (\mathbf{u}_{i,j} - \mathbf{w}_{i,j})}} \quad (12)$$

As shown in Lemma 1, the assignment returned from (7) guarantees $\mathbf{u}_{i,j}^T \mathbf{w}_{i,j} > 0$ for all pairs of robots. Using this fact, we can see from (12) that the minimum distance possible between two robots will occur when $\mathbf{u}_{i,j}^T \mathbf{w}_{i,j} = 0$. Further if $\|\mathbf{u}_{i,j}\|$ and $\|\mathbf{w}_{i,j}\|$ are each allowed to be as small as possible which we called Δ in (A5), the minimum distance encountered using the assignment returned from (7) is:

$$\|\mathbf{x}_i - \mathbf{x}_j\|_{\min} = \frac{\Delta}{\sqrt{2}}$$

Since $\Delta > 2\sqrt{2}R$, we can rearrange and substitute to find the smallest distance between agents:

$$\begin{aligned} \sqrt{2}\|\mathbf{x}_i - \mathbf{x}_j\|_{\min} &= \Delta > 2\sqrt{2}R \\ \|\mathbf{x}_i - \mathbf{x}_j\|_{\min} &> 2R \end{aligned}$$

Thus the robots can never be in the ball of another robot and collision avoidance is guaranteed.

It should be noted that minimizing the sum of distance traveled squared arrives at the collision free assignment in Fig. 3(b).

4 Decentralized Algorithm

In this section, we exploit our knowledge of the centralized solution proposed in Sect. 3.2 to formulate a computationally-tractable, online, decentralized algorithm which will guarantee collision free paths for all robots. The decentralized method takes inspiration from (11) and is based on reassignment of goal locations to arrive at a locally-optimal solution.

As a result of robots communicating with neighboring robots within the communications range h , a moving robot can constantly be encountering new neighbors, and therefore learn new information about its neighbors, and by extension, information from its neighbors' neighbors and so on. The key feature of this algorithm is for every message sent, the system is locally minimizing a modified version of the cost functional in (7). Before we present the decentralized algorithm, Algorithm 1, we first introduce some new notation.

As there is no centralized bookkeeper, ϕ is no longer known to any one robot. Therefore we define \mathbf{f}_i as the goal currently assigned to robot i , if it exists, such that:

$$\sum_{i=1}^N \phi_{i,j} \mathbf{f}_i = \mathbf{g}_j$$

We now define the proximity set $\mathcal{C}_i(t)$ as a list of all robots within the communications range of agent i at time t :

$$\mathcal{C}_i(t) = \{j \mid \|\mathbf{x}_j(t) - \mathbf{x}_i(t)\| \leq h\} \subset \mathcal{N} \quad (13)$$

We define the update list $\mathcal{U}_i(t) \subset \mathcal{C}_i(t)$ as the list of robots to which robot i will attempt to send new information.

We will use t_c to denote the current time of computation such that $t_0 \leq t_c < t_f$. We also modify the definitions of $\mathbf{u}_{i,j}$ and $\mathbf{r}_{i,j}$:

$$\mathbf{r}_{i,j} \equiv \mathbf{x}_j(t_f) - \mathbf{x}_i(t_c) \quad \mathbf{u}_{i,j} \equiv \mathbf{x}_j(t_c) - \mathbf{x}_i(t_c)$$

In the decentralized algorithm (Algorithm 1), the i^{th} robot locally minimizes the contribution to (7) from every pair of i and j that satisfy (13):

$$\underset{\mathbf{f}_i, \mathbf{f}_j, \gamma(t)}{\text{minimize}} \quad \int_{t_c}^{t_f} \dot{\mathbf{x}}_i(t)^T \dot{\mathbf{x}}_i(t) dt + \int_{t_c}^{t_f} \dot{\mathbf{x}}_j(t)^T \dot{\mathbf{x}}_j(t) dt \quad (14)$$

The trajectory for the remaining time $[t_c, t_f]$ is computed in a similar fashion to the centralized version (8):

$$\begin{aligned} \mathbf{x}_i(t) &= \left(1 - \frac{t - t_c}{t_f - t_c}\right) \mathbf{x}_i(t_c) + \left(\frac{t - t_c}{t_f - t_c}\right) \mathbf{f}_i && \text{if } \mathbf{f}_i \text{ exists} \\ \mathbf{x}_i(t) &= \mathbf{x}_i(t_c) && \text{otherwise} \end{aligned} \quad (15)$$

We do not actively control robots without assigned goal states and as such are unconcerned with their final locations, only that the vehicles do not collide.

Note that Algorithm 1 ensures only new information is transmitted to the robots in $\mathcal{C}_i(t)$ without making unnecessary communications. Using similar reasoning to Lemma 1, we will show in Lemma 2 that Algorithm 1 converges to a locally optimal solution to (7).

Lemma 2. *A change in goal locations between any pair of robots i and j in Algorithm 1 results in a decrease of the sum of distance remaining squared.*

Proof. We show that when a pair of robots exchange goals locations, the sum of distance remaining for those two agents decreases. All other robots are unaffected by the trade so the total sum of distance remaining squared decreases for the whole system.

Case 1: Both robots have assigned goals.

After two robots trade their assigned goals, we have:

$$\mathbf{u}_{i,j}^T \mathbf{w}_{i,j} > 0$$

Using algebra similar to that in Lemma 1, we find that:

$$\|\mathbf{r}_{i,i}\|^2 + \|\mathbf{r}_{j,j}\|^2 < \|\mathbf{r}_{i,j}\|^2 + \|\mathbf{r}_{j,i}\|^2$$

Algorithm 1. Goal Assignment of Agent i

```

compute trajectory using (15)
initialize  $\mathcal{U}_i = \mathcal{C}_i(t_0)$ 
while  $t < t_f$  do
     $t_c \leftarrow t$ 
    for  $j \in \mathcal{U}_i$  do
        request  $\mathbf{x}_j(t_c)$  and  $\mathbf{f}_j$  from agent  $j$ 
        if  $\mathbf{f}_i$  exists AND  $\mathbf{f}_j$  does not exist AND  $\|\mathbf{x}_i - \mathbf{f}_i\| > \|\mathbf{x}_j - \mathbf{f}_i\|$  then
            reassign  $\mathbf{f}_i$  to  $\mathbf{f}_j$ 
            set  $\mathcal{U}_i = \mathcal{C}_i(t_c)$  and recompute trajectory using (15)
        else if  $\mathbf{f}_i$  does not exist AND  $\mathbf{f}_j$  exists AND  $\|\mathbf{x}_i - \mathbf{f}_j\| < \|\mathbf{x}_j - \mathbf{f}_i\|$  then
            reassign  $\mathbf{f}_j$  to  $\mathbf{f}_i$ 
            set  $\mathcal{U}_i = \mathcal{C}_i(t_c)$  and recompute trajectory using (15)
        else if both  $\mathbf{f}_i$  and  $\mathbf{f}_j$  exist AND  $\mathbf{u}_{i,j}^T \mathbf{w}_{i,j} < 0$  then
            exchange goal states  $\mathbf{f}_i$  and  $\mathbf{f}_j$ 
            set  $\mathcal{U}_i = \mathcal{C}_i(t_c)$  and recompute trajectory using (15)
            remove  $j$  from  $\mathcal{U}_i$ 
        if agent  $j$  requests a change of  $\mathbf{f}_i$  then
            update  $\mathbf{f}_i$ 
            set  $\mathcal{U}_i = \mathcal{C}_i(t_c)$  and recompute trajectory using (15)
            remove  $j$  from  $\mathcal{U}_i$ 
        if agent  $j$  is added to  $\mathcal{C}_i(t_c)$  then
            add agent  $j$  to  $\mathcal{U}_i$ 
        if agent  $j$  is removed from  $\mathcal{C}_i(t_c)$  then
            ensure  $j \notin \mathcal{U}_i$ 

```

In other words, the sum of remaining distance squared has decreased from the value before the reassignment and the re-planning.

Case 2: Either \mathbf{f}_i or \mathbf{f}_j don't exist. After reassignment of a goal from agent j to agent i :

$$\|\mathbf{x}_i(t_c) - \mathbf{f}_i\| < \|\mathbf{x}_j(t_c) - \mathbf{f}_i\| \quad \rightarrow \quad \|\mathbf{x}_i(t_c) - \mathbf{f}_i\|^2 < \|\mathbf{x}_j(t_c) - \mathbf{f}_i\|^2$$

After reassignment of a goal from agent i to agent j :

$$\|\mathbf{x}_j(t_c) - \mathbf{f}_j\| < \|\mathbf{x}_i(t_c) - \mathbf{f}_j\| \quad \rightarrow \quad \|\mathbf{x}_j(t_c) - \mathbf{f}_j\|^2 < \|\mathbf{x}_i(t_c) - \mathbf{f}_j\|^2$$

Thereby showing that the sum of distance remaining squared has decreased from the original assignment.

Theorem 3. Algorithm 1 results in each goal being occupied by one robot without any collisions.

Proof. Define the cost-to-go function V :

$$V = \sum_{i=1}^M \|\mathbf{x}_i - \mathbf{f}_i\|^2 \tag{16}$$

For an arbitrarily small constant, ϵ , we can use Lemma 2 and the trajectories defined in (15) to see that V is strictly decreasing:

$$V(t + \epsilon) < V(t)$$

Further, by (15), the final value of the cost-to-go function will be zero:

$$V(t_f) = 0.$$

From Theorem 2, we know all trajectories will be free of collisions.

It should be noted that for the decentralized algorithm, there exist initial conditions which, resulting from the meeting of disconnected groups of robots, could potentially result in a collision. However, there were no instances of this pathological failure mode in millions of randomly generated simulations. Instead, artificial construction was required to experience the failure. Due to the extremely low likelihood of occurrence in a real world system, we defer detailed analysis of this failure case to a future work which will present our solution to this failure modality.

5 Simulation Results

We simulate our algorithm on a large variety of boundary conditions to study its performance. For each trial, we randomly generate starting and goal locations that satisfy (A5).

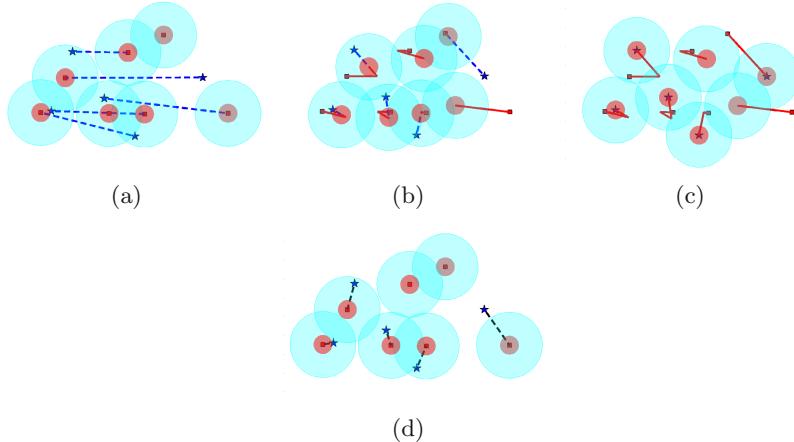


Fig. 4 Figs. 4(a)-4(c) show snapshots of a representative 2-dimensional simulation with $N = 7$, $M = 5$ and limited communications range. In Figs. 4(a)-4(c), dotted lines represent expected trajectory, solid lines represent the path followed, stars are goal locations, boxes are initial locations, and communications range h is denoted by the translucent area. For comparison, Fig. 4(d) shows the optimal solution to the centralized problem with dotted lines.

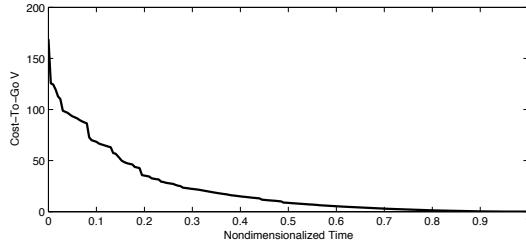


Fig. 5 Visualization of decay of the cost-to-go function V in (16) for $N = 100$, $M = 50$, and $n = 3$. The instantaneous drops are a result of reassignments and the continuous decay is a result of trajectory tracking.

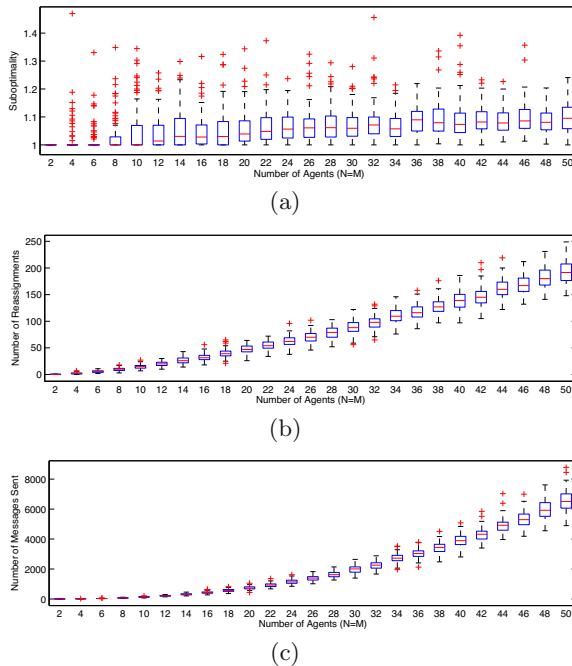


Fig. 6 Box plots for the properties of the decentralized algorithm for 100 simulated random configurations and assignments for N agents and $M = N$ goals. Figure 6(a) shows the distance traveled squared divided by the optimal value returned from the centralized solution in Sect. 3.2. The “+” marks designate statistical outliers. These plots are for $n = 3$.

The first simulation is shown in Fig. 4 and demonstrates the functioning of the on-line decentralized algorithm for $N = 7$ robots $M = 5$ goal destinations in a 2-dimensional space with a small communications range $h = 1.2\Delta$.

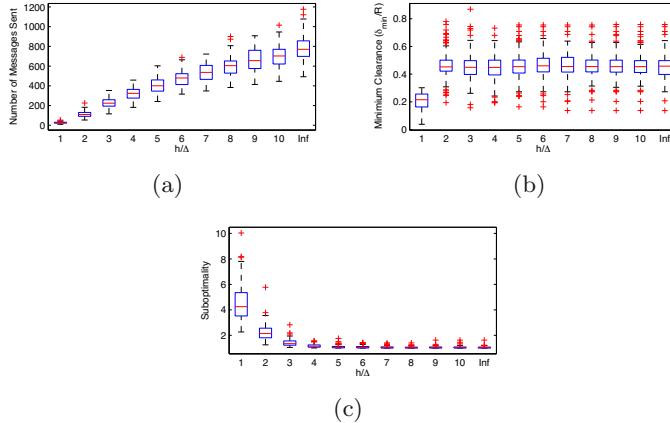


Fig. 7 Box plots for the properties of the decentralized algorithm for 100 simulated random configurations with varying communications distances with $N=M=20$. Figure 7(a) clearly shows that as the communications range decreases, the number of messages sent in the decentralized algorithm drastically decreases at the expense of clearance in Fig. 7(b) as well as the optimality of the solution in 7(c). The clearance requirement in (4) is never violated. “+” marks designate statistical outliers.

The second result shown in Fig. 5 is to verify that the energy function defined in (16) is indeed strictly decrescent by using a simulation with $N = M = 100$ in three dimensions with $h = 1.5\Delta$. It can be easily seen that $\dot{V} < 0$ and $V(t_f) = 0$.

In the third set of simulation runs we explore the scaling of the decentralized algorithm with the number of robots. In Fig. 6, we vary $N = M$ from 2 to 50 and present the result for 100 trials with $\frac{h}{\Delta} \gg 1$ in three dimensions ($n = 3$). We compare the sum of distance traveled squared to the optimal value returned from (7) using the Hungarian Algorithm. We also note that the number of reassignments increases approximately linearly in N . We can see that the total number of communications grows almost exactly as N^2 .

The final set of simulations can be seen in Fig. 7 where varied communications ranges are used for $N = M = 20$. The communications range is varied from the minimum value of $\frac{h}{\Delta} = 1$ through large values ($\frac{h}{\Delta} \gg 1$). Note that the number of messages sent decreases as the communications range decreases at the cost of becoming quite suboptimal. Additionally, we see that minimum clearance δ between robots decreases with smaller communications ranges as one might expect, but never violates the clearance requirement in (4).

6 Conclusion and Future Work

In this paper, we have addressed the problem of concurrently assigning goals and planning trajectories to the goals for a team of N robots with $M \leq N$ destinations. The centralized assignment and planning problem is well known in the literature. However, the same problem applied to robots with finite size and requirements of collision-free trajectories introduces challenges. We first develop a centralized solution to the problem of assigning goals and planning trajectories that minimize a cost functional based on the square of velocity along the trajectory and show that the resulting trajectories are globally optimal and safe. The second contribution of the paper is a decentralized algorithm that relies on the robots exchanging information about their current state and their intended goal when they are within communication range. The algorithm requires local reassignment and re-planning across a pair of robots when maximum distance traveled can be reduced while simultaneously increasing minimum clearance. We show this algorithm yields suboptimal assignments but safe trajectories. The performance of the algorithm improves with the density of the robots requiring more reassessments but with a net cost that is closer to the globally optimal cost. The computational complexity of Algorithm 1 scales very well when applied to large swarms where the number of communications is proportional to ratio of the communications radius h times the spatial density of robots. From simulations, it appears that the number of reassessments necessary scale linearly with N and the number of messages exchanged quadratically with N .

Our current work addresses incorporating the dynamics of robots and extending the cost functional L to include higher order time derivatives of trajectories. We are also interested in adapting the decentralized algorithm to consider robot failures.

References

1. Beard, R., Lawton, J., Hadaegh, F.: A coordination architecture for spacecraft formation control. *IEEE Transactions on Control Syst. Technol.* 9(6), 777–790 (2001)
2. Chaimowicz, L., Michael, N., Kumar, V.: Controlling swarms of robots using interpolated implicit functions. In: Proc. of the IEEE Int. Conf. on Robotics and Automation, pp. 2487–2492. IEEE, Barcelona (2005)
3. Das, A., Fierro, R., Kumar, V., Ostrowski, J., Spletzer, J., Taylor, C.: A vision-based formation control framework. *IEEE Transactions on Robotics and Automation* 18(5), 813–825 (2002)
4. Fox, D., Burgard, W., Kruppa, H., Thrun, S.: A probabilistic approach to collaborative multi-robot localization. *Autonomous Robots* 8(3), 325–344 (2000)
5. Gerkey, B., Matarić, M.: A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research* 23(9), 939–954 (2004)

6. Ji, M., Azuma, S., Egerstedt, M.: Role-assignment in multi-agent coordination. *Int. Journal of Assistive Robotics and Mechatronics* 7(1), 32–40 (2006)
7. Kloder, S., Hutchinson, S.: Path planning for permutation-invariant multirobot formations. *IEEE Transactions on Robotics* 22(4), 650–665 (2006)
8. Kuhn, H.: The hungarian method for the assignment problem. *Naval Research Logistics Quarterly* 2(1-2), 83–97 (1955)
9. Liu, L., Shell, D.: Multi-level partitioning and distribution of the assignment problem for large-scale multi-robot task allocation. In: Proc. of Robotics: Science and Systems, Los Angeles, CA (2011)
10. Mataric, M., Nilsson, M., Simsarin, K.: Cooperative multi-robot box-pushing. In: Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, Pittsburgh, PA, pp. 556–561 (1995)
11. Molnár, P., Starke, J.: Control of distributed autonomous robotic systems using principles of pattern formation in nature and pedestrian behavior. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics* 31(3), 433–435 (2001)
12. Rendl, F.: On the euclidean assignment problem. *Journal of Computational and Applied Mathematics* 23(3), 257–265 (1988)
13. Smith, S., Bullo, F.: Target assignment for robotic networks: Asymptotic performance under limited communication. In: Proc. of the American Control Conference, pp. 1155–1160. IEEE, New York (2007)
14. Turpin, M., Michael, N., Kumar, V.: Trajectory design and control for aggressive formation flight with quadrotors. In: Proc. of the Intl. Sym. on Robotics Research, Flagstaff, AZ (2011)
15. Wagner, G., Choset, H.: M*: A complete multirobot path planning algorithm with performance bounds. In: Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, San Francisco, CA, pp. 3260–3267 (2011)
16. Zavlanos, M., Pappas, G.: Potential fields for maintaining connectivity of mobile networks. *IEEE Transactions on Robotics* 23(4), 812–816 (2007)

k*-Color Multi-robot Motion Planning

Kiril Solovey and Dan Halperin

Abstract. We present a simple and natural extension of the *multi-robot motion planning* problem where the robots are partitioned into groups (colors), such that in each group the robots are interchangeable. Every robot is no longer required to move to a specific target, but rather to some target placement that is assigned to its group. We call this problem *k-color multi-robot motion planning* and provide a sampling-based algorithm specifically designed for solving it. At the heart of the algorithm is a novel technique where the *k*-color problem is reduced to several discrete multi-robot motion planning problems. These reductions amplify basic samples into massive collections of free placements and paths for the robots. We demonstrate the performance of the algorithm by an implementation for the case of disc robots in the plane and show that it successfully and efficiently copes with a variety of challenging scenarios, involving many robots, while a straightforward extension of prevalent sampling-based algorithms for the *k*-color case, fails even on simple scenarios. Interestingly, our algorithm outperforms a state-of-the-art implementation for the standard multi-robot problem, in which each robot has a distinct color.

1 Introduction

Motion planning is a fundamental problem in robotics and has applications in different fields such as the study of protein folding, computer

Kiril Solovey · Dan Halperin
School of Computer Science, Tel-Aviv University, Israel
e-mail: {kirilsol,danha}@post.tau.ac.il

* This work has been supported in part by the 7th Framework Programme for Research of the European Commission, under FET-Open grant number 255827 (CGL—Computational Geometry Learning), by the Israel Science Foundation (grant no. 1102/11), and by the Hermann Minkowski–Minerva Center for Geometry at Tel Aviv University.

graphics, computer-aided design and manufacturing (CAD/CAM), and computer games.

The problem of motion planning, in its most basic form, is to find a collision-free path for a robot from start to goal placements while moving in an environment cluttered with obstacles.

An obvious extension of this problem is *multi-robot motion planning*, where several robots share a workspace and have to avoid collision with obstacles as well as with fellow robots. In many situations it is natural to assume that some robots are identical, in form and in functionality, and therefore are indistinguishable. In this setting every target position should be occupied by some robot of a kind (and not necessarily by a specific robot).

We consider the problem of *k-color multi-robot motion planning*—a simple and natural extension of the multi-robot problem where the robots are partitioned into k groups (colors) such that within each group the robots are interchangeable. Every such group has a set of target positions, of size equal to the number of robots in that group. Every robot is no longer required to move to a specific target, but rather to some target position that is assigned to its group. However, we still require that all the target positions will be covered by the end of the motion of the robots. We term the special case where $k = 1$ the *unlabeled multi-robot motion planning* problem.

As an example consider a fleet of mobile robots operating in a factory that are given the task of cleaning a set of specific locations. The robots are indistinguishable from one another, and therefore any robot can be assigned to any location. Now assume that in addition to the mobile robots, another class of maintenance robots is employed by the factory; again, we consider all the maintenance robots to be of the same kind and interchangeable for the given task. This turns the unlabeled problem into a k -color problem, where $k = 2$ in this case.

From now on we will refer to the classic multi-robot motion planning problem as *fully-colored*, as it is a special case of the k -color problem where k is equal to the number of robots and every group is of size one.

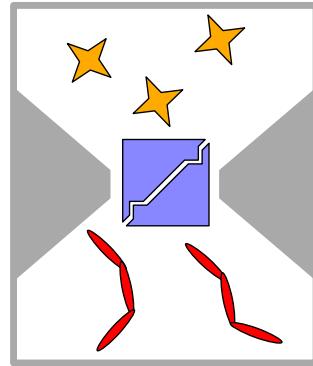


Fig. 1 An example of a 3-color scenario where three different groups of robots occupy the same workspace. The star-shaped (orange) robots are required to exchange “rooms” with the snake (red) robots while the two puzzle-like (purple) robots should return to their start positions in the end of the motion. Obstacles are drawn in gray.

1.1 Previous Work

Throughout this work we will assume some familiarity with the basic terms in the area of motion planning. For more background on motion planning, see, e.g., [6, 16].

The first efforts in motion planning in general, and the multi-robot case in particular, were aimed towards the design of *complete* algorithms, guaranteed to find a solution when one exists or report that none exists otherwise. Schwartz and Sharir were the first to give [22] a complete algorithm for a multi-robot problem, specifically dealing with the case of coordinating disc robots in the plane. The running time of their algorithm is exponential in the number of robots. A work by Hopcroft et al. [10] presented soon after suggested that in some cases the exponential running time may be unavoidable, showing that even the relatively simple setting of rectangular robots bound in a rectangular region is PSPACE-hard in the number of robots.

The hardness of the multi-robot problem involving a large number of robots can be attributed to its high number of *degrees of freedom* (or *dofs*)—the sum of the dofs of the individual robots. Some efforts were made in the direction of reducing the effective number of dofs. Aronov et al. [2] showed that for systems of two or three robots a path can be constructed, if one exists, where the robots are moved while maintaining contact, thus reducing the number of dofs by one or two, depending on the number of robots. van den Berg et al. [4] proposed a general scheme for decomposing a multi-robot problem into a sequence of subproblems, each composed of a subset of robots, where every subproblem can be solved separately and the results can be combined into a solution for the original problem. This method reduces the number of dofs that need to be treated simultaneously from the number of dofs of the entire problem to the number of dofs of the largest subproblem.

An opposite approach to the complete planners is the *decoupled* approach, trading completeness with efficiency. Decoupled algorithms solve separate subproblems (usually for individual robots) and combine the individual solutions into a global solution. Although this approach can be efficient in some cases, it does not guarantee finding a solution if one exists and usually works only for a restricted set of problems. An example of such an algorithm can be found in the work of van den Berg and Overmars in [3] where every robot is given a priority and for each robot, the motion path is constructed to avoid collision with both static obstacles and lower-priority robots that are considered as moving obstacles. In other works, as in Leroy et al. [17], individual paths are computed and velocity tuning is performed to avoid collision between robots.

In recent years the *sampling-based* approach for motion-planning problems has become increasingly popular due to its efficiency, simplicity and the fact that it is applicable to a wide range of problems. Unlike the complete planners that explicitly build the *configuration space (C-space)* of a given problem, the state of all possible configurations of a robot, sampling-based

algorithms construct an implicit representation of a robot C-space by sampling this space for valid robot placements and connecting nearby samples. The connections between samples form a *roadmap* whose vertices describe valid placements for the robot and the edges represent valid paths from one placement to the other. Due to the implicit representation of the C-space and their simplicity, sampling-based algorithms tend to be much faster than complete planners in practice, and are applicable to problems with a large number of dofs such as the multi-robot problem. Although these algorithms are not complete, most of them are *probabilistically complete*, that is, they are guaranteed to find a solution, if one exists, given sufficient amount of time. Examples of such algorithms are the PRM algorithm [11] by Kavraki et al. and the RRT algorithm [15] by Kuffner and LaValle. Such algorithms can be easily extended to the multi-robot case by considering the fleet of robots as one large composite robot [21]. Several tailor made sampling-based algorithms have been proposed for the multi-robot case [9, 25]. For more information on sampling-based algorithms see, e.g., [16].

An abstract form of the multi-robot motion planning problem is the *pebble motion on graphs problem* [14]. This is a general case of the famous *15-puzzle* [18] where pebbles occupying distinct vertices of a given graph are moved from one set of vertices to another, where the pebbles are bound to move on the edges of the graph. In [5] an unlabeled version of the pebble problem is discussed, as well as other variants, such as a grid topology of the graph. In [8] the feasibility of a k -color variant of the pebble problem on a general graph is discussed. We also mention the recent work [19] where an algorithm is given for a fairly general pebble problem.

1.2 Contribution

In this paper we present a sampling-based algorithm for the k -color problem (for any k). This algorithm is aimed to solve the most general cases of this problem and does not make any assumptions regarding the workspace or the structure of the robots.

Our algorithm for the k -color problem—the KPUMP algorithm—reduces the k -color problem to several discrete pebble problems. Specifically, a sample generated by KPUMP represents a local k -color problem that is embedded in a variant of the pebble motion problem. Those pebble problems are constructed in a manner that enables the algorithm to transform movements of pebbles into valid motions of the robots. This allows KPUMP to generate a wide range of motions and placements for the robots with minimal investigation of the configuration space, thus reducing the dependence of the algorithm on costly geometric tools such as the collision detector.

As reflected by the experiments reported below for the case of disc robots in the plane, KPUMP proves to be efficient, even on challenging scenes, and is able to solve problems involving a large number of robots using a

modest number of samples. Interestingly, it performs well even on inputs of the standard (fully-colored) multi-robot problem.

This algorithm is simple to implement and does not require special geometric components beyond single-robot local planners and single-robot collision detectors. We compare the performance of our algorithm with a variant of the PRM algorithm for the same problem that uses those components and show that the latter performs much slower than KPUMP and fails to solve even problems that are considered to be simple for KPUMP. This implies that using the same components KPUMP provides a much more powerful alternative. Moreover, concentrating on the fully-colored case, KPUMP outperforms a state-of-the-art implementation of the PRM algorithm. Our discussion will mainly focus on UPUMP—an algorithm for the unlabeled case, since its extension for the k -color case, namely KPUMP, is straightforward. The experiments though will demonstrate the power of KPUMP for various values of k .

The organization of the paper is as follows. In Section 2 we give formal definitions of the unlabeled and k -color problems. In Section 3 we present a variant of the pebble problem and discuss its properties which will be exploited by our algorithms. In Section 4 we present UPUMP and describe the changes necessary to extend it to KPUMP. In the following section we describe a subroutine that is used by UPUMP, which we call the *connection generator*. We present experimental results for the case of disc robots moving among polygonal obstacles in the plane in Section 6 and discuss the advantages of KPUMP in Section 7. For lack of space we omit a full description of KPUMP, as well as some other less crucial details. They are provided in the supplementary material [24].

2 Preliminaries and Definitions

Let r be a robot operating in the workspace W . We denote by $\mathcal{F}(r)$ the *free space* of a robot r —the collection of all collision-free *single-robot configurations*. Given $s, t \in \mathcal{F}(r)$, a *path* for r from s to t is a continuous function $\pi : [0, 1] \rightarrow \mathcal{F}(r)$, such that $\pi(0) = s, \pi(1) = t$.

Unlabeled Multi-robot Motion Planning. We say that two robots r, r' are *geometrically identical* if $\mathcal{F}(r) = \mathcal{F}(r')$. Let $R = \{r_1, \dots, r_m\}$ be a set of m geometrically identical robots, operating in a workspace W . We may use \mathcal{F} to denote $\mathcal{F}(r_i)$ for any $1 \leq i \leq m$. Let $C = \{c_1, \dots, c_m | c_i \in \mathcal{F}\}$ be a set of m single-robot configurations. C is a *configuration* if for every $c, c' \in C$, with $c \neq c'$, the robots $r, r' \in R$, placed in c, c' , do not collide. Notice that we reserve the unqualified term *configuration* to refer to a set of m collision-free single-robot configurations. Other types of configurations will be qualified single-robot configurations and pumped configurations.

Given two configurations $S = \{s_1, \dots, s_m\}, T = \{t_1, \dots, t_m\}$, named *start* and *target*, respectively, we define $\mathcal{U} = (R, S, T)$ as the *unlabeled problem*, which is shorthand for the *unlabeled multi-robot motion planning problem*. Our goal is to find an *unlabeled path* $\pi_{\mathcal{U}}$, defined as follows. Firstly, $\pi_{\mathcal{U}}$ is a collection of m paths $\{\pi_1, \dots, \pi_m\}$ such that for every i , π_i is a collision-free path for the robot r_i from s_i to *some* $t \in T$. Secondly, the robots have to remain collision-free while moving on the respective paths, i.e., for every $\theta \in [0, 1]$, $\pi_{\mathcal{U}}(\theta) = \{\pi_1(\theta), \dots, \pi_m(\theta)\}$ is a configuration. Notice that this also implies that $\pi_{\mathcal{U}}(1)$ is some permutation of T .

k -Color Multi-robot Motion Planning. The k -color problem \mathcal{L} is defined by the set $\{\mathcal{U}_1, \dots, \mathcal{U}_k\}$, where $\mathcal{U}_i = (R_i, S_i, T_i)$. The definition of the solution to this problem, namely a *k -color path*, immediately follows. A special case of this problem, usually named simply *multiple robots motion planning*, is a k -color problem where for every \mathcal{U}_i it holds that $|R_i| = 1$. In our context we call this special case *fully-colored*.

3 The Pebble Motion Problem

In preparation for the algorithm presented in the next section, we discuss a variant of the problem of *pebble motion on graphs*. This problem is a discretization of the unlabeled problem. This discretization is defined in a manner that will allow us to transform local unlabeled problems into pebble problems such that a movement of the pebbles can be transformed back into valid robot motions. We explain below where our formulation is different from the original presentation of the problem.

Formal Definition. A pebble problem [14] $\mathcal{P}(G, S, T, m)$ is defined by an undirected graph $G = (V, E)$, where $|V| = n$, and two sets of vertices $S, T \subseteq V$, where $|S| = |T| = m$. A *pebble placement* is an ordered set of m distinct vertices of V . Initially, m identical τ_1, \dots, τ_m pebbles are placed in the vertices S of V . We wish to find a chain of placements $\pi^* = P_1, \dots, P_\ell$, called a *pebble path*, which obeys the following set of rules. Firstly, we demand that $P_1 = S$. Secondly, for every two consecutive placements $P = \{p_1, \dots, p_m\}, P' = \{p'_1, \dots, p'_m\}$ and every $i \in [m]$ it holds that $(p_i, p'_i) \in E$ or $p_i = p'_i$, i.e., the pebble τ_i is allowed to stay in its current vertex or move to a neighboring vertex in the graph.

Next we depart from the problem definition in [14]. We demand that P_ℓ is some permutation of the elements of T . (The original formulation [14] specified which pebble will reside on which specific vertex of T .) We do, however, impose an additional requirement—the *separation rule*—which requires that the pebbles will move separately, i.e., for every two consecutive placements

P, P' , as defined above, exactly one pebble τ_i makes a move on an edge, while the other pebbles remain stationary. More formally, there exists $i \in [m]$ such that $(p_i, p'_i) \in E$ and for every $j \neq i$ it holds that $p_j = p'_j$. The necessity of this restriction will become clear later on.

Solvability. The variant of the pebble problem used in this paper possesses the following property, which will come in handy in the following sections. Let $\{G_1, \dots, G_h\}$ be the set of maximal connected subgraphs of G , where $G_i = (V_i, E_i)$. Given a placement V' we define the *signature* of V' as $\text{sig}(G, V') = \{|V' \cap V_i|_{i=1}^h\}$. The following lemma follows from [13, Section 3, first Lemma] where an algorithm for the case of a connected graph is given.

Lemma 1. *For every pebble problem $\mathcal{P}(G, S, T, m)$ such that $\text{sig}(G, S) = \text{sig}(G, T)$, there exists a pebble path from S to T .*

4 Algorithm for the Unlabeled Case: Pumped Configurations

In this section we present our main contribution — a sampling-based algorithm for the unlabeled problem. The algorithm, UPUMP, generates a collection of geometrically-embedded graphs. These are called *pebble graphs* and enable to map valid movements of pebbles from one pebble placement to the other on these graphs, into motions of robots between configurations. The vertices of such pebble graphs are single-robot configurations while the edges represent single-robot paths. We generate a pebble graph by sampling a set of single-robot configurations, called *pumped configurations*, of size larger than the actual number of robots, to seemingly accommodate an increased number of robots.

This technique makes use of the fact that our problem does not involve one complex robot, but rather a collection of robots operating in the same configuration space. This is in contrast with a popular sampling-based technique that considers the group of robots as one *composite robot*. In our opinion, the latter suffers from an acute disadvantage compared to our technique. We will demonstrate this claim experimentally and discuss the benefits of UPUMP and KPUMP in depth later on.

After discussing the construction of pebble graphs and exploring their various properties we show that they can be connected to generate more complex paths where the robots not only move within a single pebble graph but also between pebble graphs while maintaining collision-free paths. We conclude this section with a description of the sampling-based algorithm and state the changes required to extend UPUMP to KPUMP.

4.1 Construction of Pebble Graphs

We now define more formally some of the structures mentioned earlier. Recall that a configuration is a collection of m single-robot configurations, where m is the actual number of robots, i.e., $|R| = m$.

Definition 1. [Pumped Configuration] Let $V = \{v_1, \dots, v_n\}$ for $n \geq m$ be a set of single-robot configurations such that for every $v \in V$ it holds that $v \in \mathcal{F}$, where $\mathcal{F} = \mathcal{F}(r)$ for some $r \in R$. V is a pumped configuration if every $C \subseteq V$, such that $|C| = m$, is a configuration.

Given a pumped configuration V we construct the graph $G = (V, E)$ where the edges represent paths in \mathcal{F} for individual robots. We call it a *pebble graph*, and view it as embedded in the free configuration space. To generate the edges of G , and the respective paths, we utilize the following mechanism.

Definition 2. [Edge Planner] Given $v_i, v_{i'} \in V$ such that V is a pumped configuration and $v_i \neq v_{i'}$, the edge planner generates a path $\pi_{v_i, v_{i'}}$ or reports failure. $\pi_{v_i, v_{i'}}$ is a path for $r \in R$ from v_i to $v_{i'}$, such that for every $j \neq i, i'$ the robot r , while moving on this path, does not collide with a (geometrically identical) robot placed in v_j .

The edge planner is applied on every pair $v_i \neq v_j$ in V . Upon successful generation of a path π_{v_i, v_j} the edge (v_i, v_j) is added to G . An example of a pumped configuration, as well as its underlying graph, are given in Figure 2.

We now discuss the various properties of this special graph. We first note that every configuration $C \subset V$ is also a pebble placement for some pebble problem that is defined on G (and vice versa). A less obvious property of the pebble graph G , which is described in the following proposition, allows us to transform pebble paths into robot paths. It follows from Lemma 1 (see Section 3 for the definition of $\text{sig}(G, V)$). We mention that due to the

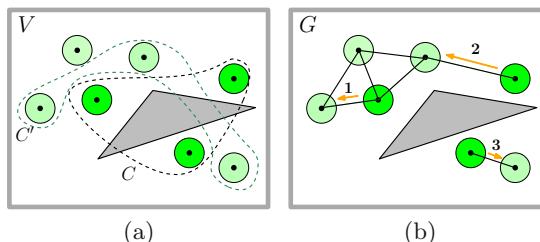


Fig. 2 (a) Pumped configuration V with $m = 3, n = 7$, for the problem of disc robots in the plane. C, C' are two configurations such that $C, C' \subset V$. (b) The pebble graph G induced by V using an edge planner that tries to connect pairs of single-robot configurations with a straight-line path. In addition, a path induced by a pebble path, from C to C' , is described, where the arrows describe the movements of the robots from one single-robot configuration to its neighbor, and the numbers indicate the order in which those movements occur.

separation rule, presented in Section 3, only movements of one robot at a time are possible.

Proposition 1. *Let $C, C' \subset V$ be two configurations such that $\text{sig}(G, C) = \text{sig}(G, C')$. Then there exists a path $\pi_{\mathcal{U}'}$ for $\mathcal{U}' = (C, C')$.*

4.2 Connecting Pebble Graphs

Proposition 1 implies that certain unlabeled problems can be solved using a single pebble graph. However, this statement does not hold for many other instances of the unlabeled problem. As an example, consider $\mathcal{U} = (S, T)$ in which there exists at least one pair $s \in S, t \in T, s \neq t$, such that a robot $r \in R$ placed in s overlaps with another robot $r' \in R$ placed in t . Thus, s, t cannot be in the same pumped configuration.

Fortunately, we can combine several graphs in order to find paths for more general unlabeled problems. For instance, robots may move from a pebble graph $G_S = (V, E)$ where $S \subset V$, through several other pebble graphs until they will finally reach $G_T = (V', E')$ where $T \subset V'$.

We first show that given two pebble graphs and an unlabeled path connecting two configurations, one from every graph, the robots can move from the first pebble graph to the second. This path serves as a “bridge” between the two graphs and connects not only the two configurations but many other configurations from the two graphs as well. Before describing a mechanism to generate such paths we provide a concrete description of the property discussed here in the form of the following lemma. We omit its proof, which is straightforward.

Lemma 2. *Let $C \subset V, C' \subset V'$ be two configurations of the pebble graphs $G = (V, E), G' = (V', E')$ and let $\pi_{C,C'}$ be a path for the unlabeled problem $\mathcal{U}' = (C, C')$. In addition, let D, D' be two configurations such that $D \subset V, D' \subset V'$ and $\text{sig}(G, D) = \text{sig}(G, C)$ and $\text{sig}(G', D') = \text{sig}(G', C')$. Then there exists a path $\pi_{\mathcal{U}''}$ for $\mathcal{U}'' = (D, D')$.*

Paths similar to $\pi_{C,C'}$ described above are generated using the following component which generalizes the component *local planner* used in standard sampling-based algorithms. A detailed algorithm for this component is given in Section 5.

Definition 3. [Connection Generator] Given two pebble graphs $G = (V, E)$, $G' = (V', E')$ and an integer q , the *connection generator* (CG) returns q unlabeled paths such that every returned path $\pi_{C,C'}$ is a solution for some unlabeled problem $\mathcal{U}' = (C, C')$ where C, C' are configurations such that $C \subset V, C' \subset V'$.

By Lemma 2, a single connection implicitly connects a collection of configurations with a specific signature from the first graph with a similar collection in the second graph. We require the CG to create several such connection in order to connect a variety of signatures between the two graphs.

4.3 Description of UPUMP

Next, we extend the result of Lemma 2 to generate still more complex paths. The UPUMP algorithm has a preprocessing phase and a query phase. In the first phase it samples a collection of pebble graphs (by sampling pumped configurations) and connects them using the CG. Those connections represent edges in a roadmap \mathcal{H} whose vertices are configurations from the different pebble graphs. Additional edges, that represent paths between configurations within the same pebble graph, are added to \mathcal{H} afterwards. In the query phase, given start and target configurations S, T , UPUMP generates two pebble graphs that contain them. These two graphs are connected to other previously sampled pebble graphs. We give a more formal description below, along with the description of the parameters used by UPUMP.

Parameters. g is the number of sampled pebble graphs; n represents the maximal size of a sampled pumped configuration; q is the maximal number of connections between two pebble graphs.

Pebble Graph Sampling. We first sample a pumped configuration by incrementally sampling single-robot configurations while discarding samples for which a robot collides with obstacles, or with another robot placed in a previously sampled position. This process stops when the number of non-colliding positions reaches n , or when the bound for the total number of samples (valid or invalid) is reached. Then, we generate the respective pebble graph by connecting pairs of single-robot configurations using the edge planner. To avoid unnecessary operations we do not connect single-robot configuration that lie in the same connected component of the current graph.

Preprocessing. UPUMP samples a collection of g pebble graphs. For every pair of sampled pebble graphs $G = (V, E), G' = (V', E')$ we apply the CG that returns at most q unlabeled paths. For every returned path $\pi_{C,C'}$ where C, C' are configurations of G, G' , respectively, C and C' are added as vertices to the roadmap \mathcal{H} together with an edge between them. To this edge the information $\pi_{C,C'}$ is attached. Then, we add edges to \mathcal{H} that represent connections that follow from Proposition 1. Specifically, we find configurations from the same pebble graph that are vertices in \mathcal{H} and share the same signature. We draw an edge between them but do not generate the respective paths at this point, as only some of them will eventually participate in a path returned in the query phase.

Query. In this phase, UPUMP is given an unlabeled problem $\mathcal{U} = (S, T)$. As S, T can be considered as pumped configurations (containing m single-robot configurations) we generate the respective pebble graphs G_S, G_T . We then connect G_S, G_T to previously sampled pebble graphs using the CG and add relevant vertices and edges to \mathcal{H} . The following corollary allows us to transform a vertex path in \mathcal{H} into a path for the unlabeled problem \mathcal{U} . It is a direct result of Lemma 2.

Corollary 1. Suppose that \mathcal{H} contains a vertex path C_0, \dots, C_ℓ such that $C_0 = S$, $C_\ell = T$, and for every $1 \leq i \leq \ell$ it holds that (C_{i-1}, C_i) is an edge in \mathcal{H} . Then there exists an unlabeled path for \mathcal{U} .

As the paths between different pebble graphs have already been constructed by the CG, we are only required to generate paths induced by pebble motions between C_{i-1} and C_i that are from the same pebble graph G with $\text{sig}(G, C_{i-1}) = \text{sig}(G, C_i)$.

4.4 Extension to KPUMP

We briefly describe some of the changes necessary to transform UPUMP to KPUMP—an algorithm for the k -color problem. KPUMP simultaneously samples several pumped configuration—each corresponds to a distinct color (namely, a different unlabeled problem). The resulting pebble graphs are constructed in a manner which prevents collision between robots from different unlabeled problems. This calls for a redefinition of the edge planner, as well as other components. Similarly to UPUMP, KPUMP construct a roadmap, only now its vertices are generalized configurations that represent non-colliding positions for all the robots in the k unlabeled problems. A more detailed description is provided in the supplementary material [24].

5 Algorithm for the Connection Generator

We describe an algorithm for the connection generator component, used by UPUMP. Recall that the CG is given two pebble graphs $G = (V, E)$, $G' = (V', E')$ and an integer q that represents the number of desired connections. Throughout this section we will use the *single-robot local planner (SLP)* mechanism, which, given two single-robot configurations $v \in V, v' \in V'$ attempts to construct a path $\pi_{v,v'}$ for a robot $r \in R$ from v to v' . In contrast with the edge planner, this mechanism does not consider the positions of the rest of the robots.

The algorithm transforms the problem of finding paths between pebble graphs into the problem of finding an *independent set* in an undirected graph. We generate the set of pairs $\mathcal{E} = \{(v, v') | v \in V, v' \in V', \pi_{v,v'} \neq \perp\}$. Namely, these are pairs of elements from V, V' for which the SLP successfully generated a path. We say that two pairs $(v, v'), (u, u') \in \mathcal{E}$ *interfere* if there exists $\theta \in [0, 1]$ such that a robot $r \in R$ placed in $\pi_{v,v'}(\theta)$ collides with another robot $r' \in R$ placed in $\pi_{u,u'}(\theta)$. Notice that every two pairs $(v, v'), (u, u') \in \mathcal{E}$, such that $v = u$ or $v' = u'$, interfere by definition. We construct the *interference graph* \mathcal{I} whose vertices are the elements of \mathcal{E} , i.e., every vertex of \mathcal{I} represents a path. We connect a pair of vertices of \mathcal{I} by an edge if they interfere.

Notice that by definition, every independent set of size m of the vertices of \mathcal{I} represents a collection of m non-colliding single-robot paths. It remains to find q such sets of size m . Even though the problem of finding an independent

set is NP-hard, in practice such sets can be found by simple greedy techniques, since \mathcal{I} is usually sparse and the number of vertices, namely $|\mathcal{E}|$, is much higher than m .

6 Experimental Results

We describe experimental results for the case of disc robots in the plane moving amidst polygonal obstacles. We show results for five challenging scenarios and compare the performance of KPUMP with two other sampling-based algorithms. Specifically we compare KPUMP with the PRM implementation of the OOPSMP package [20] on inputs of the fully-colored problem. For other inputs we use a basic sampling-based algorithm for the k -color problem called KBASIC, described later on.

KPUMP was implemented in C++ using CGAL Arrangements [7] and the Boost Graph Library (BGL) [23]. The code was tested on a PC with Intel i7-2600 3.40GHz processor with 8GB of memory, running a Windows 7 64-bit OS. For the implementation of the *edge generator* and *single-robot local planner* (used by the connection generator) a straight-line connection strategy [1] was used. This strategy attempts to move the robot along a straight line drawn between two positions.

Parameters of KPUMP. The algorithm has three parameters that affect its performance: g describes the number of the sampled pebble graphs in the UPUMP algorithm, or the number of collections of k pebble graphs in KPUMP; q is the number of connections produced by the CG between two samples; μ is the maximal number of single-robot configurations that one sample comprises. For instance, in the case of an unlabeled problem the size of every sampled pumped configuration, n , will be at most μ . The value of the latter parameter depends on the input problem. For unlabeled problems, increasing μ results in increased connectivity of the resulting pebble graphs. Thus, it will be beneficial that the pumped configurations will be as large as possible (limited by the topology of the scenario). On the other hand, in k -colored problems where $k > 1$ the value μ has to be set more carefully as an excessively high value of μ will reduce the connectivity of the pebble graphs. This stems from the fact that a single-robot path produced by the edge planner has to avoid collision with robots from other groups. Consequently, as the value of μ grows it becomes harder to connect single-robot configurations using an edge planner.

Test Scenarios. The scenarios are illustrated in Figure 3 and represent a variety of challenging problems. The unlabeled problem in (a) involves the motion of a large collection of robots. Scenarios (b) and (e) describe 2-color and 4-color problems comprising a large number of robots as well. Although scenarios (c), (d) do not involve as many robots, they are nevertheless challenging. This range of problems demonstrate the work of the

various components of the KPUMP algorithm. In the first three scenarios the resulting pebble graphs have a low number of connected components due to the low value of k (as in scenario (a)) or high clearance from obstacles (as in (e)). Therefore, large portions of the resulting paths involve the motions of the robots on paths induced by pebble problems. While the generated graphs in scenarios (c) and (d) have low connectivity, KPUMP still performs well—due to the use of the connection generator component.

The results of running KPUMP for specific parameters are given in Table 1. In addition to the parameters mentioned above, the table contains the values k for the number of colors, m the number of robots in every color and M the total number of robots. The running times are given in seconds and represent the overall duration of the preprocessing and query phases, for a single query. The

parameters used by KPUMP and other algorithms, mentioned later on, were manually optimized over a concrete set. A failure was declared when an algorithm was unable to solve a scenario for more than three runs out of five.

Comparison with Other Algorithms. The first part of the comparison involves solely inputs of the fully-colored problem. We compare KPUMP with the implementation of PRM provided by OOPSMP, which, by our experience, is very efficient. This algorithm is designed for solving fully-colored multi-robot motion planning problems. While OOPSMP required 100 seconds to solve scenario (d), KPUMP managed to solve it in 1.9 seconds. Scenario (c) proved to be even more challenging for OOPSMP, which failed to solve it, even when given 5000 seconds of preprocessing time, whereas KPUMP solved in 213.7 seconds.

In order to provide a more informative comparison, we ran both algorithms on scenarios (c),(d), only that now we increased the difficulty of these scenarios gradually—incrementally introducing the robots, i.e., starting with a single robot and adding the others one by one, as long as OOPSMP succeeded solving the new inputs in reasonable time. In this case OOPSMP was able to solve scenario (c) with five robots, while six robots was out of its reach (when given 5000 seconds of preprocessing time). The speedup of KPUMP compared to OOPSMP for this new range of scenarios is depicted in Figure 4 along with an additional test case (“decoupled-simple”), which is a simpler variant of scenario (c) with some of the obstacles removed and the radius of the robots is decreased. The latter was designed to test the performance of OOPSMP on problems involving a higher number of robots.

Table 1 Results for selected scenarios

| | Properties | | | Parameters | | | Time |
|-----|------------|-----|-----|------------|------|-------|-------|
| | k | m | M | g | q | μ | |
| (a) | 1 | 25 | 25 | 2 | 5000 | 150 | 23.2 |
| (b) | 2 | 8 | 16 | 50 | 1000 | 40 | 20.3 |
| (c) | 8 | 1 | 8 | 100 | 150 | 32 | 213.7 |
| (d) | 5 | 1 | 5 | 50 | 100 | 25 | 1.9 |
| (e) | 4 | 3 | 12 | 40 | 250 | 28 | 32.9 |

As we are not aware of any other algorithms for the k -color problem, we designed a basic algorithm to compare KPUMP with. This algorithm, which we call KBASIC, is an extension of the PRM algorithm for the k -color case. It can be described as a special case of KPUMP that samples configurations, instead of pumped configurations. The entire set of scenarios (a)-(e)(in their original form) proved to be too challenging for KBASIC, that spent at times more than ten minutes. Similarly to the previous comparison we designed a set of simple test scenarios. Specifically, scenario (e) was converted into five k -color problems for $1 \leq k \leq 5$ by partitioning the robots into k groups such that a robot number i was assigned to the group $i \bmod k$. Then, as in the previous comparison, the robots were introduced incrementally. Figure 4 depicts the speedup of KPUMP compared with KBASIC for each of the k -color problems. This shows that KPUMP outperforms KBASIC in every possible setting, be it k -color, unlabeled or fully-colored problem.

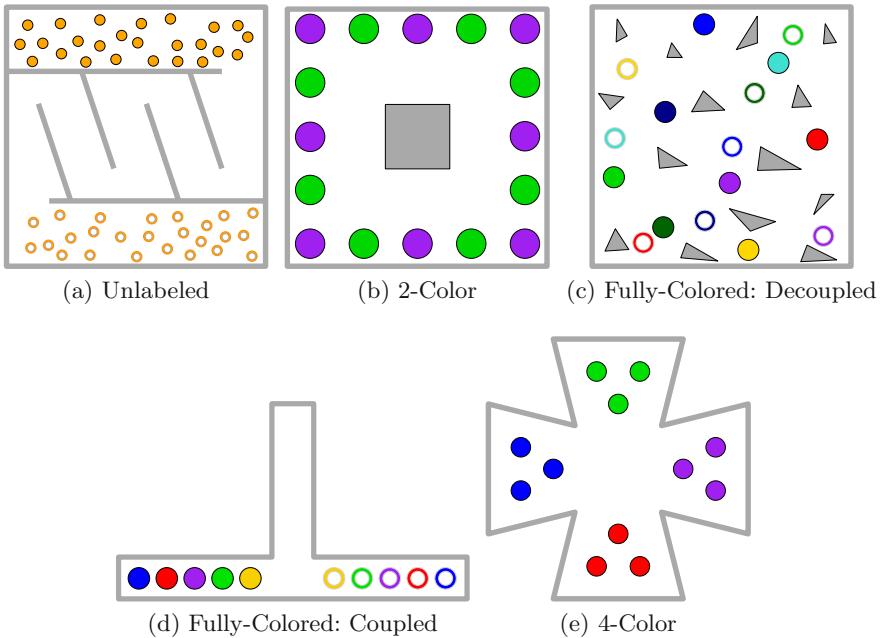


Fig. 3 [Best viewed in color] Start positions of the robots are indicated by discs while target positions are illustrated as circles in respective colors (unless otherwise indicated) (a) Unlabeled scene with twenty five robots. (b) 2-Color scene; the two groups are required to switch positions. (c) Fully-colored scene with eight robots. (d) Fully-colored scene with five robots. (e) 4-Color scene; every group has to move in a clockwise manner to the next room, e.g., blue group should move to the top room.

7 Discussion

In this section we discuss the various properties of the KPUMP algorithm and novelties it encompasses.

Shortcomings of the Composite Robot Approach. The traditional composite robot approach to the multi-robot problem treats the group of robots as one composite robot whose configuration space is the Cartesian product of the configuration spaces of the individual robots. With this approach, single-robot tools, such as sampling-based algorithms, can be used to solve multi-robot problems. For instance, this technique is used in the software packages OOPSMP and OMPL [12, 20] where PRM is applied to the fully-colored problem, and in the KBASIC algorithm discussed above. Paths generated by this approach usually force the robots to move simultaneously from one placement to the other, where none of the robots remains in the same position while the others are moving.

We believe that such paths are unnatural in the multi-robot setting and are more difficult to produce than paths that involve motion of only few robots at a time. Given collision-free placements for all the robots it is usually possible to move some of the robots to different placements without altering the placements of the rest of the robots, i.e., those robots remain still. For instance, consider a configuration $C = \{c_1, \dots, c_m\}$ for some unlabeled problem \mathcal{U} . Unless the workspace is extremely tight, another configuration C' can be derived from C where c_1 is changed to c'_1 . Moreover, connecting two such configurations by a path requires only a single-robot collision-free path for which the moving robot does not collide with the other robots placed in c_2, \dots, c_m . In contrast, the connection of two “unrelated” configurations by a path imposes much harder constraints— m single-robot collision-free paths have to be created and in addition, robots moving along those paths must not collide with each other.

KPUMP utilizes this observation by restricting the movements of the robots along certain path sections—induced by pebble problems—to motions of individual robots. We mention that KPUMP does not preclude simultaneous movements of robots when necessary, specifically on path sections where

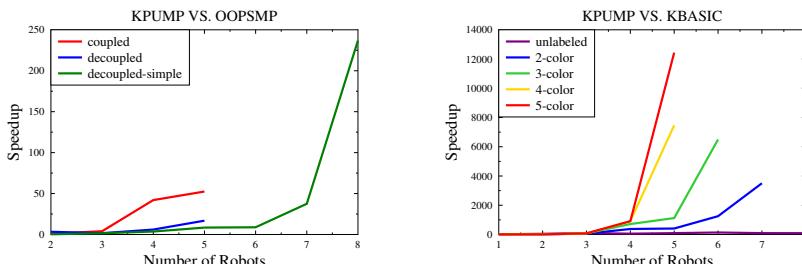


Fig. 4 [Best viewed in color] Comparing KPUMP with OOPSMP/PRM and KBASIC

the robots move from one pebble graph to the other along paths generated by the connection generator.

Amplification of Samples. Pumped configurations that are sampled by KPUMP, and the resulting pebble graphs, are fairly simple structures that require only little effort to generate. Yet, using the transformation to pebble problem, these samples are amplified to describe not only placements and paths for single robots, but also to represent an incredible amount of paths and positions for all the robots in a given problem. However, this information is not represented explicitly and only little storage space is required to represent a pebble graph. In addition, a small number of configurations must be stored. Specifically, these are configurations through which the pebble graphs connects to other graphs. Such configurations are selected by the *connection generator*. Similarly, this component does not require an explicit representation of all the configurations represented by the pebble graph. Furthermore, continuing the theme presented here that one action leads to a large number of outcomes, namely, a sample of a pumped configuration results in many configurations, a path generated by a connection generator not only connects two configurations from the two pebble graphs, but also a large number of configurations from them, which are not necessarily directly connected. Thus, these properties enable KPUMP to generate a variety of configurations and motions of the robots, using only few samples. To reproduce this variety by KBASIC one must generate far more samples.

An additional advantage of the use of pebble graphs lies in the fact that they can be connected more easily than two configurations, when a powerful component as the connection generator is at hand. Using this component, KPUMP succeeds in solving difficult scenarios even when the generated pebble graphs suffer from low connectivity, as in scenarios (c) and (d).

8 Future Work

A major question remains whether our algorithm, KPUMP, is *probabilistically complete* (see Section 1). As it is, KPUMP is rather challenging to analyze. We are currently working on a simplified version of KPUMP, which is probabilistically complete and possesses several other useful properties that we wish to investigate in the near future.

References

1. Amato, N.M., Bayazit, O.B., Dale, L.K., Jones, C., Vallejo, D.: Choosing good distance metrics and local planners for probabilistic roadmap methods. In: ICRA, pp. 630–637 (1998)
2. Aronov, B., de Berg, M., van der Stappen, A.F., Švestka, P., Vleugels, J.: Motion planning for multiple robots. In: SCG, pp. 374–382 (1998)
3. van den Berg, J., Overmars, M.: Prioritized motion planning for multiple robots. In: IROS, pp. 430–435 (2005)

4. van den Berg, J., Snoeyink, J., Lin, M., Manocha, D.: Centralized path planning for multiple robots: Optimal decoupling into sequential plans. In: RSS (2009)
5. Calinescu, G., Dumitrescu, A., Pach, J.: Reconfigurations in graphs and grids. SIAM J. Discrete Math. 22(1), 124–138 (2008)
6. Choset, H., Lynch, K., Hutchinson, S., Kantor, G., Burgard, G., Kavraki, L., Thrun, S.: Principles of Robot Motion: Theory, Algorithms, and Implementations. MIT Press (2005)
7. Fogel, E., Halperin, D., Wein, R.: CGAL Arrangements and Their Applications: A Step-by-Step Guide. In: Geometry and Computing. Springer (2012)
8. Goraly, G., Hassin, R.: Multi-color pebble motion on graphs. Algorithmica 58(3), 610–636 (2010)
9. Hirsch, S., Halperin, D.: Hybrid Motion Planning: Coordinating Two Discs Moving Among Polygonal Obstacles in the Plane. In: Boissonat, J.-D., Burdick, J., Goldberg, K., Hutchinson, S. (eds.) Algorithmic Foundation of Robotics V. STAR, vol. 7, pp. 239–255. Springer, Heidelberg (2004)
10. Hopcroft, J., Schwartz, J., Sharir, M.: On the complexity of motion planning for multiple independent objects; PSPACE-hardness of the “Warehouseman’s problem”. IJRR 3(4), 76–88 (1984)
11. Kavraki, L.E., 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)
12. Kavraki Lab: The open motion planning library, OMPL (2010), <http://ompl.kavrakilab.org>
13. Kornhauser, D.: Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. M.Sc. thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology (1984)
14. Kornhauser, D., Miller, G., Spirakis, P.: Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. In: FOCS, pp. 241–250. IEEE Computer Society (1984)
15. Kuffner, J.J., LaValle, S.M.: RRT-Connect: An efficient approach to single-query path planning. In: ICRA, pp. 995–1001 (2000)
16. LaValle, S.M.: Planning Algorithms. Cambridge University Press (2006)
17. Leroy, S., Laumond, J.P., Simeon, T.: Multiple path coordination for mobile robots: A geometric algorithm. In: IJCAI, pp. 1118–1123 (1999)
18. Loyd, S.: Mathematical Puzzles of Sam Loyd. Dover (1959)
19. Luna, R., Bekris, K.E.: Efficient and complete centralized multi-robot path planning. In: IROS (2011)
20. Plaku, E., Bekris, K.E., Kavraki, L.E.: OOPS for motion planning: An online open-source programming system. In: ICRA, pp. 3711–3716. IEEE (2007)
21. Sanchez, G., Claude Latombe, J.: Using a PRM planner to compare centralized and decoupled planning for multi-robot systems. In: ICRA (2002)
22. Schwartz, J.T., Sharir, M.: On the piano movers’ problem: III. Coordinating the motion of several independent bodies. IJRR 2(3), 46–75 (1983)
23. Siek, J.G., Lee, L.Q., Lumsdaine, A.: The Boost Graph Library User Guide and Reference Manual (2002)
24. Solovey, K., Halperin, D.: Supplementary online material, <http://acg.cs.tau.ac.il/projects/kcolor>
25. Svestka, P., Overmars, M.H.: Coordinated path planning for multiple robots. Robotics and Autonomous Systems 23, 125–152 (1998)

Distributed Construction of Truss Structures

Quentin Lindsey and Vijay Kumar

Abstract. We address the construction of truss structures with standardized parts and simple fasteners using multiple aerial robots. Robotic construction, unlike assembly in industrial settings, often occurs in unstructured environments where it may be difficult to manipulate objects to a high level of precision. This is particularly true with aerial assembly. There are challenges in positioning aerial robots and in deploying complicated manipulators or material handling devices in constrained environments characteristic of partially-built structures. We consider a scaled-down version of the problem with quadrotors, lightweight truss elements, and magnetic fasteners. Specifically, we present a provably-correct distributed construction algorithm that allows multiple robots to assemble structures according to a specified blue print with only local knowledge. In addition, we describe several heuristics for choosing the order in which parts are placed to improve performance measures like completion time. We illustrate the performance of the algorithm and the heuristics with simulations of our multi-quadrotor testbed.

1 Introduction

This paper addresses the construction and assembly of structures with autonomous rotorcrafts. Rotorcrafts are frequently used in construction work, especially for aerial lifting or transport to hard-to-access sites including downtown skyscrapers, mountainous terrain and oil rigs, or tasks requiring assembly of tall towers. However, in all these applications, aerial vehicles are operated manually.

Robotic construction has received some attention in recent years — see, for example, [5, 6, 16]. In contrast, assembly, a similar application, specifically industrial automation, is a well studied problem. Robot assembly is extensively used by automotive, electronics and packaging industries. However, unlike construction,

Quentin Lindsey · Vijay Kumar
University of Pennsylvania, Philadelphia, PA 19104
e-mail: {quentin1, kumar}@seas.upenn.edu

assembly in industrial settings is a well structured problem, where simple and reliable position and hybrid controllers can almost entirely reduce the need for noisy and unreliable sensors and complicated perception algorithms [4].

Construction, on the other hand, usually occurs in a highly unstructured environment, where the process tolerance is significantly larger. Though unstructured environments offers substantial resistance to automated solutions, the tolerances required to assemble beams are in the millimeter to centimeter range as opposed to industrial settings where assembly requires sub millimeter to micron level precision thereby allowing for larger process tolerances associated with unstructured environments. To assemble any structure in either structured or unstructured environments, basic assembly principles must be followed to guarantee the design and manufacturing of parts yield tolerances, which are compatible with the assembly plan. This process tolerance, in turn, must be matched to the capabilities of the robots and end effectors (process variation) [1].

This paper examines the construction of a class of three-dimensional truss structures, which includes scaffolds, tower cranes, power transmission towers and many other examples. Teams of autonomous aerial robots are well suited for the assembly of truss structures with appropriately designed parts and fasteners. Since designing application-specific robots may not be practical or cost effective, we consider structures that can be built with commercial aerial robots. Moreover, we design truss-like elements [3] and end effectors appropriate for aerial grasping, transport and assembly that reduce the need for precision position control.

This paper builds on the extensive literature in the areas of robotic assembly [13], robotic grasping [12], autonomous helicopters [2, 14] and modular robotics [3]. Work on multirobot assembly has been presented in [15, 16, 11] where simple robots are able to use local rules to build complex 2.5-D structures. Algorithms derived from stochastic process models of chemical reactions approaches have been developed in [10, 9]. While these are potentially more robust to failure due to external (environmental) or internal (mechanical) disturbances, the guarantees are only probabilistic and yield rates are often small. In our own previous work, we developed a deterministic construction algorithm for aerial robots that establishes guarantees for a special class of 2.5-D structures. However, due to several conservative assumptions, the rate of construction does not scale appropriately with the number of robots.

In this paper we develop a truly distributed algorithm for aerial assembly of truss-like structures by a team of robots using basic truss elements. The algorithm successfully produces feasible assembly plans for any 2.5-D structure, which do not lead to deadlock configurations. These plans allow multiple robots to work concurrently on assembly while relying only on local information. This paper provides a description and theoretical evaluation of this algorithm, several heuristics, which assess its distributive nature, and analysis of the overall system performance for multiple quadrotors.

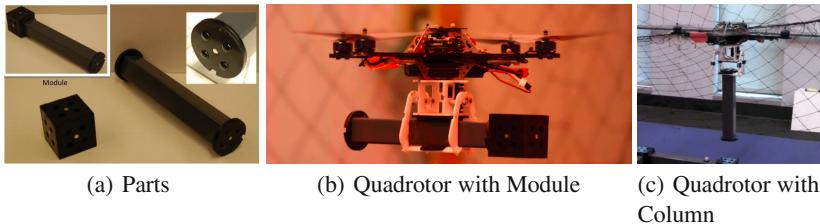


Fig. 1 Figure 1(a) shows the parts used to construct truss structures. These parts include a node (bottom left), a member (right) and a module (top left). Members can be either horizontal beams or vertical columns. Figures 1(b) and 1(c) illustrate a quadrotor carrying a module and a column, respectively.

2 Relationship to Previous Work

In this section we briefly describe our previous work [8] upon which this paper is based. We also describe the capabilities of our experimental system.

2.1 Part Design and Robotic Assembly

These truss structures are composed of two elementary components, nodes and members as shown in Fig. 1(a) similar in design to those presented in [3]. Each small cubic node can be attached to six members. Attaching a single node and member results in an additional component called a module also shown in Fig. 1(a). We will refer to members in their horizontal and vertical configuration as beams and columns, respectively. This design lends itself to the construction of cubic structures or extend to other structures like tetrahedral-octahedral structures by modifying the node design as shown later in Fig. 7(b).

In our previous work, members and modules were placed in appropriately located pallets in known positions. Quadrotor robots with specially-designed grippers were able to pick up parts, transport them to the partially-built structure and assemble parts using simple trajectory controllers. Figs. 1(b) and 1(c) show snapshots of robots carrying parts. Since robots assemble the structure, one 2-D horizontal layer or stratum at a time, from above, all assembly sites can be reached assuming the algorithm does not cause deadlocks as discussed later.

2.2 Special Cubic Structures

In [8], we describe a construction paradigm for building *Special Cubic Structures (SCSs)*. SCSs are 2.5-D structures consisting of multiple horizontal layers, each of

which is without holes, and in which each horizontal layer is supported by vertical columns. The *no holes* requirement was necessary to eliminate the need for robots to insert modules or beams in between already assembled adjacent nodes. Individual 2-D stratum of a 2.5-D structure are successively built on top of each other using the *Wave Front Raster (WFR)* algorithm. The key algorithmic contribution in [8] is the WFR algorithm which builds a 2-D stratum by recursively attaching squares to previously placed squares in raster order (bottom to top, left to right). These squares are subsequently built using 4 assembly primitives. A main tenet of the WFR algorithm is that squares are built one at a time. This constraint results in more robust construction at the cost of limiting the type of structures to those without holes.

2.3 New Contributions

In this paper, we overcome three limitations of the algorithm in [8] and extend the ideas to new cubic structures. First, we relax the assumption that planar stratum are required to be free of holes. Second, we allow for multiple quadrotors to assemble parts onto the structure concurrently. Finally, we only require local information for each quadrotor. In other words, quadrotors do not need to have global information about the state of the structure during assembly.

3 Distributed Construction

3.1 Assumptions

We consider 2.5-D truss structures built from parts shown in Fig. 1(a). 2.5-D structures are those structures where next layer is fully contained by the previous layer. The 2.5-D restriction is purely a result of the part design. We also impose the constraint that parts cannot be inserted between previously placed parts like the deadlock illustrated in Fig. 2. Finally, we assume that each 2-D layer of the 2.5-D structure is connected. If there are multiple, disconnected components, we consider each component independently.

3.2 Distributed Construction Algorithm

We propose the Distributed Assembly Truss Structure (DATS) algorithm (Alg. 1) for this problem. From Fig. 2, it becomes evident that deadlocks can be avoided by not placing independently placed nodes with unbuilt sections between them. This *Independent Placement* constraint is the central tenet of the DATS algorithm. Since each robot can select part placements that satisfies this constraint, the DATS algorithm can take advantage of the multiple build sites in a partially-assembled structure by



Fig. 2 The figure illustrates a scenario, which results in a deadlock. Since the nodes on either side of the beams have been placed, it is impossible to place the final two beams to complete the structure.

fully distributing the task among a large number of robots. Using the DATS algorithm to build each 2-D stratum, we can build 3-D structures in the same manner done in [8] by sequentially building 2-D stratum followed by a layer of vertical columns until the final structure is completed.

The challenge of avoiding deadlocks only arises in each 2-D stratum. The vertical columns do not cause deadlocks and can be placed in any order. Hence, we focus on the construction of each stratum. The DATS algorithm approaches the stratum construction problem on a *beam-by-beam* basis. During each iteration, the DATS algorithm examines the local state of each available beam. Available beams are those unbuilt beams in the current stratum that are immediately adjacent to the currently built structure. We define the adjacent beams τ_i and nodes N_i of τ_* as the six beams and nodes that are positioned relative to τ_* as shown in Fig. 3. Thus at every iteration, the DATS algorithm has a current list of available beams. Using the rules of the Next Beam algorithm (Alg. 2), the DATS algorithm determines whether the beam τ_* can be placed at all and if it should be placed as a beam τ_* or module $\tau_* N_i$. In the algorithm, A , B , and C refer to the columns and rows illustrated in Fig. 3. We denote that a beam or node is occupied $O(*)$ or empty $E(*)$ where $*$ can either be τ_i or N_i . Further, $O(*)$ also denotes that a row or column contains a node when $*$ is A , B , or C and similarly for $E(*)$. In Alg. 2, we use the notation $\&$, $|$, \oplus , and $(*)$ to represent the boolean operations *AND*, *OR*, *XOR*, and *NOT*, respectively. After a beam has been placed, a list of available beams is generated and sorted in any manner. The manner in which beams are sorted has no bearing on the correctness of the DATS algorithm and only serves to increase performance as discussed in Sec. 4.

As mentioned before, these decisions are made using information, which can be locally sensed for each part. In the context of a truss structure, the ‘local’ area of a node is the union of the row, the contiguous set of parts aligned with the x -axis, and column, those parts aligned with y -axis. (Note: Contiguous set of parts denotes adjacent parts in the blueprint emanating from a point of interest along both the x and y -axis ending at a discontinuity (hole) or the boundary of the structure.) Only those local states where the addition of a node does not violate the *Independent Placement* constraint will result in a module being placed. The only exception to this condition is Rules 5 of the Next Beam algorithms because the placement of only one beam can never result in an additional node.

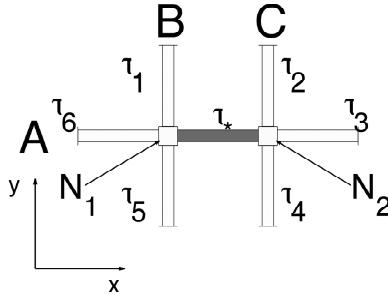


Fig. 3 This figure illustrates the arrangements of nodes and beams attached to a horizontal beam τ_* , which is being currently placed. The arrangement of nodes and beams for a vertical beam τ_* can be found by rotating the the figure by 90 degrees clockwise. This figure is used in conjunction with the rules of the Next Beam algorithm.

Algorithm 1. Distributed Assembly for Truss Structures (DATS) Algorithm (*BEAMS* is a list of unbuilt beams, which are immediately adjacent to the currently built stratum)

- 1: BEAMS \leftarrow A seed beam in the current stratum
 - 2: **while** not finished **do**
 - 3: Next Beam Algorithm $\leftarrow \tau_*$
 - 4: All beams immediately adjacent to previously placed beams \rightarrow BEAMS
 - 5: Sort BEAMS
-

Algorithm 2. Next Beam Algorithm (*A*, *B*, and *C* denote the row and columns shown in Fig. 3)

Require: τ_*

- 1: **if** $O(N_2) \& (E(B)|(O(\tau_1) \oplus O(\tau_5)))$ **then**
 - 2: Place module τ_*N_1 {Rule 1}
 - 3: **else if** $O(N_1) \& (E(C)|(O(\tau_2) \oplus O(\tau_4)))$ **then**
 - 4: Place module τ_*N_2 {Rule 2}
 - 5: **else if** $((E(C)|(O(\tau_2) \oplus O(\tau_4))) \& (O(\tau_3)|E(A)))$ **then**
 - 6: Place module τ_*N_2 {Rule 3}
 - 7: **else if** $((E(B)|(O(\tau_1) \oplus O(\tau_5))) \& (O(\tau_6)|E(A)))$ **then**
 - 8: Place module τ_*N_1 {Rule 4}
 - 9: **else if** $O(N_2) \oplus O(N_1)$ **then**
 - 10: Place module τ_* {Rule 5}
 - 11: **else if** $(E(\tau_1) \& E(\tau_2) \& E(\tau_3) \& E(\tau_4) \& E(\tau_5) \& E(\tau_6))$ **then**
 - 12: Place module τ_*N_2 {Rule 6}
-

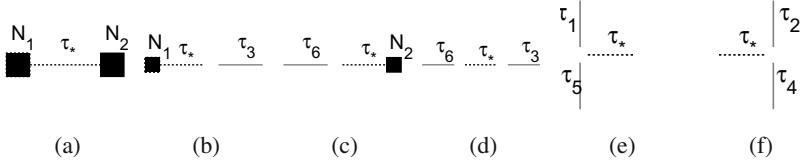


Fig. 4 Figures 4(a)-4(f) illustrate the six situations, which violate the conditions described in Eqn. 1. The dashed beam τ_* depicts an unbuilt beam, which is being considered at this iteration. Each situation has an unbuilt beam or node between previously built parts.

3.3 Algorithm Properties

We complete the discussion of the DATS algorithm by stating some properties and theoretical guarantees. One property of the DATS algorithm is that it is non-deterministic. Since the manner in which beams are sorted may not be deterministic and the order that quadrotors place parts may differ, the algorithm may not produce a unique sequence of part placements given the same initial conditions and parameters. Furthermore, the algorithm can be implemented in a decentralized manner. As long as the current state of the structure can be updated across all robots or sensed, each robot can ‘locally’ determine what action to take. With a decentralized approach, special measures must be taken to ensure that robots can coordinate which robot places a particular part. Finally, we provide some theoretical guarantees of the DATS algorithm.

Lemma 1. *The structure is always connected throughout construction*

Proof. Parts can only be attached to previously placed parts. After the first part is placed, all other parts are attached to previously placed parts. \square

Theorem 1. *The DATS algorithm can correctly place parts without causing any inconsistencies.*

Proof. We must show that the Next Beam algorithm (Alg. 2) does not cause inconsistencies i.e. deadlocks both locally and globally. We first consider the local implications of the Next Beam algorithm. In order to ensure that there are no unplaced parts between previously placed parts, we must verify that the following conditions are satisfied based on Fig. 3. Pictorially, we illustrate the six situations, which violate these conditions, in Fig. 4.

$$\frac{(O(N_1)|O(\tau_6)) \& (O(N_2)|O(\tau_3)) \& E(\tau_*)}{E(N_1) \& O(\tau_1) \& O(\tau_5)}, \quad \frac{(O(N_2)|O(\tau_2)) \& O(\tau_4)}{E(N_2) \& O(\tau_2) \& O(\tau_4)} \quad (1)$$

These conditions are trivially satisfied at the first iteration ($k = 0$) because we start with a single node. We must show that if iteration k satisfies these conditions, by building τ_* , τ_*N_1 , or τ_*N_2 . We consider the two cases in which a horizontal τ_*

shown in Fig. 3 becomes either τ_6 for a subsequent horizontal τ'_* or τ_4 for a subsequent vertical τ'_* in the vertical configuration of. It can be shown exhaustively that any action (the placement of τ_* , τ_*N_1 , or τ_*N_2) dictated by the Next Beam algorithm will result in the satisfaction of the conditions. Thus by mathematical induction, we can show that the Next Beam algorithm does not cause local inconsistencies.

Similarly, we define a global inconsistency as the independent placement of a part in a contiguous row or column, which already contains a part. Here, we also assume that this condition is satisfied at the first iteration. This proof follows directly from the Next Beam algorithm (Alg. 2). In order to cause an inconsistency, the *Independent Placement* constraint must be violated. Rules 1 – 4 of the Next Beam algorithm are the only rules capable of placing nodes in a row or column; however, they require either that there no other part exist in a row or column or that the part is attached to an existing part in the row or column in question. Thus since no row or column initially has two independently placed nodes on a single row or column, the rules cannot cause a global inconsistency. \square

Lemma 2. *Unless a 2-D stratum is finished, there is at least one part placement that can occur.*

Proof. The proof follows from Thm. 1. For a partially built structure, there is at least one unplaced beam or module. When there is exactly one unplaced part, this part can be placed using Rules 1 – 2 of the Next Beam algorithm. We must show that this part can indeed be placed by these rules. Since all prior parts have been placed using the rules of the Next Beam algorithm and these rules cover the span of all possible valid configurations of beams and nodes illustrated in Fig. 3, it must be true that any configuration of the single unplaced part can be placed by the Next Beam algorithm.

When there are two or more unplaced parts, we must show that at least one of these parts must be in a configuration, which can be completed with the Next Beam algorithm. For these cases, the unplaced beams/modules can be categorized into three groups: parts, which can be placed immediately, parts, which can be placed but not at the current moment because the placement would eventually result in a deadlock, and parts, which cannot be placed due to an inconsistency. We can exclude the latter set through Thm. 1. We must show the first set of beams is not empty, that is, there must be at least one part placement, which can occur immediately. We provide a sketch of a proof for the sake of space. See [7] for a more detailed proof. Since this set of parts cannot be placed because of an associated occupied row or column, somewhere along those rows or columns are potential sites to build. Thus, one can traverse this occupied row or column until reaching either another occupied row or column associated with an adjacent unplaceable part or a dead end is found. This dead end corresponds directly to a part belong to the first set. If no dead end is found, then the path must loop back to the original unplaceable part. We then show that these paths or loops cannot exist. \square

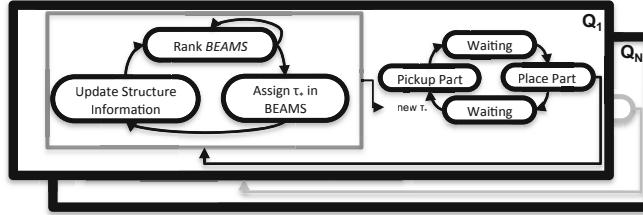


Fig. 5 Block diagram of the architecture for distributed construction

3.4 Architecture for Distributed Construction

Figure 5 depicts a conceptual model of the architecture for distributed construction used in this work. The architecture consists of an internal assembly state machine, which handles the distribution of part assignments to each quadrotor Q_i . When a quadrotor Q_i has placed a part but has not been assigned another, it request a new part assignment from its assembly state machine. This request invokes a call to a process, which implements the DATS algorithm as described in Sect. 3.2. In order to provide a valid part assignment, this process must maintain the current state of the entire structure. This current state information results in a list of potential beams. These beams are ranked according to some heuristic and assigned to a quadrotor Q_i . These heuristics will be presented in detail in Sect. 4. After the part has been assigned, the structure state is updated using internal information and external information provided by each quadrotor after it has placed a part. Independent of this process, each quadrotor Q_i also runs its a low-level state machine, which controls the individual states such as picking up parts, waiting for the tower and placing parts on the structure including transit from one state to another. For a more complete discussion of each of these states, refer to [8].

3.5 Extension to Non-cubic Lattices

Not just limited to cubic structures, the construction paradigm illustrated by the DATS algorithm can be extended to other lattices; in fact, this algorithm can be generalized to the Generalized DATS algorithm. By replacing the Next Beam algorithm with Alg. 3, we can build a larger class of 2-D stratum. This algorithm requires a small number of parameters: type, $k_{i,j}$, and $\mathbf{d}_{i,j} \in D_i$. In Fig. 6(b) we show two distinct types of arrangements for τ_* for the ‘Running Square’ lattice. For each lattice and type of arrangements of τ_* , we must specify the directions $\mathbf{d}_{i,j}$ in which we must explore to determine if a node has been previously placed. $\mathbf{d}_{i,j}$ is the unit vector of the i^{th} direction emanating from node N_j as shown in Fig. 6(a); $k_{i,j} \in \mathbb{Z}^+$ is the number of units along $\mathbf{d}_{i,j}$, which must be free of nodes. Furthermore, $\tau_{i,j}$ are the beams attached to node N_j in the direction of $\mathbf{d}_{i,j}$ while the direction associated with τ_* is \mathbf{d}_* . In Tab. 1 we list the parameters for some example lattices.

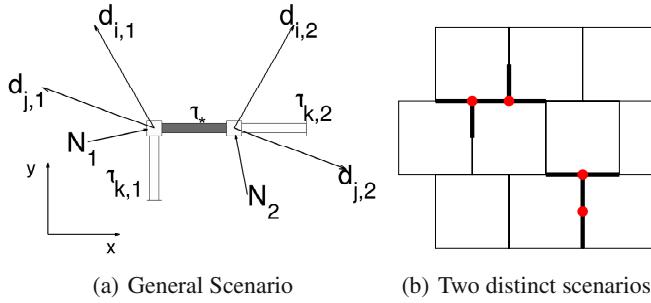


Fig. 6 Figure 6(a) show the arrangement of nodes and beams attached to a beam τ_* , which is being currently placed for an arbitrary lattice. This figure is used in conjunction with the rules of Alg. 3. Figure 6(b) illustrates the ‘Running Square’ lattice with two such distinct arrangements of τ_* .

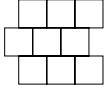
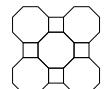
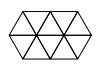
Algorithm 3. Generalized Next Beam Algorithm ($A_i = \{j | \mathbf{d}_{i,j} \in D_i\}$, $H_i = \{j | \mathbf{d}_{i,j} \parallel \mathbf{d}_*\}$.)

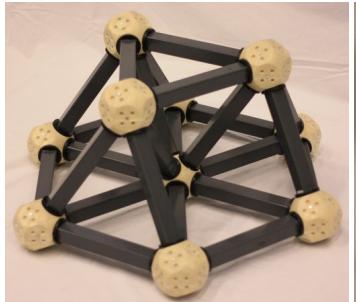
Require: τ_*

- 1: **if** $O(N_2) \& \left(\bigcap_{i \in A_1} E(k_{i,1}\mathbf{d}_{i,1}) | O(\tau_{i,1}) \right)$ **then**
 - 2: Place module τ_*N_1 {Rule 1}
 - 3: **else if** $O(N_1) \& \left(\bigcap_{i \in A_2} E(k_{i,2}\mathbf{d}_{i,2}) | O(\tau_{i,2}) \right)$ **then**
 - 4: Place module τ_*N_2 {Rule 2}
 - 5: **else if** $\left(\bigcap_{i \in A_1 \setminus H_1} E(k_{i,1}\mathbf{d}_{i,1}) | O(\tau_{i,1}) \right) \& \left(\bigcup_{i \in H_1} O(\tau_{i,1}) | E(k_{i,1}\mathbf{d}_{i,1}) \right)$ **then**
 - 6: Place module τ_*N_1 {Rule 3}
 - 7: **else if** $\left(\bigcap_{i \in A_2 \setminus H_2} E(k_{i,2}\mathbf{d}_{i,2}) | O(\tau_{i,2}) \right) \& \left(\bigcup_{i \in H_2} O(\tau_{i,2}) | E(k_{i,2}\mathbf{d}_{i,2}) \right)$ **then**
 - 8: Place module τ_*N_2 {Rule 4}
 - 9: **else if** $O(N_1) | O(N_2)$ **then**
 - 10: Place module τ_* {Rule 5}
 - 11: **else if** $\left(\bigcap_{i \in A_1} E(\tau_{i,1}) \right) \& \left(\bigcap_{i \in A_2} E(\tau_{i,2}) \right)$ **then**
 - 12: Place module τ_*N_2 {Rule 6}
-

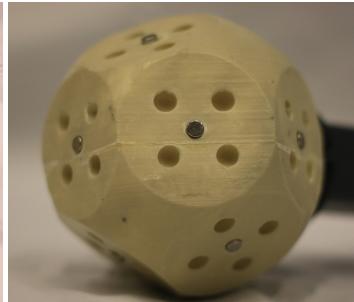
We use this Generalized DATS algorithm to build a special case of tetrahedral-octahedral lattice called the gyrate tetrahedral-octahedral lattice shown in Figure 7(a). Figure 7(a) illustrates that the 2-D stratum is composed of triangles that can be built using the parameters in Tab. 1. This extension also requires a modification of the experimental testbed described in [8]. A new node and modified gripper were designed to accommodate the 12 members (Fig. 7(b)) and to rotate members into the inclined configuration, respectively.

Table 1 Parameters for the Generalized Next Beam Algorithm for multiple lattices

| Lattice | Type | $k_{i,1}$ | $\mathbf{d}_{i,1}$ | $k_{i,2}$ | $\mathbf{d}_{i,2}$ |
|--------------------|--|----------------|----------------------|----------------|----------------------|
| Cubic |   | \mathbb{Z}^+ | (0, 1) | \mathbb{Z}^+ | (0, 1) |
| | | \mathbb{Z}^+ | (-1, 0) | \mathbb{Z}^+ | (1, 0) |
| | | \mathbb{Z}^+ | (0, -1) | \mathbb{Z}^+ | (0, -1) |
| Hexagon |   | 1 | (- cos 60, sin 60) | 1 | (cos 60, sin 60) |
| | | 1 | (- cos 60, - sin 60) | 1 | (cos 60, - sin 60) |
| Running Square |     | \mathbb{Z}^+ | (1, 0) | 1 | (0, -1) |
| | | \mathbb{Z}^+ | (-1, 0) | | |
| | | \mathbb{Z}^+ | (-1, 0) | \mathbb{Z}^+ | (1, 0) |
| | | 2 | (0, -1) | 2 | (0, 1) |
| Octagon and Square |   | 1 | (0, 1) | 1 | (0, -1) |
| | | 1 | (-1, 0) | 1 | (1, 0) |
| | | 1 | (- cos 45, sin 45) | 1 | (cos 45, sin 45) |
| | | 1 | (0, -1) | 1 | (0, -1) |
| Triangle |   | \mathbb{Z}^+ | (cos 60, sin 60) | \mathbb{Z}^+ | (- cos 60, sin 60) |
| | | \mathbb{Z}^+ | (- cos 60, sin 60) | \mathbb{Z}^+ | (cos 60, sin 60) |
| | | \mathbb{Z}^+ | (-1, 0) | \mathbb{Z}^+ | (1, 0) |
| | | \mathbb{Z}^+ | (- cos 60, - sin 60) | \mathbb{Z}^+ | (cos 60, - sin 60) |
| | | \mathbb{Z}^+ | (cos 60, - sin 60) | \mathbb{Z}^+ | (- cos 60, - sin 60) |



(a) Tetrahedral-octahedral Lattice



(b) Spherical Node

Fig. 7 Figure (a) depicts an example of a structure with a tetrahedral-octahedral lattice. Figure (b) is a spherical node with 12 attachment faces for a tetrahedral-octahedral lattice structure.

4 Efficient Structure Expansion

An important aspect of the DATS algorithm outlined in Sec. 3.2 is the manner of sorting beams, which has not been discussed; however, this sorting order or

expansion methods plays a key role. Since the taxonomy of structures is vast, a single technique for sorting cannot possibly be optimal for every type of structure except for an optimization method, which become intractable for large number of robots and parts.

We seek to understand how expansion methods influence characteristics of the construction problem. One aspect is the inactivity of robots during construction. For scenarios where the number of robots N and the number of parts required for a structure P are mismatch, namely $N > P$, significant portion of robots will be inactive and thus unneeded. Likewise, when $N \approx P$, robots may still spend unreasonable time waiting. Additionally, increasing the number of available robots appears to minimize completion time, but this may not be so straightforward.

We examine the effect of three such expansion method: *randomized*, *directional*, and *depth first*. We will define three sets of beams: Beams in transit (beams, which have been assigned to a quadrotor, but have yet to be placed), Beams considered (beams that are immediately adjacent to the currently built structure), and Beams to be placed (beams, which have not been placed and are not in transit or being considered). It is important to note that Beams considered are all valid selections and will result in the correct construction of the structure. The first method randomly sorts Beams considered, resulting in a random expansion of the structure. The second method utilizes a directional approach by sorting beams preferentially that are parallel to the x -axis and then beams parallel to the y -axis. Subsequently, beams are sorted from bottom to top and from left to right.

The final expansion method attempts to expand to the frontier of the structure as quickly as possible. Every beam in Beams considered is sorted by the largest distance between it and a corresponding frontier beam in Beams considered is found. Furthermore, since the placement of parts depends on whether other parts are placed, we minimize these dependencies by sorting parts by the number of dependencies and the time in which the latest dependency was assigned. The latter sorting criteria reduces the probability that a dependency is in transit.

5 Simulations and Results

DATS algorithm (§ 3.2) and the expansion methods discussed in the previous section provide a provably correct means of constructing truss stuctures. In this section, we evaluate the performance of the DATS algorithm and the expansion methods using a numerical simulation of the experimental setup described in [8]. We chose to simulate the construction because the evaluation of the robustness of the individual behaviors (picking up parts, waiting for the structure, placing parts and transit among the behaviors) is identical to results presented in [8].

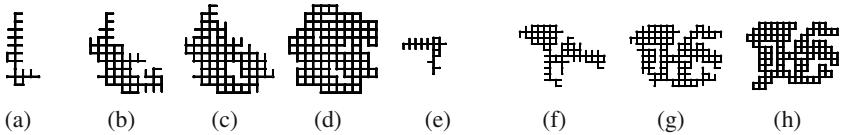


Fig. 8 Snapshots of a dense structure (Figs. 8(a)-8(d)) and a sparse structure (Figs. 8(e)-8(h)) at several instances during two simulations. Figs. 8(d) and 8(e) illustrate the target dense and sparse structure. The number of parts in each structure is similar as well as the number of internal holes.

5.1 Simulation Environment

The simulation environment incorporates parameters obtained during previous experiments. The particular parameters relevant for this simulation are the average durations necessary to pickup a part and to place it. The transit time component of the assembly task is integrated in these times. For this experiment, we assume that the pickup duration has a mean $\mu_{Pick} = 8.5$ secs and standard deviation $\sigma_{Pick} = 1.0$ secs while the place duration has mean $\mu_{Place} = 6.5$ and standard deviation $\sigma_{Place} = 1.0$ secs. Once a quadrotor is assigned a part, an individual pick up duration $\Delta_{pick,i}$ drawn from a normal distribution $\mathcal{N}(\mu_{Pick}, \sigma_{Pick})$ indicates the time duration required to transit to a part bin, pickup a part and transit to a waiting area near the parts intended attachment position. Since some parts are dependent on the presence of additional parts, each quadrotor must wait until all of the dependencies of its part have been placed on the structure even after the pick up duration $\Delta_{pick,i}$ has elapsed. Subsequently, the quadrotor is similarly assigned a place duration drawn from a normal distribution $\mathcal{N}(\mu_{Place}, \sigma_{Place})$. After the quadrotor has placed its part, it requests another part unless there are no other parts to place.

The simulation also assumes that all parts are placed without failures. In the simulation, it is assumed that the number of pickup sites M is greater or equal to N , the number of robots to ensure that no robot must wait for a free pickup site in order to evaluate the performance of each expansion method without regard to congestion at pickup sites. In addition, since we consider transit times to the part bins and assembly sites, the physical positions of said part bins and assembly sites are also immaterial. Finally, all part bins have an infinite supply of parts so the equally distribution of part retrieval among the part bins is unnecessary.

5.2 Efficient Expansion Results

Since each expansion method operates differently, it is unclear which method performs qualitatively and quantitatively better. In this section, we evaluate the performances of 4 methods using two metrics: percentage of the time spent waiting and completion time. The first two methods are the randomized and directional expansion methods as described by Sec. 4. The last two methods are variations of the depth-first expansion method. The first variation only ranks beams based on

distance from the currently structure while the second version first ranks beams by the number of dependencies of each beam, then by latest time in which a dependencies was assigned to a robot, and finally by distance. To evaluate the expansion methods across several conditions, we vary the number of robots and the type of structures, specifically dense vs. sparse. The dense and sparse structures (Figs. 8(d) and 8(h)), which were evaluated, have a similar number of parts and internal holes.

Qualitatively, the depth first methods expand the structure radially from the center while intermittently rapidly increasing the number of available beams shown by the rapid increase in available beams in Figs. 9(a) and 9(d). By trading off current construction speed for branch-like expansion, this method seemingly becomes more efficient as the simulation advances. By utilizing a bi-direction ‘wave front’, the directional method does not effectively take advantage of less dependent part orderings. Finally, the randomized approach, although unpredictable, expands in radial manner similiar to the depth first approach.

The percentage of time that a robot waits is highly dependent on the number of robots and the type of structure. In all cases, the directional method performs the worse among all conditions. For $N \geq 2$ (cases with non-zero waiting times), the waiting time for the directional method is 4.75% – 182% more than the nearest method. The waiting time of the randomized method for $N \leq 20$ outperforms the depth first method by 1.1% – 55% while for larger N , the depth first nominally performs better than the randomized method by 3.5% – 14%.

As expected, completion time is proportional to the number of robots as shown in Fig. 10. For these particular structures, the additional benefit of more robots diminishes after 50 robots, which may not be true for other structures. Although there are always available robots, at any given moment there are significantly fewer available beams seen by the data plotted in black in Fig. 9. Among the 4 methods, the variability across number of robots becomes more pronounced. For $N \leq 2$, the deviation is less than 2% due to negligible waiting time. For $\{3 \leq N \leq 20\}$, the depth first with dependency and randomized methods remain within 3.3% of each other while they both significantly outperform the directional and, to a lesser extent, the depth-first without dependency by 29% and 5.9%, respectively. For every method, completion time for $N \geq 50$ is largely the same.

6 Future Works and Conclusions

6.1 Future Works

This work provides ample research directions, which the authors plan to pursue. Among these directions, *heterogeneity* and *3-D construction* seem two important aspects that warrant further discussion.

Heterogeneity. This work focuses entirely on the constraints and capabilities of an aerial robotic team. Heterogeneous teams of construction robots (including ground robots) may increase the capabilities and overall effectiveness of the team by complementing the inherent limitations of one robot type with the increased abilities

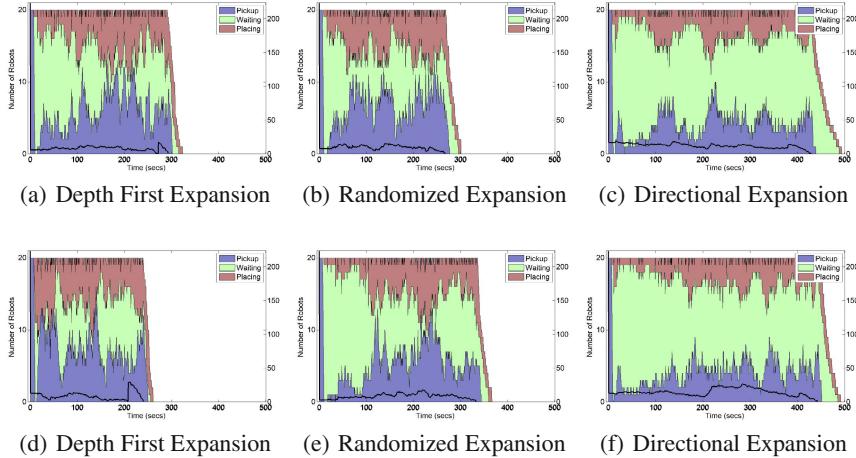


Fig. 9 Figures 9(a)-(f) show the three expansion methods applied to scenario with 20 quadrotors building a dense structure (top) and a sparse structure (bottom). The plots are further separated into the number of quadrotors in picking up parts (shaded blue), waiting (shaded green) and placing parts (shaded red). The black plot represents the number of available beam.

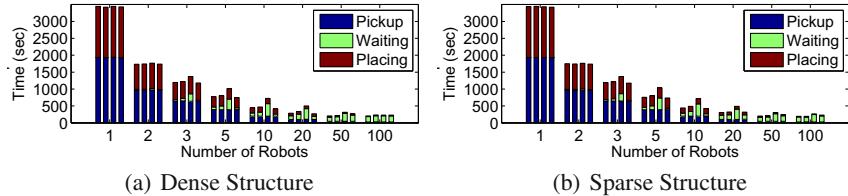


Fig. 10 Completion time for several expansion methods vs. the number of robots available. Each grouping of bars represents the depth-first method with the dependency criteria, the depth-first method without dependency, the directional method and the random approach, respectively. Each bar is further divided into the percentage of time all quadrotors are picking up parts (shaded blue), waiting (shaded green) and placing parts (shaded red).

of another type. In particular, ground robots would enable the system to move part bins thus minimizing aerial robot transit time and build structures, which require temporary scaffolding such as arches.

3-D Construction. Another limitation of this construction paradigm is the 2.5-D assembly method, which sequentially completes vertical vertical beams followed by 2-D stratum. By restricting construction to a single plane, some robots are under utilized because while 2-D stratum is being completed, no quadrotor can proceed to place vertical beams. Additionally, all structures discussed in this work and [8] have been limited to those which can be described as 2.5-D because of the limited load

bearing ability of the truss elements. By designing ‘smarter’ parts with active components such as fasteners, which are deployed automatically, we would no longer be limited to this type of structures.

6.2 Conclusions

In this work we have described a provably correct distributive algorithm for the construction of truss structures with aerial robot teams. This algorithm leverages the property that decisions only rely on local information. Additionally, we provide several heuristics for ordering assembly priorities at every update time for set of available beam. We show that a depth first approach for choosing this ordering minimizes completion time for the structure and the time each robot must wait before it can place its part as opposed to a randomized or directional method.

Acknowledgements. The authors would like to acknowledge Yash Mulgaonkar for help assembling the parts and Aditya Krishna for designing the node used in tetrahedral lattice structure.

References

1. Boothroyd, G., Knight, W.: Design for assembly. *IEEE Spectrum* 30(9), 53–55 (1993)
2. Bouabdallah, S.: Design and control of quadrotors with applications to autonomous flying. Ph.D. thesis, Ecole Polytechnique Federale de Lausanne, Lausanne, Switzerland (2007)
3. Galloway, K.C., Jois, R., Yim, M.: Factory floor: A robotically reconfigurable construction platform. In: Proc. of the IEEE International Conference on Robotics and Automation (ICRA), pp. 2467–2472 (2010)
4. Groover, M.P.: Automation, Production Systems, and Computer-Integrated Manufacturing, 3rd edn. Prentice Hall Press, Upper Saddle River (2007)
5. Heger, F., Singh, S.: Robust robotic assembly through contingencies, plan repair and re-planning. In: Proceedings of ICRA 2010 (2010)
6. Joo, H., Son, C., Kim, K., Kim, J.: A study on the advantages on high-rise building construction which the application of construction robots take. In: International Conference on Control, Automation and Systems (ICCAS), pp. 1933–1936 (2007)
7. Lindsey, Q., Kumar, V.: Distributed construction of truss structures - full proof of ‘dat’s’ algorithm. Tech. rep., University of Pennsylvania (2012)
8. Lindsey, Q., Mellinger, D., Kumar, V.: Construction of cubic structures with quadrotor teams. *Robotics: Science and Systems* (2011)
9. Matthey, L., Berman, S., Kumar, V.: Stochastic strategies for a swarm robotic assembly system. In: ICRA, pp. 1953–1958 (2009)
10. Napp, N., Klavins, E.: Robust by composition: Programs for multi-robot systems. In: 2010 IEEE International Conference on Robotics and Automation (ICRA), pp. 2459–2466 (2010)
11. Petersen, K., Nagpal, R., Werfel, J.: Termes: An autonomous robotic system for three-dimensional collective construction. In: Proc. of Robotics: Science and Systems Conference, RSS (2011)

12. Pounds, P., Dollar, A.: Hovering stability of helicopters with elastic constraints. In: ASME Dynamic Systems and Control Conference (2010)
13. Sanderson, A.C., de Mello, L.H., Zhang, H.: Assembly sequence planning. *AI Magazine* 11(1), 62–81 (1990)
14. Shen, S., Michael, N., Kumar, V.: Autonomous multi-floor indoor navigation with a computationally constrained MAV. In: Proc. of the IEEE International Conference on Robotics and Automation (ICRA), Shanghai, China (2011)
15. Werfel, J., INgber, D., Nagpal, R.: Collective construction of environmentally-adaptive structures. In: International Conference on Intelligent Robots and Systems (2007)
16. Werfel, J., Nagpal, R.: Three-dimensional construction with mobile robots and modular blocks. *Int. J. Rob. Res.* 27(3-4), 463–479 (2008)

Space-Time Group Motion Planning

Ioannis Karamouzas, Roland Geraerts, and A. Frank van der Stappen

Abstract. We present a novel approach for planning and directing heterogeneous groups of virtual agents based on techniques from linear programming. Our method efficiently identifies the most promising paths in both time and space and provides an optimal distribution of the groups' members over these paths such that their average traveling time is minimized. The computed space-time plan is combined with an agent-based steering method to handle collisions and generate the final motions of the agents. Our overall solution is applicable to a variety of virtual environment applications, such as computer games and crowd simulators. We highlight its potential on different scenarios and evaluate the results from our simulations using a number of quantitative quality metrics. In practice, our system runs at interactive rates and can solve complex planning problems involving one or multiple groups.

1 Introduction

In this work, we address the problem of planning the paths and directing the motions of agents in an environment containing static obstacles. The agents are organized into groups having similar characteristics and intentions, such as an army of soldiers in a real-time strategy game or virtual pedestrians that cross paths at an intersection. Each group has its own start and goal position (or area), and each group member will traverse its own path. Our formulation is designed for large groups and the main objective is to steer the group agents towards their destinations as quickly as possible and without collisions with obstacles or other agents in the environment. This task would have been simple, if the paths of the agents were independent of each other. However, due to space restrictions, congestions may appear and waiting or finding alternative paths may be more efficient for some agents. Furthermore, the time dimension introduces additional complexity into the planning process;

Ioannis Karamouzas
Department of CS&E, University of Minnesota
e-mail: ioannis@cs.umn.edu

Roland Geraerts · A. Frank van der Stappen
Utrecht University, The Netherlands
e-mail: {roland, frankst}@cs.uu.nl

congestions only appear at certain moments in time, while at other moments the same area may be completely empty allowing the agents to navigate through it without delay.

To resolve the aforementioned problems, we present a novel algorithm for planning and directing group motions that is based on *linear programming*. Our solution translates the group planning problem into a *network flow* problem on a graph that represents the free space of the environment. It plans in both space and time and uses the *column generation* technique [3] from the operations research theory to efficiently identify the most promising paths in the graph. The planner computes an optimal distribution of the agents over these paths such that their average traversal time is minimized. The computed space-time plan is then given as an input to an underlying agent-based method in order to handle collisions between the agents and generate their final motions.

As compared to prior solutions, our method offers the following characteristics:

- Space-time planning of multiple groups of heterogeneous agents based on a linear programming approach;
- Optimal distribution of the agents over alternative paths to resolve congestions;
- Coordination of the movements of the agents to prevent deadlocks that typically occur near bottlenecks (e.g. narrow passages) in the environment;
- Explicit waiting behavior that leads to convincing simulations of high-density crowds with no observed oscillatory behavior;
- Seamless integration of global path planning with existing schemes for local collision avoidance;
- Real-time performance in complex and challenging scenarios.

2 Related Work

The most common approach to simulate groups of autonomous agents is the flocking technique of Reynolds [19]. Although flocking leads to natural behavior, the group members act based only on local information and thus, they can get stuck in cluttered environments. To remedy this, Bayazit *et al.* [1] combined flocking techniques with probabilistic roadmaps, whereas Kamphuis and Overmars [8] used the concept of path planning inside corridors so that the group members can stay coherent.

In general, two-level planning approaches have been widely used in the graphics and animation community for multi-agent navigation and crowd simulation. In these approaches, a high-quality roadmap governs the global motion of each agent [5, 31], whereas a local planner allows the agent to avoid collisions with other agents and obstacles. Over the past few years, numerous models have been proposed for local collision avoidance including force-based approaches [7, 11] and variants of velocity-based methods [28, 27]. These approaches are known to exhibit emergent behaviors. Nevertheless, since they are separated from the global planner, they are susceptible to local-minima problems. In highly dynamic and densely packed environments, this normally leads to deadlock situations (see, e.g., Fig. 1(b)) that can only be resolved by rather unnatural motions. In such environments, even replanning is often insufficient to generate a feasible trajectory. Note that, similar to our

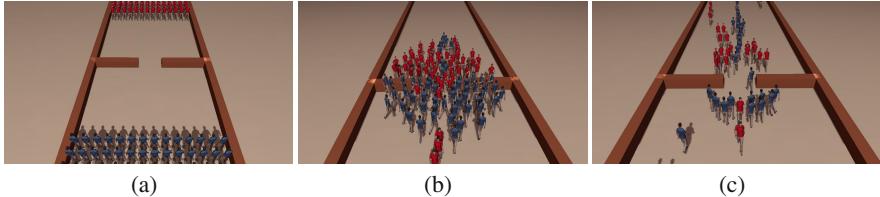


Fig. 1 (a) Two opposing groups, of 56 members each, need to pass through a narrow bottleneck. (b) Agent-based methods lead to congestion and deadlocks. (c) Our technique can successfully handle such challenging scenarios by planning in both space and time.

technique, the recent work of Singh *et al.* [23] resolves deadlocks by planning on a space-time graph. However, their formulation provides a localized space-time plan for each agent and cannot give any guarantees on the overall behavior of the agents.

Multi-agent path planning has also been extensively studied in robotics. Centralized planners, such as [15, 20], compute the motions of all agents simultaneously, whereas decoupled techniques [17, 22] plan a path for each agent independently and try to coordinate the resulting motions. However, both solutions are too computationally expensive for real-time interactive applications. Prioritized approaches have also been proposed that plan paths sequentially according to a certain priority scheme [29, 24]. This reduces the multi-agent planning problem to the problem of computing a collision-free trajectory for a single agent in a known dynamic environment, which can be addressed by roadmap-based space-time planners [30]. Even though prioritized techniques guarantee inter-agent collision avoidance, the growing complexity of the planning space limits the number of agents that they can simulate.

Prior work in the graphics community has also focused on the synthesis of realistic group motions mainly for offline simulations of crowds [14, 13]. Recently, a number of algorithms have been proposed to simulate the local behavior of pedestrian groups [18, 12], but they do not address the issue of path planning. An alternative approach has been proposed in [25] that guides large homogeneous groups using continuous density fields. This approach unifies global planning and local collision avoidance into a single framework and can become expensive when simulating a large number of groups. More recently, Patil *et al.* [16] proposed a novel technique to steer and interactively control groups of agents using smooth, goal-directed navigation fields. Although their approach can effectively handle challenging scenarios, it requires user input to guide the agent flows and successfully resolve congestion, as compared to our model that automatically accounts for time-consuming situations.

Finally, a different solution to the group path finding problem has emerged from the operations research community. Van den Akker *et al.* [26] formulated this problem as a *dynamic multi-commodity flow* problem. Their approach takes as input a graph resembling the free space of the environment and several groups of units, each having its own start and goal position. A *capacity* is associated with each arc on the graph representing the maximum number of units that can traverse this arc per unit time. The objective is to find paths that minimize the average arrival times of all units. Since this problem is known to be NP-hard, they proposed a new heuristic based on techniques from linear programming.

Our model is inspired by the work of van den Akker *et al.*, since we use a similar linear program to coordinate the global motions of the groups (Sect. 4). However, we extend their formulation to allow the simulation of *heterogeneous* groups. We also take into account capacity constraints on the *nodes* of the graph, which provides a more accurate solution to the problem and inhibits the oscillations we observed in [26] when multiple agents have to wait on a certain node. Moreover, van den Akker’s model uses a directed graph to capture the free workspace. This significantly restricts the movements of the agents; in real life, for example, pedestrians can walk in either direction along a sidewalk or simultaneously enter/exit a building through the same door. To resolve this, our method uses *undirected* edges and thus, additional constraints are considered. Finally, van den Akker *et al.* only present a theoretical framework for global planning and do not discuss simulation. We address this with a simple, yet effective, *steering* algorithm that guides the agents towards their goals without collisions (Sect. 5).

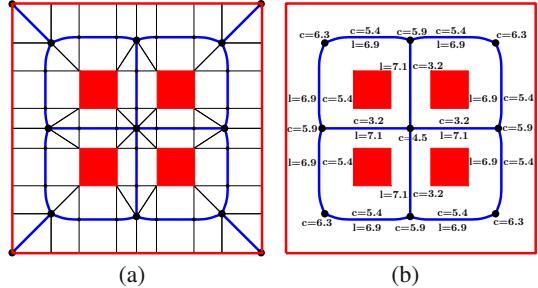
3 Problem Formulation and Overall Solution

In our problem setting, we are given a geometric description of a virtual environment in which k groups of agents must move. Each group $C_i, i \in [1, k]$ has its own start s_i and goal position t_i and consists of d_i agents that need to navigate from a specified start to a specified goal area defined around the group’s origin and destination point respectively. Furthermore, each group C_i is assigned a desired speed U_i^{des} indicating the speed at which its members prefer to move. Given a group C_i , we denote an agent belonging to C_i as A_{ij} , where $j \in [1, d_i]$. For simplicity, we assume that each agent A_{ij} is modeled as a disc having radius r_{ij} and is subject to a maximum speed v_{ij}^{\max} . We further assume that A_{ij} keeps a certain psychophysical [7] distance $\rho_{ij} \geq r_{ij}$ from other agents and static obstacles in order to feel comfortable. This distance defines the *personal space* of A_{ij} modeled as a disc centered at the agent. The task is then to steer the group members A_{ij} toward their corresponding goal areas as quickly as possible and without collisions with the environment and with each other.

We propose a two-level framework to solve the aforementioned planning problem. In the first phase, we formulate the group planning problem as a *dynamic multi-commodity flow problem* and employ an Integer Linear Programming (ILP) approach to solve it. The ILP plans in both time and space by taking into account capacity constraints on the edges and the nodes of a graph that is based on the *medial axis* of the environment. The variables in the ILP represent the number of agents using a certain path. Such a path is described by the graph nodes that the agent should visit along with the times at which these nodes should be reached. Furthermore, waiting behavior can be encoded in the path.

In the second phase of our framework, the paths computed by the ILP are given as input to an agent-based steering algorithm in order to generate the final motions of the agents. The algorithm creates a number of *lanes* across the medial axis edges and computes for each agent a trajectory along these lanes that respects the space-time

Fig. 2 (a) A medial axis graph MG enhanced with proximity information to nearest obstacles. (b) Its corresponding capacitated graph G .



plan provided by the ILP. At every step of the simulation, a local collision avoidance method is also used to guarantee collision-free navigation for the agents.

In the following sections, we elaborate on our method. For a more detailed explanation, including theorems, proofs and additional experiments, we refer to [9].

4 Path Planning for Groups

This section outlines the first phase of our framework that formulates the multi-group path planning problem as a dynamic multi-commodity flow problem.

4.1 Creating a Capacitated Graph

To solve the problem, we first need a graph that resembles the free space of the environment. We base this graph on the edges and vertices of the medial axis (MA), as it provides maximum clearance from the obstacles and includes all cycles present in the environment. In particular, we precompute a MA graph $MG = (V, E)$ by sampling *event points* (nodes of degree 2 or more) along the MA based on the approach proposed in [4]. The resulting structure fully covers the free space of the environment using a minimum amount of sample points. Also, for each sample point, its minimum clearance is stored along with its corresponding nearest obstacle points. Figure 2(a) shows an example of such a graph. Note that the graph is undirected.

Based on MG , we compute a *capacitated graph* $G = (N, E)$ as follows. We first copy all vertices and edges from MG to G . We then determine all nodes of degree 1 in the graph. Such nodes denote dead-ends in the environment and are removed from the set N in G . Their corresponding edges are also removed from the edge set E of G . For each remaining edge $e \in E$, we define its traversal time l_e as the time that an agent requires to traverse this edge and compute it as the edge length divided by the maximum speed U^{\max} among all groups' desired speeds. A capacity c_e is also assigned to each edge denoting the maximum number of agents that can traverse the edge at the same time while walking next to each other. The edge capacity is computed as the minimum clearance along the edge divided by the maximum personal space ρ^{\max} among all agents. Similarly, a capacity c_n is associated with each node n in the graph indicating the maximum number of agents that can simultaneously be

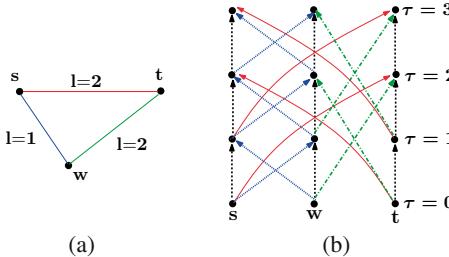


Fig. 3 (a) A simple capacitated graph with integral traversal times. (b) Its corresponding time-expanded graph with time horizon $T = 4$ and time step $\Delta t = 1s$. Note that for clarity, we omit the capacity information from both graphs.

on the node. We refer the reader to Fig. 2(b) for the capacitated graph corresponding to the MA graph depicted in Fig. 2(a).¹

The generated graph captures different paths that the agents can follow in order to reach their destinations. Nevertheless, to properly avoid congestions, we must not only know which nodes are visited but also at which times these nodes should be reached. In addition, waiting at certain nodes may be more efficient for some of the agents. For that reason a condensed *time-expanded graph* is defined as proposed in [2] that adds the time dimension to the graph G . The basic idea here is to round up the traversal time l_e of each edge e in G to the nearest multiple of an appropriately chosen time step Δt , i.e. $l_e^* = \lceil l_e / \Delta t \rceil$. The capacity of the edge is also multiplied by Δt in order to define the maximum number of agents that can traverse the edge in one time unit, i.e. $c_e^* = \lfloor c_e \cdot \Delta t \rfloor$. In a similar manner, the discrete capacity of each node n is expressed as $c_n^* = \lfloor c_n \cdot \Delta t \rfloor$.

Let $G^T = (N^T, A^T)$ denote the time-expanded graph of G , where T defines the time horizon, that is the total number of discrete time steps. G^T is directed and is constructed from the static graph G based on its discrete capacities and traversal times. The set N^T contains a copy of each node n of the static graph G for each time step $\tau = 0 \dots T - 1$. Such a copy is denoted as $n(\tau)$. In addition, for every edge $e = \{n, m\}$ in the graph G , two arcs will be created in G^T , one from $n(\tau)$ to $m(\tau + l_e^*)$ and one from $m(\tau)$ to $n(\tau + l_e^*)$. Finally, waiting arcs are also added to A^T from each node $n(\tau)$ to the same node one time-step later $n(\tau + 1)$, for all $\tau < T$. Such an arc allows an agent to stay at node n for one time-step period, i.e. $l^* = 1$. Its capacity is defined as the number of agents that can simultaneously wait at the node and is the same as the discrete capacity c_n^* of the static node n .

An illustration of a time-expanded graph is given in Fig. 3. Such a graph is not explicitly constructed. Instead, time-expanded nodes and arcs are created on-the-fly as further discussed in Sect. 4.3. In addition, an upper bound on the time horizon T is set as explained in [9].

¹ If the start s_i or goal t_i position of a group C_i is not included in the graph G , it is added to the set N as an extra node. The new node is also connected to G introducing additional edges in the set E .

4.2 ILP Formulation

In our problem formulation, we are given the origin s_i , destination t_i and the size d_i of each group C_i . For now, let us assume that we know all valid paths in the time-expanded graph G^T corresponding to each origin-destination pair. A path is valid, if it starts at a node $s_i(0) \in N^T$ for some $i \in [1, k]$ and ends at node $t_i(\tau)$ for the same i . Let \mathcal{P} denote the set of all these valid paths in G^T . For every path $p \in \mathcal{P}$ we introduce a decision variable f_p which denotes the number of agents using the path p . Then, we can model our path planning problem as an Integer Linear Programming (ILP) problem as follows.

Our goal is to minimize the average arrival time of all agents, or, equivalently, the sum of the travel times of all agents. We use l_p to denote the cost (length) of path p , which equals to the arrival time of p at its destination. This leads to the objective function shown in (1). We also formulate demand constraints to guarantee that all agents will reach their targets as expressed in (2), where \mathcal{P}_i is the set of paths in G^T starting at s_i and ending at t_i given a group C_i .

Restrictions are also needed to ensure that the arc capacity constraints are obeyed. In our problem setting, these constraints are much harder to model than in a normal multi-commodity flow problem, since the static capacitated graph G is undirected. Consequently, an edge can be traversed in both directions at the same point in time. To avoid having to add an enormous number of constraints, we introduce a non-negative decision variable u_e for every edge e in the graph G . We call one direction on the edge forward and the other one backward. Let $F(e) \subset A^T$ be the forward arcs in the time-expanded graph G^T emerging from e , and $B(e) \subset A^T$ the corresponding backward arcs. Then, the capacity c_e^* of the edge is divided between the two directions. We do not fix the capacity distribution beforehand, but we make it time-independent by putting the capacity of the forward arcs equal to u_e and the capacity of the backward arcs equal to $c_e^* - u_e$. This adds (3) and (4) to the ILP, where \mathcal{P}_a denotes the subset of paths using arc $a \in A^T$.

Finally, to account for the throughput limitation of each node v of the time-expanded graph, (5) is added to the ILP constraints, where \mathcal{P}_v denotes the subset of paths in the time-expanded graph using the expanded node v . The above considerations result in the following ILP problem:

$$\min \sum_{p \in \mathcal{P}} l_p f_p \quad (1)$$

$$s.t. \sum_{p \in \mathcal{P}_i} f_p \geq d_i \quad \forall i = 1 \dots k \quad (2)$$

$$\sum_{p \in \mathcal{P}_a} f_p - u_e \leq 0 \quad \forall e \in E, a \in F(e) \quad (3)$$

$$\sum_{p \in \mathcal{P}_a} f_p + u_e \leq c_e^* \quad \forall e \in E, a \in B(e) \quad (4)$$

$$\sum_{p \in \mathcal{P}_v} f_p \leq c_v^* \quad \forall v \in V^T \quad (5)$$

$$f_p \geq 0 \text{ and integral} \quad \forall p \in \mathcal{P} \quad (6)$$

$$u_e \geq 0 \text{ and integral} \quad \forall e \in E \quad (7)$$

Obviously, we do not know the entire set of paths \mathcal{P} and enumerating it would be impractical, since there are infinite paths in the time-expanded graph. Therefore, we will make a selection of paths that we consider ‘possibly useful’, that is, paths that might improve the value of our objective function, and we will solve the ILP for this small subset. We determine these paths by considering the LP-relaxation of the ILP, which is obtained by removing the integrality constraints from the decision variables f_p and u_e . We solve the LP-relaxation through the technique of *column generation*, which was first described by Ford and Fulkerson [3].

4.3 Column Generation and Path Finding

The basic idea of column generation is to solve the LP problem for a restricted set of variables (the set of valid paths in G^T in our case) and then add variables that may improve the objective value until no additional variables can be found anymore. In case of a minimization problem, it is well known from the theory of column generation that the addition of a variable will only improve the solution if its *reduced cost* is negative. The reduced cost of a path $p \in \mathcal{P}_i$ for a group C_i is equal to

$$l_p + \sum_{a \in \alpha(p)} \omega_a + \sum_{v \in \lambda(p)} \phi_v - \psi_i, \quad (8)$$

where $\alpha(p), \lambda(p)$ denote the arcs and nodes respectively in G^T used by path p , and $\psi_i, \omega_a, \phi_v \geq 0$ are the dual variables for the demand, arc and node constraints respectively that derive from the *dual* of the current LP^{2,3}. Consequently, for each group C_i we need to find a path $p \in \mathcal{P}_i$ such that

$$l_p + \sum_{a \in \alpha(p)} \omega_a + \sum_{v \in \lambda(p)} \phi_v - \psi_i < 0. \quad (9)$$

² Every linear programming problem, referred to as a primal problem, can be converted into a dual problem. For a primal problem expressed in its symmetric form $\{\min \mathbf{c}^T \mathbf{x} : \mathbf{A}\mathbf{x} \geq \mathbf{b}, \mathbf{x} \geq \mathbf{0}\}$, the corresponding symmetric dual problem is given by $\{\max \mathbf{b}^T \mathbf{y} : \mathbf{A}^T \mathbf{y} \leq \mathbf{c}, \mathbf{y} \geq \mathbf{0}\}$, where $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{c}, \mathbf{x} \in \mathbb{R}^n$, $\mathbf{b} \in \mathbb{R}^m$ and $\mathbf{y} \in \mathbb{R}^m$.

³ Given a linear program in its symmetric form, the reduced cost can be computed as $\mathbf{c} - \mathbf{A}^T \mathbf{y}$, where \mathbf{y} denotes the variables of the dual of the linear program. In our case, our LP formulation results in a dual variable ψ_i for the demand constraint (2) corresponding to the group C_i , a dual variable ω_a for the constraints (3)-(4) corresponding to the arc a and a dual variable ϕ_v for the constraint (5) corresponding to the node v .

This inequality can actually be rewritten to gain more insight into its meaning. The cost l_p of a path p is equal to the sum of the length of the arcs contained in p , that is $l_p = \sum_{a \in \alpha(p)} l_a^*$, where l_a^* denotes the discrete length (traversal time) of arc $a \in A^T$ (see Sect. 4.1). Note, though, that the traversal time l_a^* is computed based on the maximum desired speed U^{\max} among all groups. However, since each group C_i has its own desired speed U_i^{des} , we define the correct length of the arc a for $p_i \in \mathcal{P}$ as $l'_a = \lceil l_a^* \cdot U^{\max} / U_i^{\text{des}} \rceil$. Regarding the $\sum_{v \in \lambda(p)} \phi_v$ term of (9), it is clear that if a node $v \in N^T$ is contained in the path p , the path also contains precisely one arc that terminates at this node (with the exception of the origin node). Consequently, the effective contribution of the path's nodes to the reduced cost of p becomes $\sum_{a \in \alpha(p)} \phi_m$, given that $a = (n, m)$. Finally, reorganizing (9) we obtain:

$$\sum_{a \in \alpha(p)} (l'_a + \omega_a + \phi_m) < \psi_i. \quad (10)$$

Deciding whether (10) holds defines the *pricing problem* and can be efficiently solved for each group C_i as follows. We compute the shortest path for C_i in the time-expanded graph G^T , where each arc $a \in A^T$ has a modified length $l'_a + \omega_a + \phi_m$. We use an A* search to efficiently find such a path from the origin $s_i(0)$ of the group to some copy of its destination $t_i(\tau)$. In A*, explored nodes of the time-expanded graph are created and added to an open set based on a function $f(v) = g(v) + h(v)$ that determines how promising each node v is. The cost $g(v)$ of reaching the current node v from the origin $s_i(0)$ is based on the modified arc lengths. Regarding the heuristic $h(v)$, we use the shortest path length (expressed in discrete time units) from v to t_i in the static graph G . This heuristic is clearly admissible, since the path length in a time-expanded graph may be enlarged due to the modified lengths of its arcs and the possible presence of waiting arcs. As our proposed heuristic is also consistent, our search algorithm retains the optimality of an A* search and can efficiently compute the shortest path for the group C_i . If the length of this path is shorter than ψ_i , we can add it to the LP and iterate. Otherwise the optimum has been found, since adding the path will not decrease the objective value of the LP.

Algorithm 1 summarizes our column generation algorithm. Note that, initially, the LP has no columns and hence, we need to introduce an initial set of paths, one for each group C_i . In general, we can start with any set, as long as it constitutes a feasible solution [9].

Algorithm 1. Column Generation

- 1: Find initial paths and add them as columns to the LP
 - 2: **repeat**
 - 3: Solve LP-relaxation
 - 4: **for** each group C_i **do**
 - 5: Find a shortest $(s_i(0) - t_i(\tau))$ -path in G^T w.r.t. the modified arc lengths
 - 6: **if** length of the path is less than ψ_i **then**
 - 7: add the path as a new column to the LP
 - 8: **end if**
 - 9: **end for**
 - 10: **until** no path has been added
-

4.4 Obtaining an Integral Solution

At the end of the column generation algorithm, we have an optimal solution for the LP. Most likely, though, some of the variables f_p in our solution will have a fractional value and hence, we have to ensure that we end up with an integral solution, since we cannot send fractions of agents along a path. We alleviate the problem as follows. Since we have generated useful paths when we solved the LP, we include all these paths in the ILP. We then round down their decision variables f_p , which leads to an integral solution that satisfies the capacity constraints. Because of the rounding down, though, some of the agents will be left without paths. For these agents, we construct additional paths using the Cooperative A* algorithm [21]. These paths are added to the ILP, which then can be solved to optimality. Note that an alternative approach would be to artificially increase the sizes d_i of the groups while solving the LP-relaxation problem. Therefore, after rounding down the f_p variables all agents will be assigned a path.

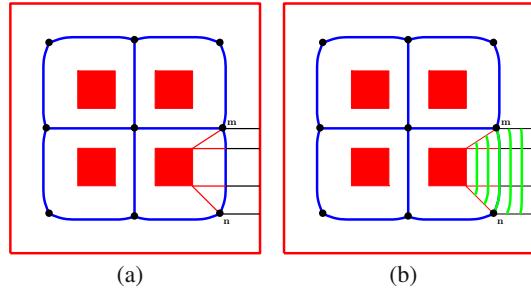
5 Agent Motion Planning

In this section, we elaborate on the second phase of our framework. The goal here is to generate the final motion of each agent A_{ij} based on its path p_{ij} computed by the ILP. Such a path is defined in the time-expanded graph G^T and can be described by the arcs that it uses. A time-expanded arc is either a waiting arc or a traversal arc that corresponds to an edge in the MA graph MG . Therefore, a straightforward approach for generating the final trajectory of the agent A_{ij} would be to let A_{ij} follow the MA edges, while adjusting its speed in order to adhere to the time-constraints of its ILP path. Note, though, that multiple agents may have the same path in G^T or share the same edge or node at the same time. Since all paths produced by the ILP satisfy the capacity constraints, this means that there is enough clearance for these agents to walk or stand next to each other. Based on this observation, we can utilize the maximum capacity of each edge and discretize the Voronoi regions of the environment into a number of *lanes*. The agents will use these lanes to spread over the environment and reach their goals, respecting at the same time the space-time constraints provided by the ILP paths.

5.1 Constructing Lanes

We construct the lanes by exploiting the properties of the MA graph $MG = (V, E)$. Let $e = \{n, m\}$ denote an edge in the set E , where both n and m are not dead-end vertices. Let us also define a pseudo-orientation for the edge, e.g. from n to m . As described in Sect. 4.1, an edge consists of a number b of sample points B_i , $1 \leq i \leq b$. For each sample point its minimum clearance is also stored along with its closest points to the obstacles' boundaries. Given the orientation of the edge, let l_i and r_i denote the left and right closest obstacle points assigned to each sample B_i .

Fig. 4 (a) An edge can be expressed as a sequence of portals. (b) Lanes are created along the edge by placing waypoints on the portals.



Then the edge e can be expressed as a sequence of *portals* where each portal is described by the tuple (B_i, r_i, l_i) (see Fig. 4(a)).

Lanes can be easily constructed along the edge by placing waypoints on the navigation portals. Let c_e^* be the integral capacity of the edge in the capacitated graph G . This capacity defines the maximum number of agents that can simultaneously traverse the edge and hence, c_e^* lanes will be created. For each lane $L_k, k \in [1, c_e^*]$, a waypoint is generated on each of the b portals of the edge based on a parameter $\omega_k = k/(c_e^* + 1), \omega_k \in (0, 1)$. In case the waypoint is located at the right side of the portal ($\omega_k \leq 0.5$), its exact position w_i is given by: $w_i = r_i + 2\omega_k(B_i - r_i)$. If the waypoint is at the left side of the portal, its position between l_i and B_i linearly depends on $2 - 2\omega_k$. We refer the reader to Fig. 4(b) for the lanes corresponding to the edge e , given that $c_e^* = 5$. Since each edge can also be traversed in the opposite direction, for each lane its reverse one L'_k is also added using the same procedure as above.

5.2 Trajectory Synthesis for an Agent

Given the lanes as constructed above and the ILP path p_{ij} corresponding to the agent A_{ij} , we now describe how we can determine a trajectory for A_{ij} . Let p_{ij} consists of q arcs. We first express this path as a sequence of tuples $(a_1, T_1), \dots, (a_q, T_q)$, where for the θ^{th} tuple, $a_\theta = (n(\tau), m(\tau + l'_{a_\theta}^*))$ denotes an arc in the time-expanded graph and T_θ is the time instant at which the task of the arc (waiting or traversing a lane) should be completed, that is, $T_\theta = (\tau + l'_{a_\theta}^*)\Delta t$.

Having determined the times T corresponding to each arc a , we can steer the agent towards its goal as follows. Let \mathbf{x}_{ij} denote the current position of the agent, \mathbf{v}_{ij} its current velocity, and $\mathbf{v}_{ij}^{\text{pref}}$ its preferred velocity. Let also (a_θ, T_θ) denote the next tuple to be processed. If a_θ is a waiting arc, we set the preferred velocity to zero for the next $T_\theta - t$ seconds, where t is the current time of the simulation; note that if $T_\theta \leq t$, we simply ignore the waiting arc and query the next tuple in the list. In case a_θ is a traversal arc, we determine its corresponding edge e in MG along with the edge's lanes \mathcal{L} that have the same orientation as the arc a_θ . The agent needs to select one of these lanes and traverse it within T_θ .

Choosing a Lane: We first determine all lanes in \mathcal{L} that are *free*. A lane L_k is considered as free, if no other agent has entered it at the current time t and if its reverse lane L'_k is unoccupied. Among the free lanes, we select the one that is closest to the agent. Given a lane L_k , the distance $D(L_k, \mathbf{x}_{ij})$ between the agent's current position \mathbf{x}_{ij} and L_k can be computed by projecting \mathbf{x}_{ij} into the segments defined by the waypoints of the lane. In case no free lanes are available⁴, with each lane L_k , we dynamically assign a cost that depends on the distance $D(L_k, \mathbf{x}_{ij})$ and the biomechanical energy $E(L_k)$ that the agent needs to spend in order to traverse L_k . This energy is approximated as in the work of Guy *et al.* [6] and is based on a series of experimental studies regarding the relationship between the walking speed and the energy expenditure of real humans. Given $E(L_k)$, the total cost of the lane L_k can now be defined as: $cost(L_k) = c_D D(L_k, \mathbf{x}_{ij}) + c_E E(L_k)$, where c_D, c_E are weights specifying the relative importance of each cost term. Based on the above function, the agent A_{ij} retains the lane $L_s \in \mathcal{L}$ with the lowest cost.

Moving Along a Lane: The agent A_{ij} uses the selected lane L_s as an *indicative route* in order to traverse its current arc. More precisely, an attraction point moves along L_s and attracts the agent forward as defined in [10]. Given the attraction point and the distance that A_{ij} still has to traverse within the time $T_\theta - t$ that has at its disposal, the preferred velocity $\mathbf{v}_{ij}^{\text{pref}}$ of A_{ij} can then be estimated. For additional details on the notion of indicative routes, we refer the reader to [9].

Local Collision Avoidance: At each simulation time-step, the preferred velocity $\mathbf{v}_{ij}^{\text{pref}}$ of the agent A_{ij} is given as an input to a local collision avoidance method in order to compute a collision-free velocity for the agent and update its position. Note that simply using $\mathbf{v}_{ij}^{\text{pref}}$ may not be sufficient for a collision-free motion; the agent may still need to resolve collisions with its neighboring agents and the static part of the environment while advancing along its selected lane or waiting at a specific location. In general, any local method that is based on collision prediction can be used. In our implementation, we consider the model in [11] that tackles the collision avoidance problem using social forces.

6 Experimental Results

We demonstrate the applicability of our approach in four challenging scenarios as can be seen in Figs. 1 and 5. The resulting simulations can be found at <http://sites.google.com/site/ikaramouzas/ilp>.

Quality Evaluation: Our approach identifies areas with low capacity and divides the agents over different space-time paths allowing them to avoid congestions and

⁴ An agent, due to interactions with other agents, may deviate from the time plan provided by the ILP and arrive at its next arc later than the expected time T_θ . Thus, in practice, there may be no free lanes to follow. However, a more relaxed time-plan can be provided for each agent by including an additional time t_{add} to T_θ . This will give the agent more time to complete its task and resolve challenging interactions.

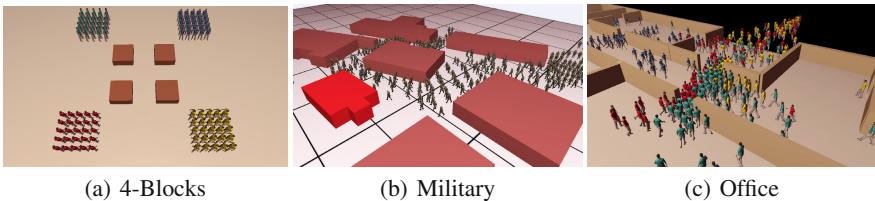


Fig. 5 Example scenarios: (a) Four groups, of 25 agents each, in opposite corners of the environment exchange positions. (b) An army of 400 soldiers needs to move towards a designated goal area in a village-like environment. (c) 700 agents organized into 7 groups have to evacuate an office building through two narrow exits.

quickly reach their destinations. In the 4-blocks scenario, for example, the agents anticipate that a congestion will occur at the center of the environment and avoid it by moving around the obstacles. Our method also prevents deadlocks from occurring by regulating the starting times of the agents, incorporating explicit waiting behavior and successfully dealing with opposing flows of agents through the use of lanes, as can be seen in the narrow bottleneck scenario (Fig. 1). Furthermore, by controlling the starting and waiting times of the agents, no oscillatory behavior is observed when dense crowds of agents thread through narrow spaces, like the doorways in the office scenario.

Our method is also able to generate well-known crowd phenomena which have been noted in the pedestrian literature (see [7] for an overview), such as lane formations (e.g. 4-blocks and military scenarios), queuing and following behaviors (narrow bottleneck and office), as well as slowing down and waiting behaviors to resolve challenging interactions (narrow bottleneck, military and office). Furthermore, in the narrow bottleneck scenario, arch-like blockings are formed at either side of the passage when the two opposite flows meet (Fig. 1(c)). This phenomenon is in accordance with real-life behavior [7].

Besides the visual inspection of the simulations, we are also interested in a quantitative evaluation of our model. As shown in Table 1, by planning in both time and space and exploiting the notion of lanes, interactions among agents are efficiently solved resulting in smooth trajectories and faster traveling times as compared to existing agent-based systems. In the 4-blocks scenario, for example, using the RVO method in combination with a global roadmap [31] led to an average traveling time of 37.06 s, whereas the latest arrival time was 52.59 s. Similarly, in the narrow bottleneck example, it took 219.4 s for the last agent to arrive at its goal and the average traverse time was 139.46 s. Such high traveling times are expected, since the agents' motions are not taken into account during the global planning and hence, congestions and deadlock situations arise. In contrast, using our approach, such situations are predicted and avoided. In the 4-blocks, even when we used a dynamic roadmap to account for congestion as in [6] along with the ORCA method [27], the maximum travel time was 45.79 s, as the agents were still prone to local minima problems.

Table 1 Results for the four scenarios. The optimality error is the average percentage error between the actual and the estimated arrival times of the agents.

| Scene | Max. travel time (s) | Avg. travel time (s) | Optimality error (%) |
|------------|----------------------|----------------------|----------------------|
| 4-Blocks | 32.41 | 30.01 | 1.29 |
| Bottleneck | 186.90 | 129.75 | 6.36 |
| Military | 149.78 | 117.61 | 4.69 |
| Office | 189.10 | 123.59 | 8.63 |

To adequately assess the quality of our approach, we also need to determine the efficiency of the steering algorithm used in the second phase of our framework. For that reason, we compared the *actual* time that an agent requires in order to reach its destination with its *expected* arrival time. The latter is simply the length of the agent’s time-expanded path computed by the ILP. The *percentage error* between the actual and the expected arrival time can then be used as a measure of the optimality of the agent’s trajectory. Clearly, since the agent needs to avoid collisions with other agents and is also subject to dynamic constraints, its actual traveling time is higher than the expected one. However, as can be seen in the last column of Table 1, in all of our experiments, the optimality error is less than 10% indicating that our steering algorithm generates near time-optimal trajectories.

Performance: Table 2 summarizes the performance of our approach on the four benchmark scenarios. All computations were run on a 2.4 GHz Core 2 Duo CPU (on a single thread). The eighth column of the table indicates the total running time of the first phase of our approach. This time is split into the time used by the column generation part of our algorithm, that is the time for solving the LP-relaxation problem (t_{LP}) and for finding paths using A* in the time-expanded graph (t_{path}), the time needed for finding additional paths due to the missing capacities (t_{add}) and the time to solve the final ILP problem (t_{ILP}). The last column of the table indicates the average computation time of the second phase of our approach, that is the average time taken per simulation step to steer the agents and resolve collisions. As can be inferred, our steering algorithm combined with the underlying collision avoidance scheme is very efficient. This is expected, since most of the collisions are taken into account during the ILP planning. Thus, only a limited number of interactions need to be resolved at every step of the simulation.

Regarding the total planning time of the first phase, it is clear that the LP part of the column generation dominates the overall runtime performance. As can be observed, the *difficulty* of the problem seems to have a high impact on the computational cost of the LP. See, for example, the difference in the running times between the 4-blocks and the narrow bottleneck scenarios. The difficulty of the problem is directly related to the arc and node capacities of the time-expanded graph and can be controlled by the size of the time step Δt used for the discretization of the graph. Shorter time steps improve the accuracy of the graph at the expense of higher computational times. In contrast, large time steps result in faster running times; the capacities of the arcs and the nodes of the graph are scaled as well, reducing the

Table 2 The performance of our approach on the four example scenarios. Reported times are in msec. The average simulation time is expressed in msec/frame. The third column indicates the number of nodes and edges of the underlying capacitated graph.

| Scene | #Agents | Graph | t_{LP} | t_{path} | t_{add} | t_{ILP} | Total planning time | Avg. sim. time |
|------------|---------|------------|----------|------------|-----------|-----------|---------------------|----------------|
| 4-Blocks | 100 | (9, 12) | 0 | 15.6 | 0 | 0 | 15.6 | 0.28 |
| Bottleneck | 112 | (4, 3) | 109.2 | 46.8 | 78 | 124.8 | 358.8 | 0.29 |
| Military | 400 | (42, 56) | 343.3 | 140.4 | 0 | 31.2 | 514.8 | 0.37 |
| Office | 700 | (218, 235) | 561.6 | 205.2 | 62.4 | 327 | 1156.2 | 0.72 |

number of potential bottlenecks in the environment and leading to simpler LP problems. In general, the running time is inversely proportional to the size of the time step. This is confirmed by a number of experiments we conducted in different scenarios and for different values of Δt . In our example scenarios, the time step Δt was set to 1 s with the exception of the office scenario. Here, to obtain running times that are sufficiently low, we sacrificed the optimality of the LP solution and set $\Delta t = 2$ s.

7 Conclusion and Future Work

We presented a novel approach for planning and directing groups of virtual characters by combining techniques from Integer Linear Programming with an agent-based steering framework. We have demonstrated the applicability of our method through a wide range of challenging scenarios. Despite the computational complexity raised by the ILP formulation, our system runs at interactive rates by using the column generation algorithm and choosing an appropriate discretization of time.

Our approach makes some simplifying assumptions. Most notably, we assume that the characters choose paths so as to minimize their average traveling time. However, in real-life, other parameters can also affect the path choices that people make, such as the safety of the area that the path crosses, its attractiveness, etc. We should also point out that our system is specifically designed to model large groups. While in-group members can have different behavior characteristics, during the ILP planning, we assume that they prefer to move with the same desired speed. This enables our method to account for dynamic congestion and other time-consuming situations.

Our current work focuses on crowds of virtual characters, but it can be easily generalized to address multi-robot planning and coordination problems. Other virtual environment applications can benefit from it as well. An interesting extension, for example, would be to use our formulation for traffic simulation. In the future, we would also like to address dynamic and interactive environments, since currently we assume that the virtual world remains static and its dynamics are perfectly known *a priori*. Note, though, that in dynamically changing environments, the agents may not have the time to plan and replan optimal paths across the time-expanded graphs. To this end, we can trade-off optimality for performance by quitting the column generation algorithm before we have solved the LP-relaxation problem to optimality.

References

1. Bayazit, O.B., Lien, J.M., Amato, N.M.: Better group behaviors in complex environments using global roadmaps. In: Artificial Life, pp. 362–370 (2003)
2. Fleischer, L., Skutella, M.: Quickest flows over time. SIAM J. on Computing 36(6), 1600–1630 (2007)
3. Ford Jr., L.R., Fulkerson, D.R.: A suggested computation for maximal multi-commodity network flows. Management Science, 97–101 (1958)
4. Geraerts, R.: Planning short paths with clearance using explicit corridors. In: IEEE Int. Conf. on Robotics and Automation, pp. 1997–2004 (2010)
5. Geraerts, R., Overmars, M.H.: The corridor map method: Real-time high-quality path planning. In: IEEE Int. Conf. on Robotics and Automation, pp. 1023–1028 (2007)
6. Guy, S.J., Chhugani, J., Curtis, S., Pradeep, D., Lin, M., Manocha, D.: PLEdestrians: A least-effort approach to crowd simulation. In: Symp. on Computer Animation, pp. 119–128 (2010)
7. Helbing, D., Buzna, L., Werner, T.: Self-organized pedestrian crowd dynamics and design solutions. Traffic Forum 12 (2003)
8. Kamphuis, A., Overmars, M.H.: Finding paths for coherent groups using clearance. In: Symp. on Computer Animation, pp. 19–28 (2004)
9. Karamouzas, I.: Motion planning for human crowds: from individuals to groups of virtual characters. Ph.D. thesis, Utrecht University, the Netherlands (2012)
10. Karamouzas, I., Geraerts, R., Overmars, M.: Indicative routes for path planning and crowd simulation. In: Foundations of Digital Games, pp. 113–120 (2009)
11. Karamouzas, I., Heil, P., van Beek, P., Overmars, M.H.: A Predictive Collision Avoidance Model for Pedestrian Simulation. In: Eggels, A., Geraerts, R., Overmars, M. (eds.) MIG 2009. LNCS, vol. 5884, pp. 41–52. Springer, Heidelberg (2009)
12. Karamouzas, I., Overmars, M.: Simulating and evaluating the local behavior of small pedestrian groups. IEEE Trans. Vis. Comput. Graph. 18(3), 394–406 (2012)
13. Kwon, T., Lee, K.H., Lee, J., Takahashi, S.: Group motion editing. ACM Trans. on Graphics 27(3), 1–8 (2008)
14. Lee, K.H., Choi, M.G., Hong, Q., Lee, J.: Group behavior from video: a data-driven approach to crowd simulation. In: Symp. on Computer Animation, pp. 109–118 (2007)
15. Li, T.Y., Chou, H.C.: Motion planning for a crowd of robots. In: IEEE Int. Conf. on Robotics and Automation, pp. 4215–4221 (2003)
16. Patil, S., van den Berg, J., Curtis, S., Lin, M.C., Manocha, D.: Directing crowd simulations using navigation fields. IEEE Trans. Vis. Comput. Graph. 17, 244–254 (2011)
17. Peng, J., Akella, S.: Coordinating multiple robots with kinodynamic constraints along specified paths. Int. J. of Robotics Research 24, 295–310 (2005)
18. Peters, C., Ennis, C.: Modeling groups of plausible virtual pedestrians. IEEE Computer Graphics and Applications 29(4), 54–63 (2009)
19. Reynolds, C.W.: Flocks, herds, and schools: A distributed behavioral model. Computer Graphics 21(4), 24–34 (1987)
20. Sánchez, G., Latombe, J.C.: Using a PRM planner to compare centralized and decoupled planning for multi-robot systems. In: IEEE Int. Conf. on Robotics and Automation, pp. 2112–2119 (2002)
21. Silver, D.: Cooperative pathfinding. In: Artificial Intelligence and Interactive Digital Entertainment, pp. 117–122 (2005)
22. Simeon, T., Leroy, S., Laumond, J.P.: Path coordination for multiple mobile robots: A resolution-complete algorithm. IEEE Trans. on Robotics and Automation 18(1), 42–49 (2002)

23. Singh, S., Kapadia, M., Hewlett, B., Reinman, G., Faloutsos, P.: A modular framework for adaptive agent-based steering. In: ACM Symp. on I3D, pp. 141–150 (2011)
24. Sung, M., Kovar, L., Gleicher, M.: Fast and accurate goal-directed motion synthesis for crowds. In: Symp. on Computer Animation, pp. 291–300 (2005)
25. Treuille, A., Cooper, S., Popović, Z.: Continuum crowds. ACM Trans. on Graphics 25(3), 1160–1168 (2006)
26. van den Akker, M., Geraerts, R., Hoogeveen, H., Prins, C.: Path Planning for Groups Using Column Generation. In: Boulic, R., Chrysanthou, Y., Komura, T. (eds.) MIG 2010. LNCS, vol. 6459, pp. 94–105. Springer, Heidelberg (2010)
27. van den Berg, J., Guy, S.J., Lin, M.C., Manocha, D.: Reciprocal n-body collision avoidance. In: Int. Symp. on Robotics Research, pp. 3–19 (2009)
28. van den Berg, J., Lin, M., Manocha, D.: Reciprocal velocity obstacles for real-time multi-agent navigation. In: IEEE Int. Conf. on Robotics and Automation, pp. 1928–1935 (2008)
29. van den Berg, J., Overmars, M.H.: Prioritized motion planning for multiple robots. In: IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, pp. 2217–2222 (2005)
30. van den Berg, J., Overmars, M.H.: Roadmap-based motion planning in dynamic environments. IEEE Trans. on Robotics and Automation 21, 885–897 (2005)
31. van den Berg, J., Patil, S., Sewall, J., Manocha, D., Lin, M.C.: Interactive navigation of multiple agents in crowded environments. In: ACM Symp. on I3D, pp. 139–147 (2008)

Multi-robot Coverage and Exploration in Non-Euclidean Metric Spaces*

Subhrajit Bhattacharya, Robert Ghrist, and Vijay Kumar

Abstract. Multi-robot coverage and exploration is a fundamental problem in robotics. A widely-used, efficient and distributable algorithm for achieving coverage of a convex environment with Euclidean metric is that proposed by Cortes, *et al.*, which is based on the discrete-time Lloyd's algorithm. It is significantly difficult to achieve the same in non-convex environments and with non-Euclidean metrics. In this paper we generalize the control law based on minimization of the *coverage functional* to spaces that are inherently non-Euclidean and are punctured by obstacles. We also propose a practical discrete implementation based on standard graph search-based algorithms. We demonstrate the applicability of the proposed algorithm by solving efficient coverage problems on a sphere and exploration problems in highly non-convex indoor environments.

1 Introduction

The geometry underlying configuration spaces of multiple robots is a critical feature implicit in several important challenges in planning and coordination. Metric considerations are fundamental to problems of coverage [12, 6, 7, 3], exploration [16, 15, 14], and more. A well-known approach to solving coverage problems with n robots involve partitioning the appropriate configuration space into *n-tessellations* (a partition of the configuration space into simply-connected domains) [12, 6]. In particular, this method requires a Voronoi tessellation that implicates the configuration space geometry. While such a tessellation is easy to achieve in a convex environment with Euclidean metric, it becomes increasingly difficult in

Subhrajit Bhattacharya · Robert Ghrist · Vijay Kumar
GRASP Laboratory,
University of Pennsylvania
e-mail: {subhrabh, ghrist, kumar}@seas.upenn.edu

* We gratefully acknowledge the support of ONR Grants N00014-07-1-0829 and N00014-09-1-1031, and AFOSR Grant FA9550-10-1-0567.

environments with obstacles and non-Euclidean metrics. The ubiquitous presence of obstacles removes all hope of convexity. Non-Euclidean metrics can arise in the geometry of a configuration space as inherited from the structure of the underlying domain (*e.g.*, from irregular terrain), or via direct manipulation of the configuration space geometry for problem goals (*e.g.*, in multi-robot cooperative exploration problems [2]).

The problem of attaining balanced coverage of an environment is fundamental to many practical multi-robot problems. One common coverage control approach — efficient and distributable — is through the definition of feedback control laws defined with respect to the centroids of Voronoi cells resulting from the Voronoi tessellation of the domain. Lloyd’s algorithm [12] is a discrete-time algorithm that minimizes a *coverage functional*. A continuous-time version of the algorithm is described in [5], where the authors propose gradient descent-based individual robot control laws that guarantee optimal coverage of a convex environment given a density function which represents the desired coverage distribution. To address the limitation of requiring a convex environment, the authors of [13] propose the use of *geodesic Voronoi tessellations* determined by the geodesic distance rather than the Euclidean distance. However such a method both involves computationally difficult geometric computations and is still limited to Euclidean environments with polygonal obstacles. Recent work [2] has used a graph search-based approach to develop tools for solving the coverage problem in non-convex environments with a non-Euclidean metric. However, in order to explicitly compute an analogue of a *generalized centroid* in non-convex tessellations, an approximate method involving *centroid projection* was used. Such a method is, admittedly, ad hoc, gives weak guarantees, and is difficult to implement when the configuration space is not sufficiently topologically simple (equivalent to a punctured simply-connected domain). There exists search-based discrete-time algorithms that explicitly searches every vertex in a tessellation to find the best position for the robot in every time-step (as in [10]). Although such a controller can solve the problem of multi-robot decentralized coverage on arbitrary metric graphs, the high computational complexity of this approach makes it impractical for fine discretization or large graphs.

In this paper we generalize the method for computing the control law that is described in [13] and adapt it to non-Euclidean metric spaces with obstacles that are not necessarily polygonal. The principal theoretical tools are Proposition 1 and Corollary 1, which relate geodesics, distance derivatives, and the metric tensor. This inspires the use of the control law in concert with a graph search-based methods to achieve an efficient discrete implementation. Our results are supported by the following demonstrations:

1. Coverage problems on non-Euclidean Riemannian manifolds with boundaries;
2. Multi-robot cooperative exploration; and
3. Human-robot interaction in exploration.

2 Background: Coverage Functional, Voronoi Tessellation and Continuous-Time Lloyd's Algorithm

In this section we discuss the basic concepts behind deriving the control laws in the continuous-time version of the Lloyd's algorithm [12, 13]. Let Ω be a path-connected metric space that represents the environment, equipped with a distance function, d . In [12, 13] Ω is assumed to be a subset of \mathbb{R}^D and is equipped with the Euclidean metric tensor (a Riemannian metric) at every point. However, in the present scenario we will relax d to a more general class of distance functions (for example, the *path metric* induced by general Riemannian metric – *i.e.* the length of the shortest rectifiable path [11]. For Corollary 1, we will in fact be able to consider more general Finsler metrics.).

There are n mobile robots in the environment, and in particular the position of the k^{th} robot is represented by $\mathbf{p}_k \in \Omega$ and the tessellation [12, 13] associated with it by W_k , $\forall k = 1, 2, \dots, n$ (Thus, $\{W_k\}_{k=1,\dots,n}$ is a cover of Ω). By definition, the tessellations are such that $\text{Int}(W_k) \cap \text{Int}(W_l) = \emptyset, \forall k \neq l$, and $\cup_{k=1}^n W_k = \Omega$. For a given set of robot positions $P = \{\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_n\}$ and tessellations $W = \{W_1, W_2, \dots, W_n\}$ such that $\mathbf{p}_k \in \text{Int}(W_k), \forall k = 1, 2, \dots, n$, the *coverage functional* is defined as:

$$\mathcal{H}(P, W) = \sum_{k=1}^n \mathcal{H}(\mathbf{p}_k, W_k) = \sum_{k=1}^n \int_{W_k} f_k(d(\mathbf{q}, \mathbf{p}_k)) \phi(\mathbf{q}) \, d\mathbf{q} \quad (1)$$

where $f_k : \mathbb{R}^+ \rightarrow \mathbb{R}$ are smooth and strictly increasing functions, $\phi : \Omega \rightarrow \mathbb{R}$ is a weight or density function, and $d\mathbf{q}$ represents an infinitesimal volume element.

The name “*coverage functional*” is indicative of the fact that \mathcal{H} measures how *bad* the coverage is. In fact, for a given set of initial robot positions, P , the eventual aim of the algorithm is to devise a control law that minimizes the function $\tilde{\mathcal{H}}(P) := \min_W \mathcal{H}(P, W)$ (*i.e.* the best value of $\mathcal{H}(P, W)$ for a given P). It is easy to show [12, 13] that $\tilde{\mathcal{H}}(P) = \mathcal{H}(P, V)$, where $V = \{V_1, V_2, \dots, V_n\}$ is the Voronoi tessellation given by

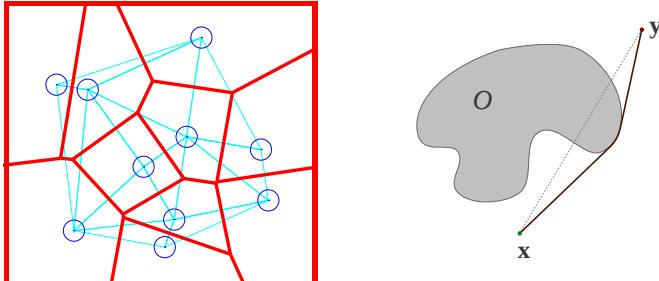
$$V_k = \{\mathbf{q} \in \Omega \mid f_k(d(\mathbf{q}, \mathbf{p}_k)) \leq f_l(d(\mathbf{q}, \mathbf{p}_l)), \forall l \neq k\} \quad (2)$$

Thus the control law for minimizing $\tilde{\mathcal{H}}(P) = \sum_{k=1}^n \int_{V_k} f_k(d(\mathbf{q}, \mathbf{p}_k)) \phi(\mathbf{q}) \, d\mathbf{q}$ can be reduced to the problem of following its gradient. Although V_k are functions of P , it can be shown using methods of differentiation under integration [13] that

$$\frac{\partial \tilde{\mathcal{H}}(P)}{\partial \mathbf{p}_k} = \int_{V_k} \frac{\partial}{\partial \mathbf{p}_k} f_k(d(\mathbf{q}, \mathbf{p}_k)) \phi(\mathbf{q}) \, d\mathbf{q} \quad (3)$$

In practice, it is adequate to choose $f_k(x) = x^2$ for most practical implementations. However, a variation of the problem for taking into account finite sensor footprint of the robots, constructs a *power Voronoi tessellation* [13], in which one chooses $f_k(x) = x^2 - R_k^2$, where R_k can represent, for example, the radius of the sensor footprint of the k^{th} robot. In this paper we will be working with the following form for f_k

$$f_k(x) = x^2 + c_k \quad (4)$$



(a) Voronoi tessellation of a convex Ω (rectangular region) with $n = 10$ robots (blue circles). The red line segments show the boundary of the tessellations. Note how a boundary segment is the perpendicular bisector of the cyan line joining the robots sharing the boundary segment.

(b) Presence of holes/punctures (due to obstacles) in the Euclidean space changes the distance function in the punctured space. For example, in this figure, $d(\mathbf{x}, \mathbf{y})$ is the length (induced by the Euclidean metric) of the thicker curve that is the shortest connecting \mathbf{x} and \mathbf{y} and lying entirely in $(\mathbb{R}^2 - O)$.

Fig. 1 Voronoi tessellation in a convex environment, and the difficulty in computing it in non-convex environments

2.1 Euclidean and Non-Euclidean Metric Spaces

Until now we haven't made any major assumption on the distance function d . However, if the space Ω is convex, and the metric η is Euclidean, then d is the Euclidean distance given by $d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_2$. Under this assumption, and using the form of f_k in (4), the formula of (3) can be simplified to obtain

$$\frac{\partial \tilde{\mathcal{H}}(P)}{\partial \mathbf{p}_k} = 2A_k(\mathbf{p}_k - \mathbf{p}_k^*) \quad (5)$$

where, $A_k = \int_{V_k} \phi(\mathbf{q}) d\mathbf{q}$ is the weighted *volume* of V_k , and $\mathbf{p}_k^* = \frac{\int_{V_k} \mathbf{q} \phi(\mathbf{q}) d\mathbf{q}}{A_k}$ is the weighted centroid of V_k . Moreover, the Euclidean distance function makes computation of the Voronoi tessellation very easy: V , due to Equation (2), can be constructed from the perpendicular bisectors of the line segments $\overline{\mathbf{p}_k \mathbf{p}_l}$, $\forall k \neq l$, thus making each V_k a convex polygon, which are also simply connected (Figure 1(a)). This also enable closed-form computation of the volume, A_k , and the centroid, \mathbf{p}_k^* when the weight function ϕ is uniform. Equation (5) yields the simple control law in continuous-time Lloyd's algorithm: $\mathbf{u}_k = -\kappa A_k(\mathbf{p}_k - \mathbf{p}_k^*)$, with some positive *gain*, κ . Lloyd's algorithm [12] and its continuous-time asynchronous implementations [5] are distributed algorithms for minimizing $\mathcal{H}(P, W)$ with guarantees on completeness and asymptotic convergence to a local optimum, when Ω is convex Euclidean.

However, the above simplification does not work when the distance function is not Euclidean. In robot configuration spaces punctured by obstacles, non-Euclidean distance functions can appear in the computations in two primary ways: **i.** Due to the presence of holes/obstacles, even when the path-metric is locally Euclidean in the interior of the domain, the global distance function is not Euclidean (Figure 1(b)), and **ii.** Locally non-Euclidean metric (*e.g.* in the case of a sphere, or a topologically Euclidean space with non-zero curvature). This makes the computation of the Voronoi tessellation in Equation (2) as well as the gradient of $\tilde{\mathcal{H}}$ in Equation (3) significantly difficult.

In Equation (3), with $f_k(x) = x^2 + c_k$, we observe that

$$\frac{\partial}{\partial \mathbf{p}_k} f_k(d(\mathbf{q}, \mathbf{p}_k)) = 2 d(\mathbf{q}, \mathbf{p}_k) \frac{\partial}{\partial \mathbf{p}_k} d(\mathbf{q}, \mathbf{p}_k)$$

Thus, our first step will be to find an efficient way for computing $\frac{\partial}{\partial \mathbf{p}_k} d(\mathbf{q}, \mathbf{p}_k)$. In Section 3 we will show that in arbitrary metric spaces (satisfying certain conditions) this gradient of the distance function can be expressed in terms of tangents to geodesics. In Section 5 we will demonstrate how the later is computationally more favorable in a graph search-based algorithm for developing a generalized continuous-time Lloyd's algorithm.

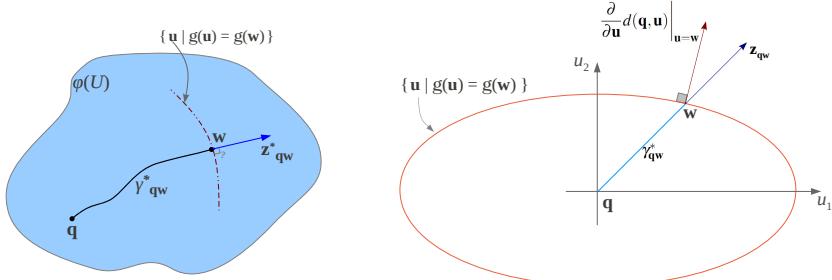
3 Gradient of Distance Function in Non-Euclidean Metric Spaces

For computing $\frac{\partial}{\partial \mathbf{p}_k} d(\mathbf{q}, \mathbf{p}_k)$ we have the following proposition and corollary, which together gives a generalization of Proposition 4 of [13]. In the discussions that follow, we will assume summation over repeated indices, i and j , following the Einstein summation convention.

In Proposition 1 we essentially establish a relationship between gradient of the distance function and the tangent to a geodesic, provided the distance function satisfies some very restricted conditions (geodesics being unique and induced by Riemannian metric). In most robot configuration spaces, due to presence of non-convexity and obstacles, this proposition will not be sufficient. Thus we devise Corollary 1 to encompass a wider class of distance functions.

Proposition 1. *Let $C = (U, \phi)$ be a coordinate chart on a open subset U of a D -dimensional manifold, Ω with coordinate variables u^1, u^2, \dots, u^D . Suppose U is Riemannian everywhere, equipped with a metric η , and the geodesic connecting any two points in U lies entirely in U .*

*Let $d : \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}$ be the distance function in $U \subseteq \Omega$ in terms of the coordinate chart C (*i.e.* $d(\mathbf{q}, \mathbf{w})$, for $\mathbf{q}, \mathbf{w} \in \text{Img}(\phi) \subseteq \mathbb{R}^D$, is the length of the shortest path connecting $\phi^{-1}(\mathbf{q})$ and $\phi^{-1}(\mathbf{w})$ in $U \subseteq \Omega$). Suppose inside $\text{Img}(\phi)$, the distance d is induced by the Riemannian metric η , is smooth everywhere, and suppose there exists a unique geodesic of length $d(\mathbf{q}, \mathbf{w})$ connecting any two points $\mathbf{q}, \mathbf{w} \in \text{Img}(\phi)$.*



(a) Illustration for Proposition 1. It establishes a relation between the tangent to the geodesic $\gamma_{\mathbf{q}\mathbf{w}}^*$ at \mathbf{w} , and the normal to the surface $\{\mathbf{u} | g(\mathbf{u}) = g(\mathbf{w})\}$ at \mathbf{w} .

(b) Illustration with a simple non-Euclidean, anisotropic metric. Note that the normal to the ellipse is not parallel to the tangent to the geodesic, $\mathbf{z}_{\mathbf{q}\mathbf{w}}$. It is however parallel to the cotangent, $\mathbf{z}_{\mathbf{w}\mathbf{q}}^*$, with coefficients $z_{j,\mathbf{q}\mathbf{w}}^* = \eta_{ij}(\mathbf{w}) z_{\mathbf{q}\mathbf{w}}^j$.

Fig. 2 Relationship between tangent to a geodesic and the derivative of the distance function

Then the following is true for every $\mathbf{q}, \mathbf{w} \in \text{Img}(\phi) \subseteq \mathbb{R}^D$ and every coordinate chart, C , defined on U (Figure 2(a)),

$$\left[\frac{\partial}{\partial \mathbf{u}} d(\mathbf{q}, \mathbf{u}) \Big|_{\mathbf{u}=\mathbf{w}} \right]_i \equiv \frac{\partial}{\partial u^i} d(\mathbf{q}, \mathbf{u}) \Big|_{\mathbf{u}=\mathbf{w}} = \frac{\eta_{ij}(\mathbf{w}) z_{\mathbf{q}\mathbf{w}}^j}{\sqrt{\eta_{mn}(\mathbf{w}) z_{\mathbf{q}\mathbf{w}}^m z_{\mathbf{q}\mathbf{w}}^n}}$$

where, $\mathbf{z}_{\mathbf{q}\mathbf{w}} = [z_{\mathbf{q}\mathbf{w}}^1, z_{\mathbf{q}\mathbf{w}}^2, \dots, z_{\mathbf{q}\mathbf{w}}^D]^T$ is a normalized coefficient vector of the tangent vector at \mathbf{w} to the shortest geodesic connecting \mathbf{q} to \mathbf{w} , and by $\left[\frac{\partial f}{\partial \mathbf{u}} \right]_i$ we mean the i^{th} component of $\left[\frac{\partial f}{\partial u^1}, \frac{\partial f}{\partial u^2}, \dots, \frac{\partial f}{\partial u^D} \right]$.

Proof. This result follows from well-known theorems in Riemannian geometry. A detailed proof is provided in [1]. \square

If we define $g(\mathbf{u}) := d(\mathbf{q}, \mathbf{u})$, $\forall \mathbf{u} \in \text{Img}(\phi)$ (i.e., $g(\mathbf{w})$ is the length of the shortest geodesic connecting \mathbf{q} to \mathbf{w}), the statement of the proposition essentially implies that the normals to the constant g surfaces in \mathbb{R}^D are parallel to the geodesics. This is illustrated in Figure 2(a). The statement of the proposition essentially expresses the gradient of the distance function d (with respect to one of its arguments) in terms of the tangent to the geodesic connecting two points.

Examples

1. We note that when the metric is Euclidean in the given chart (i.e. $\eta_{ij} = \delta_{ij}$ everywhere as was the case in [13]), the result of the proposition simply reduces to $\frac{\partial}{\partial u^i} d(\mathbf{q}, \mathbf{u}) \Big|_{\mathbf{u}=\mathbf{w}} = z_{\mathbf{q}\mathbf{w}}^i$. This is no surprise since we know that the vector $\frac{\partial}{\partial \mathbf{u}} d(\mathbf{q}, \mathbf{u}) \Big|_{\mathbf{u}=\mathbf{w}}$ is essentially a unit normal to the sphere with center \mathbf{q} (which

is the surface of constant $d(\mathbf{q}, \mathbf{u})$ at the point $\mathbf{u} = \mathbf{w}$, which is well-known to be parallel to the straight line connecting \mathbf{q} to \mathbf{w} (a radial line of the sphere).

2. If the metric is locally isotropic in the given chart (*i.e.* if the matrix representation of the metric is a multiple of the identity matrix at every point), and can be written as $\eta_{ij}(\mathbf{q}) = \zeta(\mathbf{q})\delta_{ij}$ for some $\zeta : \mathbb{R}^D \rightarrow \mathbb{R}$, then the result of the proposition reduces to $\frac{\partial}{\partial \mathbf{u}} d(\mathbf{q}, \mathbf{u}) \Big|_{\mathbf{u}=\mathbf{w}} = \sqrt{\zeta(\mathbf{w})} \mathbf{z}_{\mathbf{qw}}^T$ (where, $\mathbf{z}_{\mathbf{qw}}^T = [z_{\mathbf{qw}}^1, z_{\mathbf{qw}}^2, \dots, z_{\mathbf{qw}}^D]$ is the transpose of the coefficient vector, $\mathbf{z}_{\mathbf{qw}}$, of the tangent to the geodesic).
3. Finally, we consider a simple, yet nontrivial, example of a non-Euclidean, anisotropic metric. Consider the metric $\eta_{\bullet\bullet} = \begin{bmatrix} 1 & 0 \\ 0 & 4 \end{bmatrix}$. Since the Christoffel symbols vanish in this coordinate chart, one can infer from the geodesic equation that the geodesics are essentially represented by straight lines when plotted with u^i as orthogonal axes (Figure 2(b)). However, the curves of constant distance from \mathbf{q} become ellipses centered at \mathbf{q} and with aspect ratio of 2. Now consider the point $\mathbf{w} = \mathbf{q} + [1, 1]^T$. A direct computation of the normal at this point to the ellipse, $(u^1 - q^1)^2/4 + (u^2 - q^2)^2 = c$, passing through this point, reveals the coefficient co-vector of $\frac{\partial}{\partial \mathbf{u}} d(\mathbf{q}, \mathbf{u}) \Big|_{\mathbf{u}=\mathbf{w}}$ to be parallel to $[\frac{1}{2}, 2]$. However, the coefficient vector of the tangent to the geodesic is $\mathbf{z}_{\mathbf{qw}} = [\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}]^T$. This gives the following:
 $\sqrt{\eta_{mn}(\mathbf{w}) z_{\mathbf{qw}}^m z_{\mathbf{qw}}^n} = \sqrt{\frac{5}{2}}, \quad z_{1,\mathbf{qw}} = \sum_j \eta_{1j} z_{\mathbf{qw}}^j = \frac{1}{\sqrt{2}}, \quad z_{2,\mathbf{qw}} = \sum_j \eta_{2j} z_{\mathbf{qw}}^j = 2\sqrt{2}$.
Thus, the coefficient co-vector of $\mathbf{z}_{\mathbf{qw}}^*$ is parallel to $[\frac{1}{\sqrt{2}}, 2\sqrt{2}]$. This indeed is parallel to $[\frac{1}{2}, 2]$. The exact computation of the scalar multiple will require a more careful computation of $\frac{\partial}{\partial \mathbf{u}} d(\mathbf{q}, \mathbf{u})$.

Corollary 1. Let $C = (V, \psi)$ be a coordinate chart on a open subset V of a D -dimensional manifold, Ω . Let $d : \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}$ be the distance function on V in terms of the coordinate chart C (*i.e.* $d(\mathbf{q}, \mathbf{w})$, for $\mathbf{q}, \mathbf{w} \in \text{Img}(\psi) \subseteq \mathbb{R}^D$, is the distance between $\psi^{-1}(\mathbf{q})$ and $\psi^{-1}(\mathbf{w})$ in $V \subseteq \Omega$).

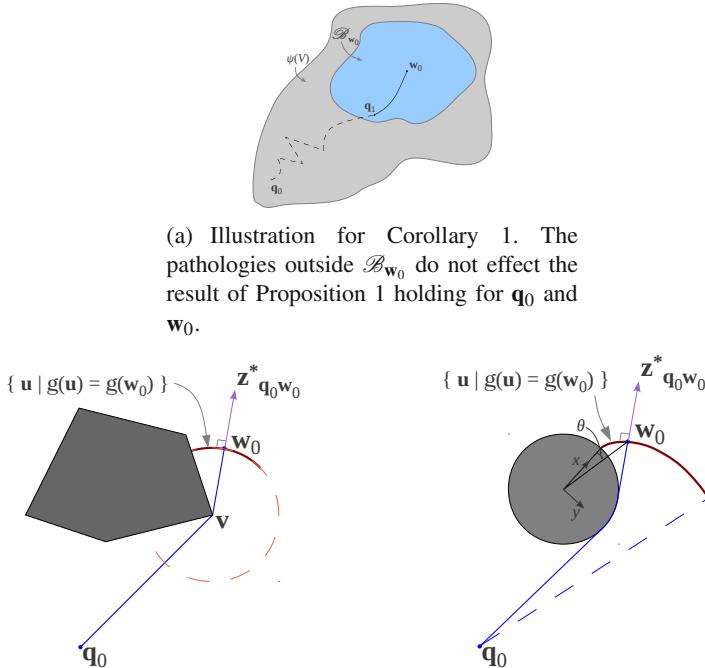
We are given $\mathbf{q}_0, \mathbf{w}_0 \in \text{Img}(\psi) \subseteq \mathbb{R}^D$. Suppose there exists a open neighborhood $\mathcal{B}_{\mathbf{w}_0} \subseteq \mathbb{R}^D$ of \mathbf{w}_0 (Figure 3(a)) such that,

- a. $g(\cdot) := d(\mathbf{q}_0, \cdot)$ is smooth everywhere in $\mathcal{B}_{\mathbf{w}_0}$,
- b. The distance function restricted to $\mathcal{B}_{\mathbf{w}_0} \times \mathcal{B}_{\mathbf{w}_0}$ is induced by a Riemannian metric, η , such that for any two $\mathbf{u}, \mathbf{v} \in \mathcal{B}_{\mathbf{w}_0}$, there is a unique shortest geodesic $\gamma_{\mathbf{uv}}$ of length $d(\mathbf{u}, \mathbf{v})$.
- c. A shortest path (of length $d(\mathbf{q}_0, \mathbf{w}_0)$) is defined between \mathbf{q}_0 and \mathbf{w}_0 , such that the part of the shortest path connecting \mathbf{q}_0 and \mathbf{w}_0 that lies inside $\mathcal{B}_{\mathbf{w}_0}$ is unique,

Then the following holds,

$$\left[\frac{\partial}{\partial \mathbf{u}} d(\mathbf{q}_0, \mathbf{u}) \Big|_{\mathbf{u}=\mathbf{w}_0} \right]_i \equiv \frac{\partial}{\partial u^i} d(\mathbf{q}_0, \mathbf{u}) \Big|_{\mathbf{u}=\mathbf{w}_0} = \frac{\eta_{ij}(\mathbf{w}_0) z_{\mathbf{q}_0 \mathbf{w}_0}^j}{\sqrt{\eta_{mn}(\mathbf{w}_0) z_{\mathbf{q}_0 \mathbf{w}_0}^m z_{\mathbf{q}_0 \mathbf{w}_0}^n}}$$

Note that the derivative $\frac{\partial}{\partial \mathbf{u}} d(\mathbf{q}_0, \mathbf{u}) \Big|_{\mathbf{u}=\mathbf{w}_0}$ is defined due to assumption ‘a.’, and the tangent $\mathbf{z}_{\mathbf{q}_0 \mathbf{w}_0}$ exists due to assumptions ‘b.’ and ‘c.’.



(a) Illustration for Corollary 1. The pathologies outside \mathcal{B}_{w_0} do not effect the result of Proposition 1 holding for q_0 and w_0 .

(b) The simplest example is that of a space that is equipped with Euclidean metric everywhere, but is punctured by a polygonal obstacles. This was the case considered in [13].

(c) A more interesting case is that of involutes generated in locally Euclidean space using the boundaries of arbitrary obstacles as the generating curves.

Fig. 3 Corollary 1 and examples demonstrating it

Proof. This result follows from well-known theorems in Riemannian geometry and from Proposition 1. A detailed proof is provided in [1]. \square

The statement of this Corollary encompasses a significantly wider class of metric spaces than Proposition 1. Here we only need to assume a Riemannian metric in the neighborhood of w_0 (Figure 3(a)). This will enable us to use the result for locally Riemannian manifolds with pathologies outside local neighborhoods (*e.g.* boundaries/holes/punctures/obstacles – the kind of spaces we are most interested in), as well as opens up possibilities for more general metric spaces that may not be Riemannian outside \mathcal{B}_{w_0} (*e.g.* Manhattan metric in $\mathcal{M} \subset \Omega$, Riemannian metric elsewhere).

Examples

1. The simplest example is that of a space that is locally Euclidean (*i.e.* equipped with a Euclidean metric everywhere), but is punctured by polygonal obstacles (Figure 3(b)). Due to the ‘pointedness’ of the obstacles, the constant- g manifolds

are essentially circular arcs centered at \mathbf{q}_0 or a vertex \mathbf{v} of a polygon. Thus, as illustrated by Figure 3(b), the normals to the arcs are parallel to the tangent to the segment joining \mathbf{v} to \mathbf{w}_0 .

2. A little less trivial example occurs when the obstacles are not polygonal. Then the statement of the corollary essentially reduces to the assertion that the normal at any point on an involute [8] is parallel to the ‘taut string’, the end of which traces the involute – and this is true irrespective of the curve used to generate the involute. While the statement has an obvious intuitive explanation by considering the possible directions of motion of the end of the taut string, we provide an explicit computation for an involute created using a circle (Figure 3(c)). Consider a taut string unwrapping off a circle of radius r (starting from $\theta = 0$ when it is completely wrapped). Thus, when the string has unwrapped by an angle θ , the string points at a direction $[\sin(\theta), -\cos(\theta)]^T$. Now, it is easy to verify that the involute is described by the parametric curve $x = r(\cos(\theta) + \theta \sin(\theta))$, $y = r(\sin(\theta) - \theta \cos(\theta))$. Thus we have $\frac{dx}{d\theta} = \theta \cos(\theta)$, $\frac{dy}{d\theta} = \theta \sin(\theta)$. Thus the normal to the involute pointing in the direction $[\frac{dy}{d\theta}, -\frac{dx}{d\theta}]$ is indeed parallel to the direction in which the string points.

4 Generalized Continuous-Time Lloyd’s Algorithm

Corollary 1, along with the assumption that $f_k(x)$ is of the form $x^2 + c$, enables us to re-write Equation (3) for the most general metric setup as follows

$$\left[\frac{\partial \tilde{\mathcal{H}}(P)}{\partial \mathbf{p}_k} \right]_i = 2 \int_{V_k} d(\mathbf{q}, \mathbf{p}_k) \frac{\eta_{ij}(\mathbf{p}_k) z_{\mathbf{q}\mathbf{p}_k}^j}{\sqrt{\eta_{mn}(\mathbf{p}_k) z_{\mathbf{q}\mathbf{p}_k}^m z_{\mathbf{q}\mathbf{p}_k}^n}} \phi(\mathbf{q}) d\mathbf{q} \quad (6)$$

This, as we will see later, gives an algorithm for approximately computing the gradient of $\tilde{\mathcal{H}}$. Note that $d\mathbf{q}$ represents an infinitesimal volume element. Thus, in a discretized setup (with uniform discretization of the coordinate space, which we will discuss in the next section), when we replace the integral by a summation over the vertices of a graph, we need to use the appropriate volume representing $d\mathbf{q}$ for each discretized cell. In particular, a uniform discretization of the coordinate space implies we use $\sqrt{\det(\eta_{\bullet\bullet}(\mathbf{q}))}$ (up to a constant scalar multiple) for volume of each discretized cell.

Once we have the gradient of $\tilde{\mathcal{H}}$, the control law for minimizing $\tilde{\mathcal{H}}$ would simply be for k^{th} robot to move in a direction opposite to the gradient

$$\mathbf{u}_k = -\kappa \frac{\partial \tilde{\mathcal{H}}(P)}{\partial \mathbf{p}_k} \quad (7)$$

This give a generalized Lloyd’s algorithm with guarantee of asymptotic stability (as easily seen by considering $\tilde{\mathcal{H}}$ a Liapunov function candidate, and noting that its domain is a manifold). In the final converged solution, each robot will be at the *generalized centroid* of its related Voronoi tessellation (since that’s when the gradient of $\tilde{\mathcal{H}}$ vanishes).

With the assumption of isotropy of the metric in the given chart (which will hold in most of the exploration applications we will be describing), this further reduces to

$$\frac{\partial \tilde{\mathcal{H}}(P)}{\partial \mathbf{p}_k} = 2 \sqrt{\zeta(\mathbf{p}_k)} \int_{V_k} d(\mathbf{q}, \mathbf{p}_k) \mathbf{z}_{\mathbf{q}\mathbf{p}_k}^T \phi(\mathbf{q}) d\mathbf{q} \quad (8)$$

Typically, since we will be computing the control commands in a discrete setup, we are mostly interested in the direction of \mathbf{u}_k rather than its magnitude. Moreover, the metric in the given chart will be isotropic in most practical robot planning problems. Thus we clump the leading term $2 \sqrt{\zeta(\mathbf{p}_k)}$ of (8) inside κ to obtain the following control law

$$\mathbf{u}_k = -\kappa \int_{V_k} d(\mathbf{q}, \mathbf{p}_k) \mathbf{z}_{\mathbf{q}\mathbf{p}_k} \phi(\mathbf{q}) d\mathbf{q} \quad (9)$$

5 Graph Search-Based Implementation

In order to develop a version of the generalized continuous-time Lloyd's algorithm for a general distance function, we first need to be able to compute the general Voronoi tessellation of Equation (2) for arbitrary distance function, d . We adopt a discrete graph-search based approach for achieving that, not very unlike the approach taken in previous work [2]. We consider a uniform square tiling (Figure 4(a)) of the space of coordinate variables (in a particular coordinate chart) and creating a graph G out of it (with vertex set $\mathcal{V}(G)$, edge set $\mathcal{E}(G) \subseteq \mathcal{V}(G) \times \mathcal{V}(G)$ and cost function $\mathcal{C}_G : \mathcal{E}(G) \rightarrow \mathbb{R}^+$). The costs/weights of the edges of the graph are the metric lengths of the edges. It is to be noted that in doing so we end up restricting the metric of the original space to the discrete graph. Because of this, as well as due to the discrete computation of the integrations (as discussed later), this discrete graph-search based approach is inherently an approximate method, where we trade off the accuracy and elegance of a continuous space for efficiency and computability with arbitrary metric.

The key idea is to make a basic modification to Dijkstra's algorithm [9, 4]. This enables us to create a geodesic Voronoi tessellation. For creating Voronoi tessellations we initiate the *open set* with multiple start nodes from which we start propagation of the wavefronts. Thus the wavefronts emanate from multiple sources. The places where the wavefronts collide will hence represent the boundaries of the Voronoi tessellations. In addition, we can conveniently alter the distance function, the level-set of which represents the boundaries of the Voronoi tessellations. This enables us to even create *geodesic power Voronoi tessellation*. Figure 4(b) illustrates the progress of the algorithm in creation of the tessellations.

In order to compute the control command for the robots (*i.e.* the action of the robot in the next time step), we use the formula in Equation (9). In a general metric setup, the vector $\mathbf{z}_{\mathbf{q}\mathbf{p}_k}$ is the unit vector along the tangent at \mathbf{p}_k to the geodesic joining \mathbf{q} to \mathbf{p}_k . In a discretized setup, the position of the k^{th} robot corresponds to a vertex $p_k \in \mathcal{V}(G)$. Then, in the graph, the unit vectors $\mathbf{z}_{\mathbf{q}\mathbf{p}_k}$ is approximated as the unit vectors along edges of the form $[p'_k, p_k] \in \mathcal{E}(G)$ for some $p'_k \in \mathcal{N}_G(p_k)$ (the set of

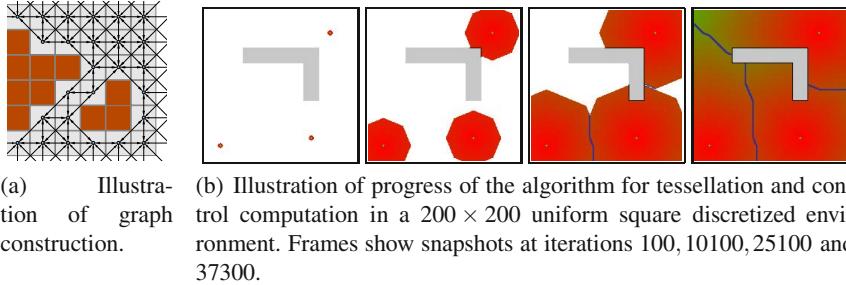


Fig. 4 (a): An 8-connected grid graph created from a uniformly discretized coordinate space. The brown cells represent obstacles. **(b):** Progress of the algorithm for tessellation and control computation in an environment with a L-shaped obstacle. The graph is constructed by 200×200 uniform square discretization of the environment (see [2]). Tessellations are created starting from three points (the location of the agents) to the complete diagram after expansion of about 37300 vertices. The filled area indicates the set of *expanded* vertices. The boundaries of the tessellations are visible in blue.

neighbors of p_k) such that the shortest path in the graph connecting p_k and q passes through p'_k . For a given q , we keep track of the index of the robot whose tessellation it belongs to, as well as the neighbor of the corresponding robot's vertex through which the shortest path leading to q passes. Thus, we can also compute the integration of (9) on the fly as we compute the tessellations. The complete pseudocode for the algorithm can be found in [1].

The complexity of the algorithm is the same as the standard Dijkstr'a's algorithm, which for a constant degree graph is $O(V_G \log(V_G))$ (where $V_G = |\mathcal{V}(G)|$ is the number of vertices in the graph). This is in sharp contrast to the complexity of search for optimal location as in [10].

5.1 Application to Coverage on Non-Euclidean Metric Spaces

In this section we will illustrate examples of coverage using the generalized continuous-time Lloyd's algorithm on a 2-sphere. We use a coordinate chart with coordinate variables $x \in (0, \pi)$, the latitudinal angle, and $y \in [0, 2\pi]$, the longitudinal angle (Figure 5(a)). The matrix representation of the metric on the sphere using this coordinate chart is $\eta_{\bullet\bullet} = \begin{bmatrix} 1 & 0 \\ 0 & \sin^2(x) \end{bmatrix}$. As usual, we use a uniform square discretization of the coordinate space to create an 8-connected grid graph [2]. However, in order to model the complete sphere (in the example of Figure 6), we need to establish appropriate edges between vertices at the extreme values of y , i.e. the ones near $y = 0$ and $y = 2\pi$. Similarly, we use an additional vertex for each pole to which the vertices corresponding to the extreme values of x connect.

Figure 5(b)-(d) shows two robot attaining coverage on a geodesically convex subset of the 2-sphere by following the control command computed using the algorithm

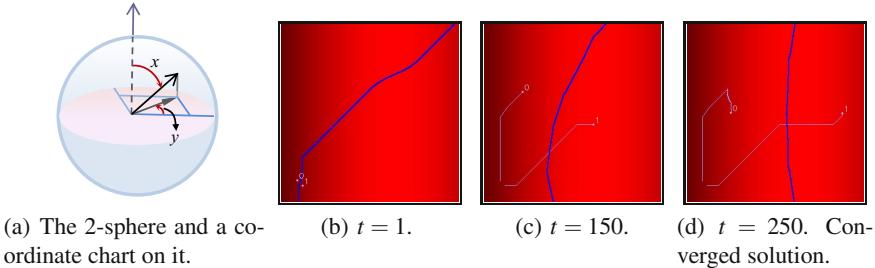


Fig. 5 Coverage using discrete implementation of generalized continuous-time Lloyd's algorithm on a part of the 2-sphere. The chosen coordinate variables, x and y , are the latitudinal and longitudinal angles respectively, and the domain shown in figures (b)-(d) represent the region on the sphere where $x \in [\pi/16, 3\pi/4]$, $y \in [\pi/16, 3\pi/4]$. x is plotted along horizontal axis and y along vertical axis on linear scales. The intensity of red indicates the determinant of the round metric, thick blue curves are the tessellation boundaries, and the thin pale blue curves are the robot trajectories.

of Section 5 at every time-step. The region of the sphere that we restrict to is that of latitudinal angle $x \in [\pi/16, 3\pi/4]$, and longitudinal angle $y \in [\pi/16, 3\pi/4]$. The robots start off from the bottom left of the environment near the point $[0.42, 0.45]$, and follow the control law of Equation (7) until convergence is achieved. Note that the tessellations have a curved boundary in Figures 5(c) and 5(d) because it has to be a segment of the great circle on the sphere (note that the jaggedness is due to the fact that the curve actually resides on the discrete graph rather than the original metric space of the sphere). In the converged solution of Figure 5(d), note how the robots get placed such that their tessellation split up the area on the sphere equally rather than splitting up the area of the non-isometric embedding that depends on the chosen coordinate chart. The weight function is chosen to be constant, $\phi(\mathbf{q}) = 1$. For this example, the program ran at a rate of about 4 Hz on a machine with 2.1 GHz processor and 3 Gb memory.

A more complete example is shown in Figure 6 in which 4 robots attain coverage on a complete 2-sphere. The robots start off close to each other on the sphere, and follow the control law of Equation (7), until they converge attaining good and uniform coverage of the sphere. In order to avoid numerical problems near the poles, we ‘chop off’ small disks near the poles (marked by the gray regions), and establish ‘invisible’ edges across those disks connecting the vertices on their diametrically opposite points. Figures 6(a)-(d) show the tessellations of the robots in the coordinate chart with x plotted along the horizontal axis and y along the vertical axis (300×600 uniformly discretized). Figures 6(e)-(h) show the same tessellations mapped on the sphere. The weight function, once again is chosen to be $\phi(\mathbf{q}) = 1$. For this example, the program ran (control computation as well as plotting of the graphics) at a rate of about 1 Hz on a machine with 2.1 GHz processor and 3 Gb memory.

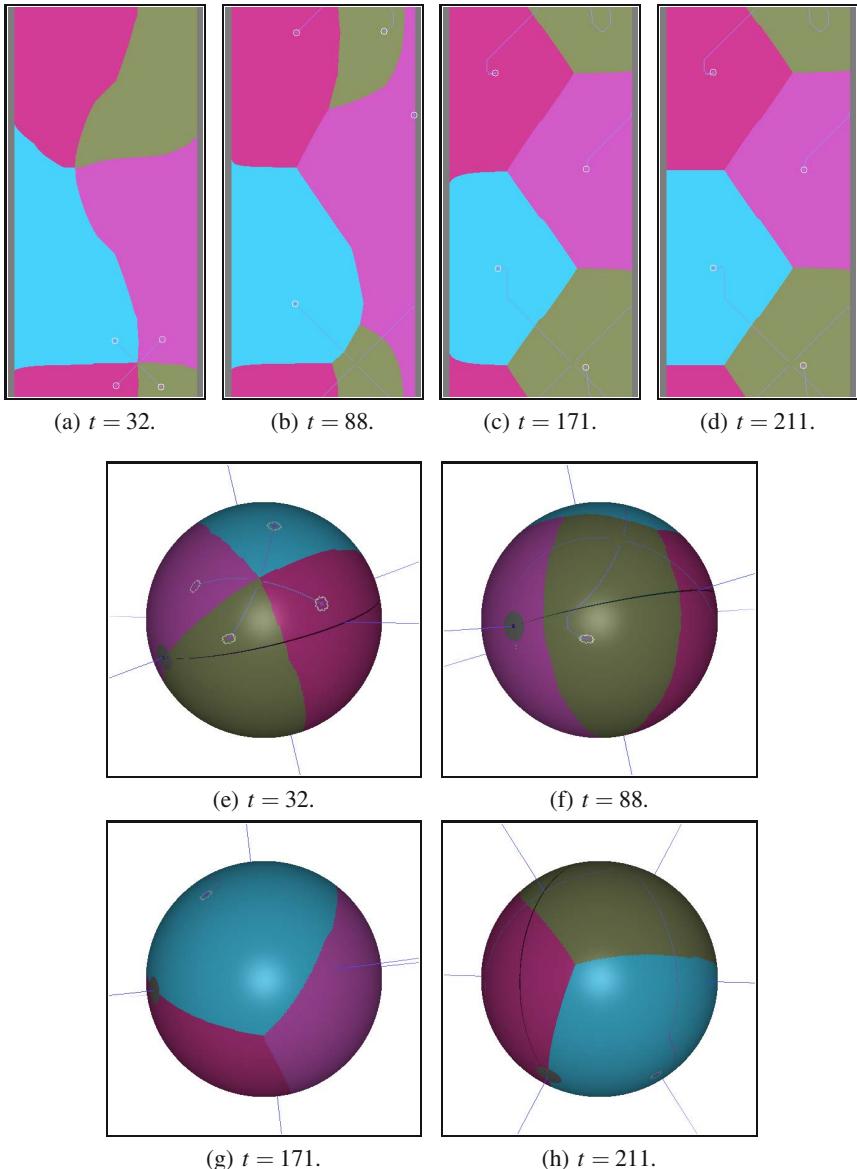


Fig. 6 Coverage on a complete sphere. In Figures (a)-(d), $x \in (0, \pi)$ (latitudinal angle) is plotted along horizontal axis and $y \in [0, 2\pi)$ (longitudinal angle) along vertical axis on linear scales. Figures (e)-(h) show the same plot mapped on the 2-sphere. The colors are used to indicate the tessellation of each robot. Note that in (e)-(h) different viewing angles are used to view the interesting parts of the sphere at a particular time-step. (d) and (h) are the converged solutions.

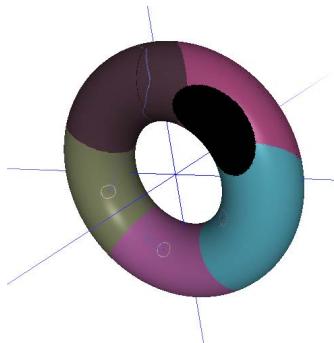


Fig. 7 Coverage on a 2-torus with obstacle on it (marked in black) attained by 5 robots

Figure 7 shows the final converged solution of a similar example of coverage on a 2-torus with obstacles by 5 robots. The metric tensor on the torus is given by $\eta_{\bullet\bullet} = \begin{bmatrix} r^2 & 0 \\ 0 & (R + r \cos x)^2 \end{bmatrix}$, where R is the radius of the axial circle, r the radius of the tube of the torus, x is the *latitudinal angle*, and y is the *longitudinal angle*. Note how the Voronoi tessellations in this case are not simply connected.

5.2 Application to Cooperative Exploration and Coverage Problem

Next we apply the tools developed to the problem of cooperative exploration and simultaneous coverage. In previous work [2] we had used an approximate and ad hoc “projection of centroid” method in order to compute an analog of *generalized centroid* and device a control law that was essentially to follow the approximate generalized centroid. However, now that we are equipped with the control law of Equation (7), we can achieve the same objectives in a more systematic way.

As detailed in [2], we choose Shannon entropy for constructing the density function as well as to weigh the metric (Figure 8). Thus, if $p(\mathbf{q})$ is the probability that the

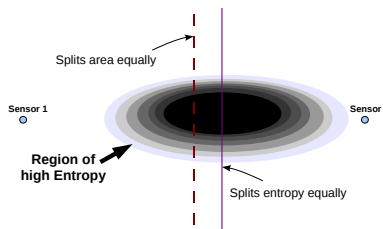


Fig. 8 Entropy-weighted metric for voronoi tessellation in exploration problem

vertex \mathbf{q} is inaccessible (*i.e.* occupied or part of an obstacle), for all $\mathbf{q} \in \mathcal{V}(G)$, then the Shannon entropy is given by $e(\mathbf{q}) = -(p(\mathbf{q}) \ln(p(\mathbf{q})) + (1-p(\mathbf{q})) \ln(1-p(\mathbf{q})))$. We use this for modeling the weight function, ϕ , and an isotropic metric, ζI (where I is the identity matrix). Noting that in an exploration problem, the occupancy probability, p , and hence the entropy e , will be functions of time as well, we use the following formulae for ϕ and ζ ,

$$\phi(\mathbf{q}, t) = \begin{cases} \varepsilon_\phi, & \text{if } e(\mathbf{q}, t) < \tau \\ e(\mathbf{q}, t), & \text{otherwise.} \end{cases}, \quad \zeta(\mathbf{q}, t) = \begin{cases} \varepsilon_\zeta, & \text{if } e(\mathbf{q}, t) < \tau \\ e(\mathbf{q}, t), & \text{otherwise.} \end{cases} \quad (10)$$

for some small ε_ϕ and ε_ζ representing zero (for numerical stability).

Each mobile robot maintains, updates and communicates a probability map for the discretized environment and updates its entropy map. We use a sensor model similar to that described in [2], as well as ‘freeze’ a vertex to prevent any change to its probability value when its entropy drops below some $\tau' (< \tau)$.

In addition, to avoid situations where a robot gets stuck at a local minima inside its tessellation even when there are vertices with entropy greater than τ in the tessellations (this can happen when there are multiple high entropy regions in the tessellations that exert equal and opposite pull on the robot so that the net velocity becomes zero), we perform a check on the value of the integral of the weight function, ϕ , within the tessellation of the k^{th} robot when its control velocity vanishes. If the integral is above the value of $\int_{V_k} \varepsilon_\phi \, d\mathbf{q}$, we switch to a greedy exploration mode where the k^{th} robot essential head directly towards the closest point that has entropy greater than the value of τ . This ensures exploration of the entire environment (*i.e.* the entropy value for every accessible vertex drops below τ). And once that is achieved, both ϕ and ζ become independent of time. Thus convergence is guaranteed.

Figure 9 shows screenshots of a team of 4 robots exploring a part of the 4th floor of the Levine building at the University of Pennsylvania. The intensity of white represents the value of entropy. Thus in Figure 9(a) the robots start of with absolutely no knowledge of the environment, explore the environment, and finally converge to a configuration attaining good coverage (Figure 9(d)).

Figure 10 shows a similar scenario. However, in this case one of the robots (Robot 0, marked by red circle) gets hijacked and manually controlled by a human user soon after they start cooperative exploration of the environment. That robot is forced to stay inside the larger room at the bottom of the environment. Moreover, in this case we use a team of heterogeneous robots (robots with different sensor footprint radii), thus requiring to compute *power voronoi tessellations*. This simple example illustrates the flexibility of our framework with respect to human-robot interaction.

For either of the above examples, with the environment 284×333 discretized, the program ran at a rate of about 3 – 4 Hz on a machine with 2.1 GHz processor and 3 Gb memory.

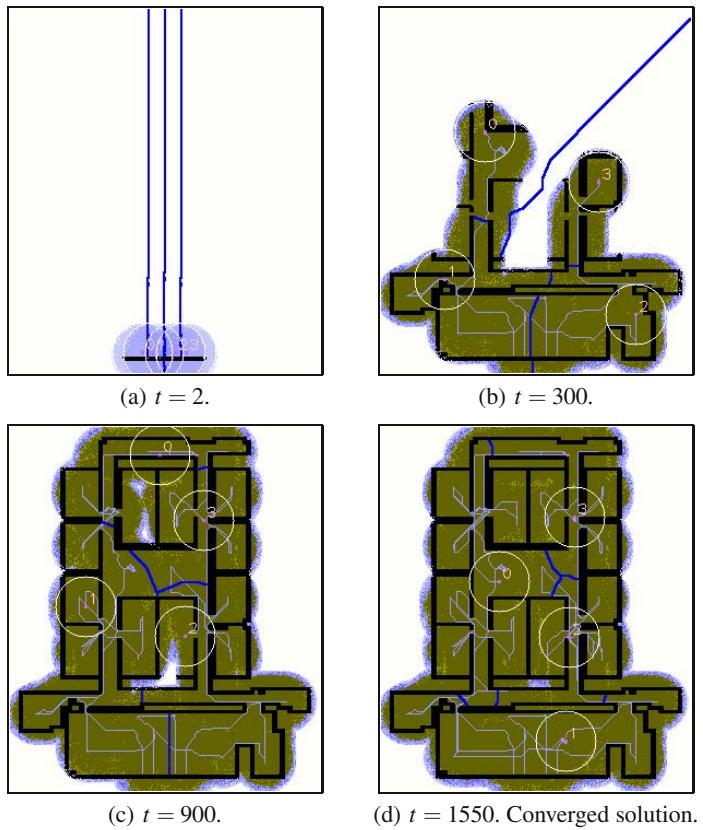


Fig. 9 Exploration and coverage of an office environment by a team of 4 robots. Blue curves indicate boundaries of tessellations, intensity of white indicates the value of entropy.

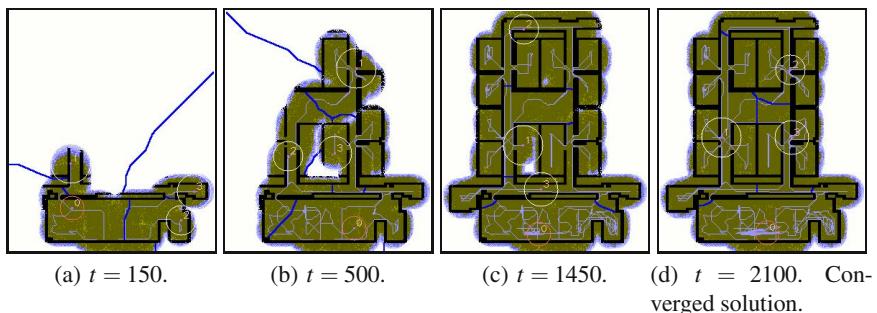


Fig. 10 Exploration and coverage of an office environment (284×333 discretized) by a team of 4 robots, with one of the robots (marked by red circle) being controlled by a human user

6 Conclusion

In this paper we have extended the coverage control algorithm proposed by *Cortes, et al.* [5], to non Euclidean configuration spaces that are, in general, non convex and equipped with metrics that are not Euclidean. The key idea is the transformation of the problem of computing gradients of distance functions to one of computing tangents to geodesics. We have shown that this simplification allows us to implement our coverage control algorithm in any space after reducing it to a discrete graph. We have illustrated the algorithm by considering multiple robots achieving uniform coverage on a 2-sphere and an indoor environment with walls and obstacles. We have also shown the application of the basic ideas to the problem of multi-robot cooperative exploration of unknown or partially known environments.

References

1. Bhattacharya, S., Ghrist, R., Kumar, V.: Relationship between gradient of distance functions and tangents to geodesics. Technical report, University of Pennsylvania, <http://subhrajit.net/wiki/index.php?SFile=DistanceGradient>
2. Bhattacharya, S., Michael, N., Kumar, V.: Distributed Coverage and Exploration in Unknown Non-convex Environments. In: Martinoli, A., Mondada, F., Correll, N., Mermoud, G., Egerstedt, M., Hsieh, M.A., Parker, L.E., Støy, K. (eds.) *Distributed Autonomous Robotic Systems*. STAR, vol. 83, pp. 61–75. Springer, Heidelberg (2013)
3. Bullo, F., Cortés, J., Martínez, S.: *Distributed Control of Robotic Networks: A Mathematical Approach to Motion Coordination Algorithms*. Applied Mathematics Series. Princeton University Press (2009)
4. Cormen, T.H., Stein, C., Rivest, R.L., Leiserson, C.E.: *Introduction to Algorithms*. McGraw-Hill Higher Education (2001)
5. Cortes, J., Martinez, S., Karatas, T., Bullo, F.: Coverage control for mobile sensing networks. *IEEE Trans. Robot. Autom.* 20(2), 243–255 (2004)
6. Cortez, J., Martinez, S., Bullo, F.: Spatially-distributed coverage optimization and control with limited-range interactions. *ESIAM: Control, Optimisation and Calculus of Variations* 11, 691–719 (2005)
7. Cortez, J., Martinez, S., Karatas, T., Bullo, F.: Coverage control for mobile sensing networks. *IEEE Trans. Robot. and Automat.* 20(2), 243–255 (2004)
8. Cundy, H., Rollett, A.: *Mathematical Models*, 3rd edn. Tarquin Pub. (1989)
9. Dijkstra, E.W.: A note on two problems in connexion with graphs. *Numerische Mathematik* 1, 269–271 (1959)
10. Durham, J.W., Carli, R., Frasca, P., Bullo, F.: Discrete partitioning and coverage control for gossiping robots. *IEEE Transactions on Robotics* 28(2), 364–378 (2012)
11. Gromov, M., Lafontaine, J., Pansu, P.: *Metric structures for Riemannian and non-Riemannian spaces*. Progress in Mathematics. Birkhäuser (1999)
12. Lloyd, S.P.: Least squares quantization in PCM. *IEEE Trans. Inf. Theory* 28, 129–137 (1982)
13. Pimenta, L.C.A., Kumar, V., Mesquita, R.C., Pereira, G.A.S.: Sensing and coverage for a network of heterogeneous robots. In: Proc. of the IEEE Conf. on Decision and Control, Cancun, Mexico, pp. 3947–3952 (December 2008)

14. Stachniss, C.: Exploration and Mapping with Mobile Robots. PhD thesis, University of Freiburg, Freiburg, Germany (April 2006)
15. Stachniss, C., Grisetti, G., Burgard, W.: Information gain-based exploration using rao-blackwellized particle filters. In: Proc. of Robot.: Sci. and Syst., Cambridge, MA, pp. 65–72 (June 2005)
16. Thrun, S., Burgard, W., Fox, D.: Probabilistic Robotics (Intelligent Robotics and Autonomous Agents). The MIT Press (2005)

Lion and Man with Visibility in Monotone Polygons^{*}

Narges Noori and Volkan Isler

Abstract. In the original version of the lion and man game, a lion tries to capture a man who is trying to escape in a circular arena. The players have equal speeds. They can observe each other at all times. We study a new variant of the game in which the lion has only line-of-sight visibility. Hence, it can observe the man's position only if the line segment connecting them does not intersect the boundary. We show that despite this limitation the lion can capture the man in any monotone polygon in finite time.

1 Introduction

In recent years, solving pursuit-evasion games in complex environments has received increasing attention. Such games model robotics applications such as surveillance and search-and-rescue. An overview of recent results on pursuit-evasion in robotics can be found in [3].

A fundamental pursuit-evasion game with immediate applications in robotics is the lion and man game. In this game, a lion (the pursuer) tries to capture a man (the evader) by moving onto the man's location (or getting close to it). There are some known cases where the pursuer wins the game. For example, in the original setting which takes place in a circular arena, the pursuer has a winning strategy [9] [11] when the players take turns in moving. Also, in the continuous time setting, i.e. when the players move simultaneously, the pursuer can get arbitrarily close to the evader [1]. In the turn-based game, if the lion can see the man at all times, a single pursuer can capture the man in any simply-connected polygon [7] and three lions suffice in arbitrary polygons with holes [2, 8].

Narges Noori · Volkan Isler
Department of CS&E, University of Minnesota
e-mail: {noori, isler}@cs.umn.edu

* This work was supported in part by NSF Awards #0916209, and #0917676.

In this paper, we focus on a variant of the turn-based lion-and-man game in which the pursuer has only line of sight vision. That is, the pursuer can see the evader only if the line segment connecting them is free of obstacles. This variant models robotics applications where the pursuer is a robot equipped with a camera or a laser scanner.

With the limited vision power, the pursuer has to first detect the evader. This problem called the search problem where the pursuer's goal is to find the evader has been studied extensively [6]. The necessary and sufficient conditions for a simple polygon to be searchable by a pursuer with various degrees of visibility power is presented in [12].

We study the game in monotone polygons and present a pursuit strategy which successfully combines search and capture while guaranteeing that the evader will be captured in a finite number of steps. A simple polygon is called monotone with respect to a line l if for any line l' perpendicular to l the intersection of the polygon with l' is connected [4]. In this work, without loss of generality we consider x -monotone polygons.

In monotone polygons, merely searching for the evader is straightforward: the evader can be found for example by moving along the shortest path that connects the leftmost vertex to the rightmost vertex. This is because every point inside the polygon is visible from a location on this path and the evader cannot move into a “cleared” region without revealing itself. Similarly, the capture strategy is simple when the pursuer knows the location of the evader at all times: it can capture the evader by starting from the leftmost vertex O_L and performing *the lion's move* in which it moves toward the evader along the shortest path that connects O_L to the evader's current location. The distance of the pursuer from O_L provides a natural notion of “progress” which is monotonically increasing¹. What is not obvious is whether such progress can be maintained when the pursuer has to search for the evader when it disappears. If the pursuer ends up retreating after a search, the evader might be able to design a strategy in which the pursuer oscillates between search and progress and the game can last forever.

We show that the pursuer can successfully combine search and making progress toward capture in monotone polygons. We are not aware of any other results which combine these two objectives for a single pursuer while providing guarantees about the outcome of the game. We present a deterministic pursuer strategy which includes a guarding phase that must be executed after a search phase. In this case, the lion's progress is not monotone. However, we show that the pursuer can push the evader further to the right after a finite number of steps. An interesting aspect of deterministic pursuit strategies, which makes them harder to design, is that the evader can simulate the pursuer's strategy which is fixed in advance. Therefore, we can analyze the worst case scenario for the pursuer: that the evader knows the pursuer's location at all times.

Our results provide a step toward understanding the lion and man game with visibility constraints in general polygonal environments. An interesting question is

¹ This argument works in any simply connected polygon [7].

whether a single pursuer suffices in a specific class of polygons². We show that monotone polygons are included in this class.

In this paper, we present the main ideas and an overview of the correctness proof and leave the details to our technical report [10] due to space limitations. We start with some preliminary definitions used throughout the text.

2 Preliminaries

Without loss of generality, we assume that the game takes place in an x -monotone polygon P . The leftmost vertex and the rightmost vertex are denoted by O_L and O_R respectively. The boundary of the polygon connects these vertices by two x -monotone chains denoted by $Chain_U$ and $Chain_L$, see Fig. 1-left.

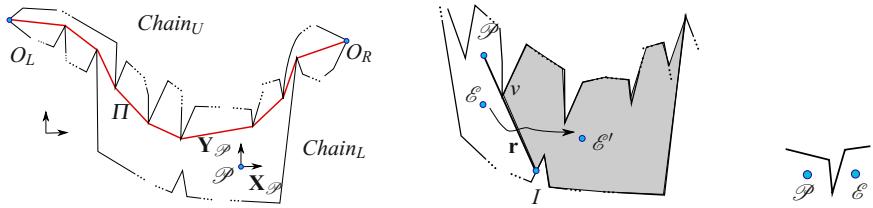


Fig. 1 **Left:** An x -monotone polygon. **Middle:** The pocket $pocket(v, r)$ is the shaded sub-polygon. **Right:** Since \mathcal{E} is not visible, capture condition is not satisfied.

We refer to the pursuer's and the evader's location at time-step t as $\mathcal{P}(t)$ and $\mathcal{E}(t)$ respectively. When the time is obvious from the context, we use \mathcal{P} and \mathcal{E} . Our *Game Model* is as follows: (1) The players move alternately in turns. (2) Each turn takes a unit time step. (3) In each turn the players can move along a line segment of length at most one to a point visible to themselves. (4) The evader has global visibility, but the pursuer sees the evader only if the line segment joining the two is not blocked by the boundary of the polygon. Note that since the pursuer has a deterministic strategy, the evader can simulate the pursuer's moves and hence it knows the location of \mathcal{P} at all times. (5) The pursuer captures \mathcal{E} if at any time, the distance between them is less than or equal to one (the step-size) while \mathcal{P} can see \mathcal{E} , see Fig. 1-right.

We refer to the segment between points u and v as uv . Whenever direction is also important we refer to the ray pointing from u to v as \mathbf{uv} . We define a local reference frame whose origin coincides with \mathcal{P} . Its axes $\mathbf{X}_{\mathcal{P}}$ and $\mathbf{Y}_{\mathcal{P}}$ are parallel to the axes of the reference frame, see Fig. 1-left. We refer to the boundary of P as ∂P , and the number of vertices in P as n . The shortest path between the two points u and v is denoted by $\pi(u, v)$, and the length of $\pi(u, v)$ is denoted by $d(u, v)$. The diameter of

² It should also be noted that capture using only deterministic strategies remains an open question.

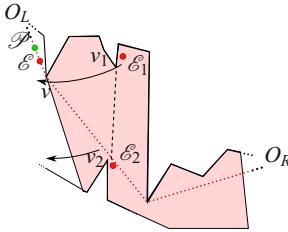


Fig. 2 A difficult situation for \mathcal{P} : \mathcal{E} can recontaminate the cleared region depending on how \mathcal{P} enters the pocket

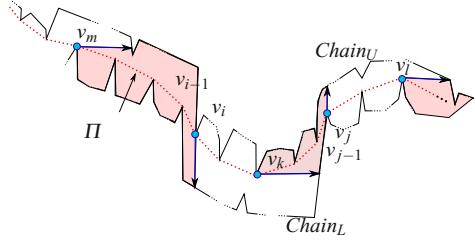


Fig. 3 The path Π is shown in red dots. The critical sub-polygon defined by v_m and v_i is type 4, v_i and v_k is type 1, v_k and v_j is type 2, and v_j and v_l is type 3.

the polygon is $D = \max_{u,v \in P} d(u,v)$. The shortest path tree rooted at the point o in P is defined as $\cup_{v \in V} \pi(o,v)$ where V denotes vertices of P . For a point p inside the polygon, the parent of p , denoted by $\text{parent}(p)$, is the first vertex on the shortest path $\pi(o,p)$ from p to o . In the rest of the paper, we consider the shortest path tree rooted at $o = O_L$. We denote $\pi(O_L, O_R)$ by Π (Fig. 1-left). For simplicity we denote $d(O_L, p)$ by $R(p)$ for a point $p \in P$.

Suppose that it is the evader's turn to move. See Fig. 1-middle. Imagine that the pursuer and the evader can see each other before the evader's move but the evader disappears behind a vertex v after moving to \mathcal{E}' . Let \mathbf{r} be a ray originating from \mathcal{P} and passing through v . Let I be the intersection of this ray with the polygon. The sub-polygon which contains \mathcal{E} and is bounded by the ray \mathbf{r} plus the boundary of the polygon from v to I is called a *pocket* [7]. The ray \mathbf{r} is called the entrance of the pocket and the pocket is referred to as $\text{pocket}(v, \mathbf{r})$.

In our pursuit strategy, we extensively use the *lion's move* strategy [9] [11] proposed for capturing \mathcal{E} in circular arena. This strategy is performed with respect to a center C . Initially, the pursuer has to lie on the line segment connecting C to \mathcal{E} in order to be able to perform the lion's move strategy. In each turn, the pursuer moves so that it remains on the line segment connecting C and \mathcal{E} .

The lion's move is generalized to the *extended lion's move* [7] for capturing the evader in simply-connected polygons (when the pursuer can see the evader). Suppose that \mathcal{E} is visible to \mathcal{P} and \mathcal{P} is on the edge connecting $\text{parent}(\mathcal{E})$ and \mathcal{E} in the shortest path tree rooted at O_L . Then the pursuer can follow \mathcal{E} by lion's move with respect to $\text{parent}(\mathcal{E})$. Note that as \mathcal{E} is moving, $\text{parent}(\mathcal{E})$ is changing.

3 Monotone Polygon Capture Strategy

Designing a pursuer strategy which guarantees capture in addition to finding the evader is not a trivial problem (unlike the search problem alone as we mentioned earlier). Fig. 2 provides some intuition. Suppose that \mathcal{P} is following \mathcal{E} by lion's move with respect to O_L . Also suppose that \mathcal{E} disappears behind v in the shaded

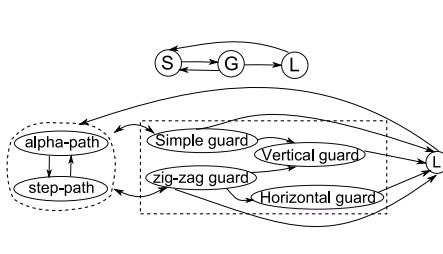


Fig. 4 The state diagram for the MPC strategy. The sub-states in S and G are shown at the bottom.

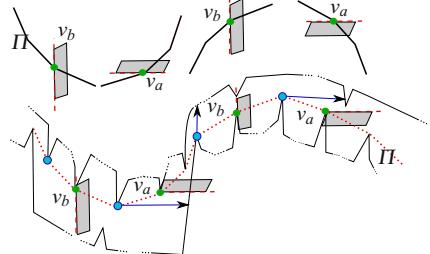


Fig. 5 The path Π inside each of the critical sub-polygons, and the half-planes in Property 1. From top-left to top-right are types 1 to 4.

pocket. A careless search strategy may result in the pursuer's losing its progress: If the pursuer first visits v_2 , then the evader hidden at v_1 will escape to v and re-enter the previously cleared region. Likewise, if the pursuer first visits v_1 , the evader hidden at v_2 can escape to the cleared region. However, we will show that the pursuer can execute a guarding strategy and recover its progress.

Before presenting our pursuit strategy in detail, let us introduce the notion of *critical sub-polygons*, which we use to partition the monotone polygon P . These sub-polygons allow us to enumerate all possible configurations between \mathcal{P} , \mathcal{E} and the structure of P since our proposed strategy is symmetric in different types of these critical sub-polygons.

Definition 1 (Critical Sub-polygons). Let Π be the shortest path from O_L to O_R and denote the vertices on Π by $\{v_0, \dots, v_i, \dots\}$. Let θ_{i-1} be the slope of the edge $v_{i-1}v_i$ and $\delta\theta_i = \theta_i - \theta_{i-1}$. Then, the *critical vertices* are those vertices on Π that either θ or $\delta\theta$ changes sign. For example, in Fig. 3, the vertices v_i , v_k and v_j are the critical points. For a critical vertex $p = v_i$, we assign \mathbf{Y}_p if in $p = v_i$, the values θ_i and $\delta\theta_i$ have different signs and \mathbf{X}_p if they have the same signs. Then, each two consecutive critical vertices, say v_i and v_k , define a *critical sub-polygon* given by the sub-polygon formed by ∂P and the rays assigned to v_i and v_k . See Fig. 3. Depending on the sign of θ and $\delta\theta$ in the critical sub-polygons, we get four types of critical sub-polygons: Type (1) when $\theta < 0$ and $0 < \delta\theta$, Type (2) when $0 < \theta$ and $0 < \delta\theta$, Type (3) when $0 < \theta$ and $\delta\theta < 0$, Type (4) when $\theta < 0$ and $\delta\theta < 0$.

Remark 1. Throughout the paper, we present the strategy for the 1st type critical sub-polygons. The strategy for the other types is symmetric.

We will present a pursuit strategy called *Monotone Polygon Capture (MPC)* strategy which guarantees capture. The state diagram of the MPC strategy is given in Fig. 4. It consists of three states: *Search*, *Guard* and *Extended Lion's Move* denoted by S , G , and L respectively. Initially, \mathcal{P} is at O_L . It starts by the S state if \mathcal{E} is invisible and the L state otherwise. The pursuer *searches* for \mathcal{E} when it is invisible. In the rest of the

paper, we refer to the pocket that the evader is hidden inside of as $pocket(v, \mathbf{r})$ (also the *contaminated region*) where v is the location of the pursuer at the time that the evader has disappeared and $\mathbf{r} = \mathbf{p}v$. See Fig. 1-middle. Without loss of generality we assume that at the beginning of the search state the pursuer is at v . This is because, after \mathcal{E} disappears, the pursuer can move toward v along the straight line pv until it reaches at v . If in the meantime \mathcal{E} appears, the pursuer continues the last state's strategy, i.e. G or L .

When \mathcal{E} is found as a result of the S strategy, the pursuer performs the *guard* strategy in order to establish the extended lion's move with respect to O_L . During the lion's move state or the guard state, \mathcal{E} might disappear. At this time \mathcal{P} switches to the search strategy. The sequence of state transitions is $((SG)^*L)^*$ and the loops $(SG)^*$ and $L(SG)^*L$ are possible.

To prove that our proposed strategy guarantees capture, it is necessary to show that these loops terminate after finite time. To do so, we define a *reference vertex*, denoted by p_{ref} , which is a vertex of the polygon (initially $p_{ref} = O_L$). We show that \mathcal{P} maintains the following invariants and then we define the pursuer's notion of progress based on pushing p_{ref} to the right. We refer to the S state and its following G state as the *combined Search/Guard* state i.e. (SG) . At the beginning of a combined search/guard state (SG) or an L state we have:

- **Invariant (I1)** $R(p_{ref}) \leq R(\mathcal{P})$ and $R(p_{ref}) < R(\mathcal{E})$.
- **Invariant (I2)** whenever \mathcal{E} is invisible, it is inside $pocket(v, \mathbf{r})$ where v is always the leftmost vertex of $pocket(v, \mathbf{r})$.

The pursuer maintains the aforementioned invariants by exploiting properties of the monotone polygons. After providing an important property of P , we explain how \mathcal{P} makes progress.

Definition 2 (The half-plane of a vertex). For a vertex $v \in P$, the *open half-plane*, denoted by $hf(v)$, is defined as the half-plane to the right of v if v is inside the 1st or the 3rd type critical sub-polygons. For the 2nd type, $hf(v)$ is the half-plane above v (including the points p with $x(v) < x(p)$). Finally for the 4th type, $hf(v)$ is the half-plane below v (including the points p with $x(v) < x(p)$). The corresponding closed half-planes are denoted by $chf(v)$. See Fig. 5.

The following property of monotone polygons is crucial in this paper [10].

Property 1. Consider the critical sub-polygons defined above. See Fig. 5. First consider those types that θ and $\delta\theta$ have different signs i.e. the 1st and the 3rd types, and let v_b be a vertex in them so that:

- For the 1st type: The vertex v_b is a vertex from the upper chain where the slope of the edge between $parent(v_b)$ and v_b is negative.
- For the 3rd type: The vertex v_b is a vertex from the lower chain where the slope of the edge between $parent(v_b)$ and v_b is positive.

Then for all points $p \in hf(v_b)$, we have $R(v_b) < R(p)$. The similar result is valid for the 2nd and the 4th types as follows. Let v_a be a vertex from the upper chain

(lower chain respectively) where the slope of the edge between parent(v_a) and v_a is positive (negative respectively). Then for all $p \in hf(v_a)$ we have $R(v_a) < R(p)$.

The pursuer achieves one of the following notions of *progress* after a finite number of time-steps. Consider two consecutive combined (SG) states, and let t and t' be their corresponding starting time-steps. Suppose that p_{ref} and p'_{ref} are the old and the new reference vertices at t and t' respectively. Also let v and v' be the old and new vertices defining the pockets $pocket(v, \mathbf{r})$ and $pocket(v', \mathbf{r}')$ respectively. Then:

- **Progress (P1)** either the pursuer updates p_{ref} to a new vertex p'_{ref} so that $R(p_{ref}) < R(p'_{ref})$.
- **Progress (P2)** or p_{ref} remains the same and the pursuer updates the contaminated region $pocket(v, \mathbf{r})$ to $pocket(v', \mathbf{r}')$ so that $v' \in hf(v)$ where $hf(v)$ is the open half-plane assigned to the vertex v as defined in Definition 2.

Our main result is the following theorem:

Theorem 2. (Progress) Suppose that P is a monotone polygon. Then \mathcal{P} by following the MPC strategy can capture \mathcal{E} in $O(n^7 \times D^{13})$ steps where n is the number of vertices of P and D is the diameter of P .

Before proving Theorem 2, we present the details of each state. The proof of this theorem is provided in Sect. 5.3.

4 Search State

The pursuer performs the search strategy in order to find \mathcal{E} . As we mentioned in Section 3, when \mathcal{E} disappears, \mathcal{P} walks toward the blocking vertex v . Hence without loss of generality we can assume that at the beginning of the S state \mathcal{P} is at v .

In order to find \mathcal{E} , the pursuer moves along a path which we refer to as the *search path* defined below. As stated below in Observation 3, as \mathcal{P} is moving along the search path, it will eventually find \mathcal{E} . At this time, the pursuer starts the guard state. Note that Observation 3 also implies that $x(\mathcal{P}) < x(\mathcal{E})$ when \mathcal{E} is found.

Observation 3. Let p_1 and p_2 be two points inside P where $x(p_1) < x(p_2)$ and suppose that $x(p_1) < x(\mathcal{E})$. Suppose that \mathcal{P} moves from p_1 to p_2 along any arbitrary path. Then if the pursuer reaches p_2 and the evader is still invisible, it must be that $x(p_2) < x(\mathcal{E})$. Otherwise at the time that \mathcal{E} is found it must be that $x(\mathcal{P}) < x(\mathcal{E})$.

The search path consists of two types of paths, the α -path and the *step-path* which we will define shortly. See also Fig. 4. In the following, we first define the search path for the first type critical sub-polygon.

We define the α *lines* as follows. The lines that make angle $-\alpha$ (or α for other types as we will see) with the x -axis are called the α -*lines*. As we will see in Sect. 5.3, the angle α is used to bound the time spent in lion's move [10].

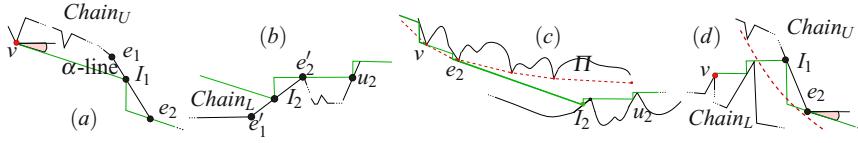


Fig. 6 The search path is shown in green. The path Π is shown in red dashed line. (a) the α -path and the α -step e.g. the portion of the search path from v to e_2 is one α -step. (b) The step-path e.g. from I_2 to u_2 is one step. Note that I_2 is the floor point. (c) the search path consists of the step-path and the α -path. (d) the search path when $v \in Chain_L$.

Definition 3. The angle α is chosen as the minimum of the two angles $\psi_1 = \arcsin \frac{1}{D}$ and $\psi_2 = (\frac{\pi}{2} - 2 \times \arctan 0.5)$.

If $v \in Chain_U$ (which is in the 1st type critical sub-polygon), the search path starts by the α -path, otherwise it starts by the step-path as described below (see Fig. 6).

Suppose that $v \in Chain_U$. The search path starts by the α line passing through v . Let $e = e_1e_2$ be the edge on ∂P that this α line intersects and also let I_1 be the point of intersection. The edge e can be either on $Chain_U$ or $Chain_L$. Suppose that $e \in Chain_U$. The search path continues with the vertical line passing through I_1 until this line intersects the α line passing through e_2 , and then continues along this α line. We refer to this part of the search path as the *α -path*. See Fig. 6-(a). We can divide the α -path into a number of *α -steps* e.g. the portion of the search path from v to e_2 is an α -step.

The search path is defined like this until the intersection edge is on $Chain_L$. Now, consider the α line that intersects the lower chain and call its intersection point with ∂P the *floor* point and its first endpoint as the *ceiling* point (e.g. in Fig. 6-(c) I_2 and e_2 respectively). Let $e' = e'_1e'_2$ be the corresponding edge on $Chain_L$ and I_2 be the floor point. At this point (the floor point I_2), the search path continues along the vertical line passing through I_2 until this vertical line reaches at the horizontal line passing through e'_2 and then continues along this horizontal line. This portion of the search path is referred to as the *step-path*. See Fig. 6-(b). We also can divide the step-path into a number of *steps* e.g. from I_2 to u_2 is one step.

Now suppose that $v \in Chain_L$. Then the search path starts by the step-path and then as it hits the upper chain it continues along the α -path e.g. in Fig. 6-(d) from v to I_1 is the step-path and after e_2 we have the α -path.

5 Guard State

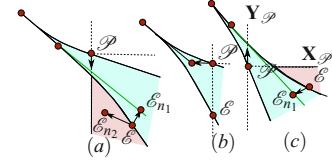
As we mentioned earlier, after \mathcal{P} finds \mathcal{E} , it starts the *guard state* in order to establish the extended lion's move state. The purpose of the guard strategy is to establish the extended lion's move by catching up to $\pi(O_L, \mathcal{E})$ while preserving progress.

Recall that according to Observation 3, at the beginning of the guard state, we have $x(\mathcal{P}) < x(\mathcal{E})$. At this time the pursuer starts either the zig-zag guard strategy

Algorithm 1. Zig-Zag Guard

```

1:  $l_v \leftarrow \mathbf{Y}_{p_{ref}}$ 
2: if  $x(\mathcal{P}) < x(\mathcal{E})$  then
3:   if  $\mathcal{P}$  is below  $\pi(O_L, \mathcal{E})$  then move parallel to the
      positive y-axis.
4:   if  $\mathcal{P}$  is above  $\pi(O_L, \mathcal{E})$  then move parallel to the
      negative y-axis.
5: else
6:   move parallel to the negative x-axis.
7: end if
```

**Fig. 7** The zig-zag guard strategy

or the *simple guard* strategy based on the type of the critical sub-polygon and the quadrant that \mathcal{E} has appeared as described below. See Fig. 4.

1. if \mathcal{P} is inside the 2nd and the 4th types: \mathcal{P} always starts by zig-zag guard. During zig-zag guard, the pursuer switches to the horizontal guard sub-state if it retreats beyond v as a result of the zig-zag strategy.
2. if \mathcal{P} is inside the 1st type (3rd): the pursuer starts by zig-zag guard if \mathcal{E} has appeared inside the fourth quadrant of \mathcal{P} (first quadrant respectively), otherwise if \mathcal{E} has appeared inside the first quadrant (fourth quadrant respectively) it starts by simple guard. Here during zig-zag guard or simple guard, the pursuer might retreat beyond v in which case it starts the vertical guard strategy.

5.1 Zig-Zag Guard

Suppose that \mathcal{P} is inside the 1st type critical sub-polygon and \mathcal{E} has appeared in the fourth quadrant of \mathcal{P} . The *Zig-Zag Guard* strategy is shown in Algorithm 1.

The idea is composed of two parts. The first one is to catch up to $\pi(O_L, \mathcal{E})$ in order to be able to follow the evader with extended lion's move. This is done by moving parallel to the x -axis or the y -axis, see Fig. 7 and Fig. 8-(a). However, the pursuer might retreat beyond v as shown in Fig. 8-(a). The second idea deals with this retreat by performing the *vertical guard* strategy presented in section 5.3. The following lemma presents the pursuer's progress in zig-zag guard in the first type critical sub-polygon.

Lemma 1 (Zig-zag guard progress). *Let $pocket(v, \mathbf{r})$ be the pocket being searched in the previous S state (v is inside the 1st type). Then the state following the zig-zag guard strategy can be: (i) the S state while Progress (P1) or (P2) is guaranteed, (ii) the L state while $R(p_{ref}) \leq R(\mathcal{P}) < R(\mathcal{E})$, (iii) the vertical guard sub-state while $y(\mathcal{E}) < y(\mathcal{P}) \leq y(v)$, $x(\mathcal{P}) = x(\mathcal{E}) = x(v)$ and the evader is crossing $l_v = \mathbf{Y}_v$ to the left (this can happen only when $v \in Chain_U$).*

In the case that v is in the 2nd type, the only difference is that when $x(v) \leq x(\mathcal{P}) < x(\mathcal{E})$ and $y(\mathcal{P}) = y(\mathcal{E}) = y(v)$, the pursuer performs the horizontal guard strategy.

The 3rd and the 4th types are symmetric to these two types (the 1st and the 2nd types) respectively.

Proof. Let \mathcal{P}_0 and \mathcal{E}_0 be the positions of \mathcal{P} and \mathcal{E} at the beginning of the zig-zag guard. Observe that $\text{parent}(\mathcal{P}_0)$ is in the second quadrant of \mathcal{P}_0 [10]. Now consider the funnel [5] formed by $\pi(O_L, \mathcal{E})$ and $\pi(O_L, \mathcal{P})$. Let d be the deepest common vertex between these two paths. Then the shortest path to all points inside this funnel starts by $\pi(O_L, d)$ and then continues inside the funnel. Observe that as \mathcal{E} moves, $\pi(O_L, \mathcal{E})$ changes continuously, i.e. $\pi(O_L, \mathcal{E})$ is not jumping over \mathcal{P} .

Suppose that \mathcal{P} is below $\pi(O_L, \mathcal{E})$ (Fig. 7-(c)). Then \mathcal{P} is getting closer to $\pi(O_L, \mathcal{E})$ just by moving upward parallel to the y -axis. The pursuer continues this strategy until one of the following happens: (i) the S state on $\text{pocket}(v', \mathbf{r}')$ where $v' \in hf(v)$ or (ii) the L state in $hf(v)$. Similarly, if the pursuer is above $\pi(O_L, \mathcal{E})$ it moves downward and then to the left (Fig. 7-(a) and (b), also Fig. 8-(a)). This continues until (i) the S state on $\text{pocket}(v', \mathbf{r}')$ where $v' \in hf(v)$, (ii) the L state in $hf(v)$, or (iii) the vertical guard while the players cross \mathbf{Y}_v (note that this is possible only when $v \in \text{Chain}_U$ in which case $p_{\text{ref}} = v$).

If $v' \in \text{Chain}_L$ we do not update p_{ref} but we have $v' \in hf(v)$ (Progress (P2)). Otherwise if $v' \in \text{Chain}_U$ we set p'_{ref} to v' . Since $p'_{\text{ref}} = v' \in hf(v) \subseteq hf(p_{\text{ref}})$ we have $R(p_{\text{ref}}) < R(p'_{\text{ref}})$ (Property 1) and hence we have Progress (P1). Note that since $hf(v) \subseteq hf(p_{\text{ref}})$, in case of L state we would have $R(p_{\text{ref}}) \leq R(\mathcal{P}) < R(\mathcal{E})$. \square

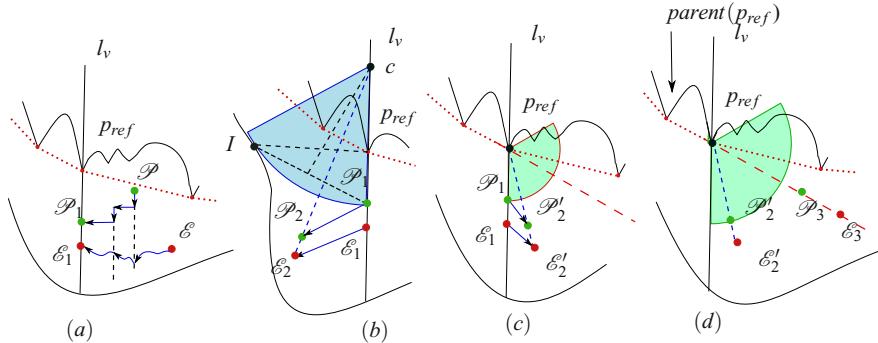


Fig. 8 The zig-zag guard followed by the vertical guard . The path Π is shown in red dots.

5.2 Simple Guard

Suppose that \mathcal{P} is the first type critical sub-polygon and \mathcal{E} has appeared in the first quadrant of \mathcal{P} . At this time \mathcal{P} performs the *Simple Guard* strategy presented in the state diagram shown in Fig. 9.

Recall that $\text{pocket}(v, \mathbf{r})$ is the pocket which has been searched in the previous S state, and $v \in chf(p_{\text{ref}})$. We define the auxiliary reference point $p_{\text{ref}, \text{aux}}$ so that

$p_{ref_aux} \in chf(p_{ref})$. To do so suppose that v_{aux} is the starting endpoint of the α -path (if $v \in Chain_U$, then $v_{aux} = v$). Moreover, let $ceil$ be the ceiling point if it exists (refer to Sect. 4), and p_0 be the location of the pursuer at the beginning of the simple guard (end of the previous S state).

- if p_0 is in the portion of the search path, from v to v_{aux} : note that only when $v \in Chain_L$, we have $v_{aux} \neq v$ e.g. in Fig. 6-(d) we have $v_{aux} = e_2$. Here p_{ref_aux} is the bottommost vertex from the upper chain which is in the region in $chf(p_{ref})$ and to the left of the ray p_0p_{ref} . See Fig. 10-(a).
- If p_0 is in the portion of the search path from v_{aux} to the floor point: then p_{ref_aux} is the starting endpoint of the corresponding α -step. For example, in Fig. 6-(c), if p_0 is on the α -step defined from e_2 to I_2 then $p_{ref_aux} = e_2$.
- If p_0 is in the portion of the search path after the floor point: let a be the intersection point between the α line passing through p_0 and Π . Suppose that w_1w_2 and $w'_1w'_2$ are the edges on Π such that $x(w_1) \leq x(ceil) < x(w_2)$ and $x(w'_1) \leq x(a) < x(w'_2)$ respectively. Then, if a is inside the next critical sub-polygon, p_{ref_aux} is the second critical endpoint that defines the current critical sub-polygon. If $w_1 \neq w'_1$, i.e. a and $ceil$ are not in between the endpoints of the same edge on Π , then $p_{ref_aux} = w'_1$, see Fig. 10-(b). Otherwise, p_{ref_aux} is the bottommost vertex from the upper chain which is inside $chf(ceil)$ and to the left of the line connecting $ceil$ to p_0 , see Fig. 10-(c).

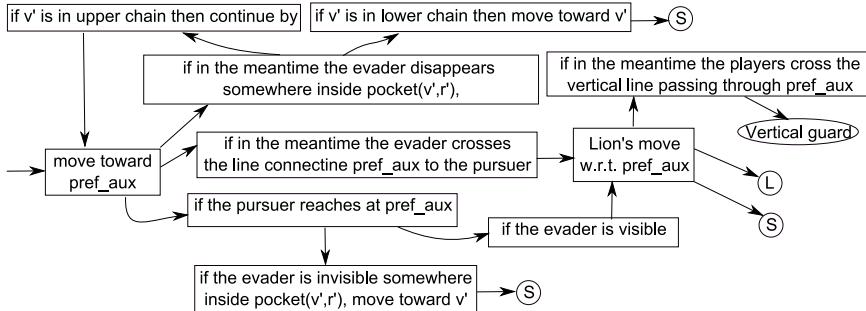


Fig. 9 The simple guard strategy in the 1st type. Note that in the 3rd type, the chains are swapped (lower to upper and vice versa).

The simple guard strategy is the following (Fig. 9). The pursuer moves back toward $Pref_aux$ along the line segment connecting $Pref_aux$ to p_0 in order to establish the lion's move with respect to $Pref_aux$, Fig. 11-(b). Note that $Pref_aux$ is visible to p_0 . In [10], we show that the angle between p_0Pref_aux and the x -axis is smaller than α . We then prove that the evader cannot cross the segment p_0Pref_aux without being captured. Therefore \mathcal{P} would reach $Pref_aux$ sooner than \mathcal{E} . This part of the strategy would result in lion's move with respect to $Pref_aux$ or the S state inside $hf(Pref_aux)$ (which is the right half-plane of $Pref_aux$ since we are in the first type).

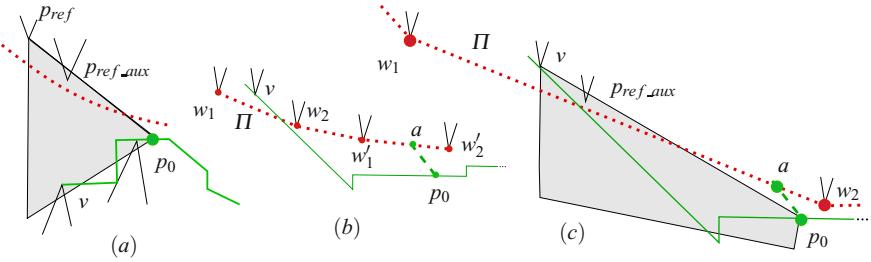


Fig. 10 The path Π is shown in red dots. Note that all upper chain vertices are above Π . (a) here p_{ref_aux} is the bottommost vertex from $Chain_U$ in the shaded region. (b) here $p_{ref_aux} = w'_1$ ($ceil = v$). (c) p_{ref_aux} is the bottommost vertex from $Chain_U$ in the shaded region ($ceil = v$).

The lion's move with respect to p_{ref_aux} itself can result in S state or the L state inside $hf(p_{ref_aux})$, or retreat beyond p_{ref_aux} (Fig. 11-(c)) in which case, similar to the zig-zag guard strategy, the pursuer performs the vertical guard strategy in order to recover his progress. See Sect. 5.3.

Lemma 2 (Simple guard progress). *Let $pocket(v, \mathbf{r})$ be the pocket being searched in the previous S state. Then the state following the simple guard strategy can be:* (i) *the L state while Progress (P1) or (P2) is guaranteed, (ii) the S state while $R(p_{ref}) \leq R(\mathcal{P}) < R(\mathcal{E})$, (iii) the vertical guard sub-state while $y(\mathcal{E}) < y(\mathcal{P}) \leq y(p_{ref_aux})$, $x(\mathcal{P}) = x(\mathcal{E}) = x(p_{ref_aux})$ and the evader is crossing $l_v = \mathbf{Y}_{p_{ref_aux}}$ to the left.*

Proof. From the description above, the simple guard is finished to (i) the L state inside $hf(p_{ref_aux})$, (ii) the S state inside $hf(p_{ref_aux})$, or (iii) vertical guard while the player are crossing l_v to the left.

In case of the L state, since $p_{ref_aux} \in chf(p_{ref})$ we would have $R(p_{ref}) \leq R(p_{ref_aux}) \leq R(\mathcal{P}) < R(\mathcal{E})$.

In case of the S state, let $pocket(v', \mathbf{r}')$ be the new pocket. Then $v' \in hf(p_{ref_aux}) \subseteq hf(p_{ref})$ (note that p_{ref_aux} can be the same as p_{ref}).

If $v' \in Chain_U$ we set p'_{ref} to v' . Since $p'_{ref} = v' \in hf(p_{ref})$ we have $R(p_{ref}) < R(p'_{ref})$ (Property 1) and hence we have Progress (P1). If $v' \in Chain_L$ we update p_{ref} to p_{ref_aux} . Since we have $v' \in hf(v)$ we will definitely have Progress (P2) [10]. \square

5.3 Vertical Guard and Horizontal Guard

Suppose that \mathcal{P} is the first type critical sub-polygon and it has invoked the vertical guard sub-state. The pursuer performs the vertical guard strategy when both players are on the line l_v where $l_v = \mathbf{Y}_{p_{ref_aux}}$ (from simple guard Lemma 2) or $l_v = \mathbf{Y}_v$ (from zig-zag guard Lemma 1). In the following, we denote the vertex that defines l_v by c_{aux} , i.e. $c_{aux} = p_{ref_aux}$ if $l_v = \mathbf{Y}_{p_{ref_aux}}$ and $c_{aux} = v$ if $l_v = \mathbf{Y}_v$. Recall that at

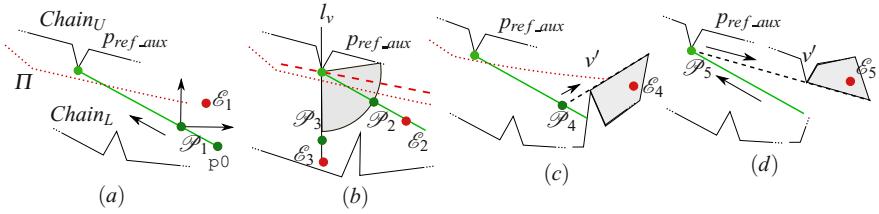


Fig. 11 The Simple Guard. (a) \mathcal{P} moves toward p_{ref_aux} . (b) as \mathcal{E} crosses the line between p_{ref_aux} and p_0 , \mathcal{P} performs lion's move w.r.t. p_{ref_aux} . (c) if \mathcal{E} hides inside pocket (v', \mathbf{r}') , \mathcal{P} moves toward v' if $v' \in Chain_L$ otherwise if $v' \in Chain_U$ it continues moving toward p_{ref_aux} and then from there it moves toward v' (d) (note that here, as the pursuer keeps moving back to p_{ref_aux} , the new pocket formed by the ray connecting \mathcal{P} to v' , includes the initial hiding pocket $pocket(v', \mathbf{r}')$).

the beginning of vertical guard, \mathcal{E} is crossing l_v to the left i.e. at this time $y(\mathcal{E}) < y(\mathcal{P}) \leq y(c_{aux})$, $x(\mathcal{P}) = x(\mathcal{E}) = x(c_{aux})$. The goal of the vertical guard strategy is to recover the progress that \mathcal{P} loses as a result of retreat beyond p_{ref} .

The vertical guard strategy is composed of two parts: lion's move with respect to a center c and lion's move with respect to c_{aux} . The center c is defined on the line l_v . The pursuer uses c as the center for the lion's move if \mathcal{E} crosses l_v to the left, and c_{aux} if \mathcal{E} crosses l_v to the right. The role of the circle centered at c is to push \mathcal{E} to the right of c_{aux} (inside $hf(c_{aux})$) and the role of the other circle centered at c_{aux} is to force \mathcal{E} to cross the ray connecting $parent(c_{aux})$ to c_{aux} and hence to establish the extended lion's move state. See Fig. 8.

The center c is defined so that the resulting lion's circle prevents \mathcal{E} from hiding behind the upper chain vertices which are before c_{aux} i.e. points $p \in Chain_U$ with $x(p) < x(c_{aux})$. In addition, the circle is defined such that if \mathcal{E} disappears into lower chain vertices which are before p_{ref} , the resulting pocket would be a *simple pocket* (see Appendix) and hence \mathcal{P} can repel \mathcal{E} from them by performing the pursuit strategy on the corresponding simple pocket (see Lemma 7 in Appendix).

The center c is defined as follows: Let I be the intersection between the horizontal line passing through c_{aux} and ∂P , see Fig. 12-(a). Then c is the intersection between bisector of \mathcal{PI} and the line l_v . Clearly the lion's circle centered at c is passing through I and \mathcal{P} . In [10] we prove that all upper chain vertices before c_{aux} are above $c_{aux}I$ and thus the lion's move with respect to c is feasible.

Here we emphasize that the lion's move with respect to c will result in progress in *finite* time if the initial radius of the corresponding circle is upper bounded [11]. In the following, we show that the angle α plays an important role in bounding the radius. Let r be the radius of the lion's circle centered at c i.e. $r = c\mathcal{P}$.

Lemma 3. *The initial radius (r) of the circle defined above is upper bounded by $r \leq \frac{2 \times l^2}{h_{min}}$ where h_{min} is a lower bound for $h = y(c_{aux}) - y(\mathcal{P})$ at the beginning of vertical guard ($h_{min} \leq h$).*

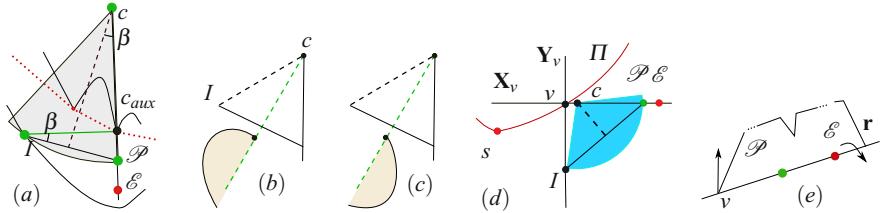


Fig. 12 (a) the vertical guard (Π is shown in red dots). (b) These pockets are impossible. (c) The possible pockets before l_v are simple pockets. (d) The horizontal guard. (e) the simple pockets.

Proof. Let β be the angle between $\mathcal{P}I$ and the horizontal line passing from c_{aux} , see Fig. 12-(a). Also let $2 \times l = \mathcal{P}I$. We have $\sin \beta = \frac{l}{r} = \frac{h}{2 \times l}$. Hence $r = \frac{2 \times l^2}{h}$. Therefore $r \leq \frac{2 \times l^2}{h_{min}}$. \square

In [10], we provide $h_{min} = \sin \alpha$ as the lower bound for h at the beginning of the vertical guard strategy.

Corollary 1. *The initial radius of the vertical guard circle centered at c is upper bounded by $r \leq \frac{D^2}{2 \cdot \sin \alpha}$. Referring to Definition 3, a rough bound for r is $O(D^3)$.*

Note that since all upper chain vertices before c_{aux} are above $c_{aux}I$ [10], the lion's circle centered at c prevents \mathcal{E} from escaping into the upper chain vertices which are not in $hf(c_{aux})$ [10].

Lemma 4. *If during vertical guard strategy \mathcal{E} disappears behind a vertex which is to the left of l_v , then the resulting pocket would be a simple pocket (see Appendix). Also refer to Fig. 12-(b) and (c).*

Lemma 5 (Vertical guard progress). *At the end of vertical guard the pursuer achieves Progress (P1) or (P2) [10].*

Lemma 6 (the time spent in guard state). *The pursuer finishes the guard state and switches to the next state in $O(n^4 \times D^{11})$ steps.*

Proof. Let T_1 be the time spent in simple guard, and T_2 be the time spent in vertical guard. Then the guard time is at most $T_1 \times T_2$.

The vertical guard strategy is composed of the simple pocket strategy (to the left of l_v) and the lion's move with respect to c_{aux} or c . In the worst case, every single step of the lion's move can be followed by the simple pocket strategy. Therefore, the total time in vertical guard would be the product of the time spent in extended lion's move and the simple pocket strategy. The time spent in simple pocket strategy is $O(n^2 \times D^3)$ (Lemma 7). The initial radius of the circle centered at c used during vertical guard is $O(D^3)$ (Corollary 1). Hence the lion's move with respect to c during vertical guard takes $O((n \times D^6))$ [7] [10]. Therefore, the vertical guard state takes $T_2 = O((n \times D^6) \times (n^2 \times D^3)) = O(n^3 \times D^9)$.

The simple guard strategy is composed of the lion's move w.r.t. $p_{ref,aux}$ which requires at most $T_1 = O(n \times D^2)$. Thus, the guard time is $O((n \times D^2) \times (n^3 \times D^9)) = O(n^4 \times D^{11})$. \square

Finally, we present the *horizontal guard* strategy. Suppose that the horizontal guard has been invoked in the 2nd type. Recall that at this time $x(v) \leq x(\mathcal{P}) < x(\mathcal{E})$ and $y(\mathcal{P}) = y(\mathcal{E}) = y(v)$ (Lemma 1). The center c for the horizontal guard is found as follows: Let I be the intersection between ∂P and \mathbf{Y}_v . Then c is the intersection point between bisector of $\mathcal{P}I$ and \mathbf{X}_v . See Fig. 12-(d). Symmetric to what we saw in vertical guard, \mathcal{P} performs lion's move with respect to c or v as \mathcal{E} moves below $l_h = \mathbf{X}_v$ or above $l_h = \mathbf{X}_v$. This continues until the next state is established in $hf(v)$.

We are now ready to present the proof of Theorem 2.

Proof (Theorem 2)

Suppose \mathcal{P} is currently in a combined (SG) state. In Lemma 1, Lemma 2, and Lemma 5, we have shown that after finite time this combined state will terminate to another combined (SG) state or an L state.

In the latter case, the aforementioned lemmas ensure that $R(p_{ref}) \leq R(\mathcal{P}) < R(\mathcal{E})$. Moreover \mathcal{P} is on $\pi(O_L, \mathcal{E})$ and $d(O_L, \mathcal{P})$ is increasing after each step of the L state [7]. Hence either \mathcal{P} captures \mathcal{E} in the L state or it switches to another (SG) state and there would be an infinite sequence of (SG) states.

Now consider two consecutive (SG) states and suppose that \mathcal{E} is not captured yet. According to the aforementioned lemmas, \mathcal{P} achieves progress (P1) or (P2). Since in (P2), v and v' are vertices of P , after at most n progress updates of type (P2), there would be one progress update of type (P1). Also since p_{ref} is a vertex, at some point $D \leq R(p_{ref})$. Recall that D is the diameter and thus $R(.) \leq D$. Thus, at some point $D = R(p_{ref})$. According to invariant (I1), we must have $D = R(p_{ref}) < R(\mathcal{E})$. This is a contradiction since $R(.) \leq D$.

Next let us provide an upper bound for the number of time-steps required for capture. Let T_1 be the time spent in the guard state plus the time spent in the search state (the combined (SG) states). Also let T_2 be the number of steps for a pursuer, which is performing the extended lion's move, to travel the diameter of the polygon. Thus, the number of time-steps between two consecutive combined states (SG) would be $T_1 \times T_2$. Since $p_{ref} \in P$ and $v \in P$, and we achieve (P1) or (P2) after each (SG) combined state, the total capture time T would be $T = n \times n \times T_1 \times T_2$. According to [7] we have $T_2 = n \times D^2$. Next, the search time is $O(D)$ [10] and the guard time is bounded by $O(n^4 \times D^{11})$ according to Lemma 6. Hence $T_1 = n^4 \times D^{11}$ and $T = O(n^2 \times (n^4 \times D^{11}) \times (n \times D^2)) = O(n^7 \times D^{13})$. \square

6 Concluding Remarks

In this paper, we showed that a single pursuer with line-of-sight visibility can capture an evader whose speed is equal to the pursuer's in any monotone polygon. Our result provides a step toward understanding the class of environments in which a single pursuer can capture the evader.

Future research directions include designing pursuit algorithms with better capture time, and investigating other sub-classes of simply-connected polygons in terms of capturability with a single pursuer. Another interesting question is whether the monotone-pursuit strategy can be used as a subroutine for polygons partitioned into monotone pieces.

References

1. Alonso, L., Goldstein, A.S., Reingold, E.M.: Lion and Man: Upper and lower bounds. *INFORMS Journal on Computing* 4(4), 447 (1992)
2. Bhaduria, D., Isler, V.: Capturing an evader in a polygonal environment with obstacles. In: Proc. International Joint Conference on Artificial Intelligence (2011)
3. Chung, T., Hollinger, G., Isler, V.: Search and pursuit-evasion in mobile robotics. *Autonomous Robots* (3) (2011)
4. De Berg, M., Cheong, O., van Kreveld, M.: Computational geometry: algorithms and applications. Springer (2008)
5. Guibas, L., Hershberger, J., Leven, D., Sharir, M., Tarjan, R.: Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica* 2, 209–233 (1987)
6. Guibas, L.J., Claude Latombe, J., LaValle, S.M., Lin, D., Motwani, R.: Visibility-based Pursuit-Evasion in a Polygonal Environment. In: Rau-Chaplin, A., Dehne, F., Sack, J.-R., Tamassia, R. (eds.) WADS 1997. LNCS, vol. 1272, pp. 17–30. Springer, Heidelberg (1997)
7. Isler, V., Kannan, S., Khanna, S.: Randomized pursuit-evasion in a polygonal environment. *IEEE Transactions on Robotics* 21(5), 875–884 (2005)
8. Klein, K., Suri, S.: Complete information pursuit evasion in polygonal environments. In: Proceedings of the 25th Conference on Artificial Intelligence (AAAI), pp. 1120–1125 (2011)
9. Littlewood, J.E.: A mathematician’s miscellany / J. E. Littlewood. Methuen, London (1953)
10. Noori, N., Isler, V.: Lion and man with visibility in monotone polygons. Technical Report 12-005, University of Minnesota - Computer Science and Engineering (2012), http://www.cs.umn.edu/research/technical_reports.php?page=report&report_id=12-005
11. Sgall, J.: Solution of david gale’s lion and man problem. *Theor. Comput. Sci.* 259, 663–670 (2001)
12. Suzuki, I., Yamashita, M.: Searching for a mobile intruder in a polygonal region. *SIAM Journal on Computing* 21(5), 863–888 (1992)

Appendix

The pocket $pocket(v, \mathbf{r})$ is called a *simple pocket* if its boundary except the entrance \mathbf{r} is a single x -monotone chain, and the angle between \mathbf{r} and the y axis is smaller than $\frac{\pi}{2}$. See [10] for the proof of the following lemma.

Lemma 7 (Simple Pockets). *By following the MPC pursuit strategy on a simple pocket $pocket(v, \mathbf{r})$, after at most $O(n^2 \times D^3)$ time-steps, \mathcal{E} is forced to cross the entrance and exit the pocket in order to prevent being captured. At the crossing time, \mathcal{P} and \mathcal{E} both lie on \mathbf{r} and \mathcal{P} is in between v and \mathcal{E} , see Fig. 12-(e). The only difference is that during search the pursuer moves along \mathbf{r} .*

Sparse Roadmap Spanners

Andrew Dobson, Athanasios Krontiris, and Kostas E. Bekris

Abstract. Asymptotically optimal planners like PRM* guarantee that solutions approach optimal as iterations increase. Roadmaps with this property, however, may grow too large. If optimality is relaxed, asymptotically near-optimal solutions produce sparser graphs by not including all edges. The idea stems from graph spanners, which produce sparse subgraphs that guarantee near-optimal paths. Existing asymptotically optimal and near-optimal planners, however, include all sampled configurations as roadmap nodes, meaning only infinite graphs have the desired properties. This work proposes an approach, called SPARS, that provides the following asymptotic properties: (a) completeness, (b) near-optimality and (c) the probability of adding nodes to the spanner converges to zero as iterations increase, which suggests that finite-size data structures may exist that have near-optimality properties. The method brings together ideas from various planners but deviates from existing integrations of PRM* with graph spanners. Simulations for rigid bodies show that SPARS indeed provides small roadmaps and results in faster query resolution. The rate of node addition is shown to decrease over time and the quality of solutions is even better than the theoretical bounds.

1 Introduction

Motivation: Sampling-based roadmaps preprocess a configuration space (C-space) to answer multiple path planning queries online [11]. They are practical solutions for challenging, relatively high-dimensional instances, that quickly return collision-free paths with probabilistic completeness guarantees. Recent progress has led to asymptotically optimal planners [9], which, however, need to include all configurations as graph nodes to provide optimality. Many applications, however, require small roadmaps that still provide good quality paths. Resource constrained robots can better communicate, store and query smaller, sparse roadmaps that are computed offline. Interactive games have a small computational budget for path

Andrew Dobson · Athanasios Krontiris · Kostas E. Bekris
Computer Science, Rutgers University, Piscataway, NJ
e-mail: kostas.bekris@cs.rutgers.edu

planning and need to quickly load and query a roadmap online [4]. In dynamic environments, small roadmaps with multiple good quality paths are advantageous when roadmap edges must be invalidated on the fly given moving obstacles [8]. This paper shows that by relaxing optimality to near-optimality, it is possible to return a sparse roadmap that asymptotically converges to near-optimal solutions.

Sampling-Based Roadmaps: The first popular method for building a roadmap using sampling was the Probabilistic Roadmap Method (PRM) [11]. The algorithm samples a point in C_{free} , the collision-free C , and adds it as a node, then tries to connect it to the k -closest neighbors (k -PRM) or those within a δ -ball (δ -PRM). If the path is in C_{free} , an edge is added. Several variations exist and various adaptations to the path planning problem have been solved with PRM [2, 23, 21, 25, 24]. This success motivated work on the study of the probabilistic completeness guarantees PRM provides [10, 5, 6, 12]. Some variations focus on small roadmaps that provide coverage, connectivity or good path quality [25]. Visibility-based roadmaps reject nodes if not needed for coverage or connectivity [23]. The Useful Cycles approach tests edges for their usefulness in terms of path quality [18] and combined with the Reachability Roadmap Method returns high clearance paths in 2D and 3D C -spaces [4]. Algorithms exist that produce roadmaps with paths in all homotopic classes and thus deformable to optimal ones [7, 22]. Nevertheless, they build relatively dense roadmaps and smoothing can be expensive for online query resolution. Hybridization graphs combine multiple solutions into a higher quality one [20].

Roadmaps require a steering method that exactly connects two states, which is not available for many dynamical systems. Tree-based planners, e.g., RRT [13] and Expansive Spaces [6], can solve problems with dynamics and already return sparse graphs. Still, tree-based planners can benefit from sparse roadmaps that can quickly return C -space distances among obstacles [14]. RRT has been shown to converge to a suboptimal solution almost certainly [17, 9].

Asymptotic Optimality: Recent work has provided the conditions under which PRM is asymptotically optimal [9]. Asymptotic optimality implies that the quality of solutions converges to optimal as computation tends to infinity. The analysis indicates that the number of neighbors is the important variable that controls asymptotic optimality [9], as indicated in Fig. 1. A simple PRM that connects samples to neighbors within a δ -ball is asymptotically optimal, but results in a dense roadmap. The roadmap's density can be reduced by considering the k -nearest neighbors, but this version is not asymptotically optimal. PRM* and k-PRM* rectify this by selecting the minimum number of neighbors required for asymptotic optimality, which is a logarithmic function of the number of nodes. Nevertheless, all samples are added as nodes, resulting in a large graph, and it is not clear when to stop sampling.

| algorithm | edges | optimal? |
|------------------|-------------------------|--------------|
| δ -PRM | $O(n^2)$ | asympt. |
| k -PRM | $O(kn)$ | no |
| PRM* | $O(n \log n)$ | asympt. |
| k -PRM* | $O(n \log n)$ | asympt. |
| SRS | $O(an^{1+\frac{1}{d}})$ | asympt. near |
| IRS | $O(n \log n)$ | asympt. near |
| IRS2 ($m < n$) | $O(m \log m)$ | no |

Fig. 1 PRM variations and asymptotic optimality. n, m : # nodes in the roadmap.

Asymptotic Near-Optimality: A way to return sparser, good-quality roadmaps is to relax the optimality guarantees by utilizing graph spanners [19]. Spanners are subgraphs, where the shortest path between two nodes on the subgraph is no longer than t times the shortest path on the original graph, where t is the *stretch factor* of the spanner. Applying an efficient spanner [3] on the output of k-PRM* resulted in a Sequential Roadmap Spanner (SRS), which reduces the expected number of edges and provides asymptotic near-optimality, i.e., as more time is spent on constructing the roadmap, the quality of solutions converges to a value at most t times the optimal. An incremental integration of spanners with k-PRM* (IRS) has been experimentally shown to provide even better results [15]. The path quality degradation with these methods is quite smaller in practice than the theoretical guarantees. They still include every sample as a roadmap node, however. Recent work proposed IRS2[16], which has a similar objective to the current paper of not including all nodes in the graph. IRS2 applies the spanner criterion on nodes. While a simple approach, it is not possible to argue about asymptotic near-optimality or about the size of the final graph with this methodology.

Contribution: This paper presents the SPArse Roadmap Spanner (SPARS) algorithm, which: (a) is probabilistically complete, (b) can connect any two query points with a path of length:

$$t \cdot c^* + 4 \cdot \Delta, \quad (1)$$

where t and Δ are input to the algorithm, c^* is the cost of the optimum path between the query points in C_{free} with clearance at least cl , if one exists, and (c) the probability of adding new nodes and edges converges to 0.

SPARS is a possible solution to the problem of finding a compact representation for answering shortest-path queries in continuous spaces, which has been described as an important challenge for motion planning [1]. To the best of the authors' knowledge, no existing work argues about the existence of finite data structures with some form of near-optimality guarantee for continuous path planning. SPARS provides an indication that such data structures can be created. Furthermore, the method allows for a natural stopping criterion inspired by Visibility PRM [23]. The criterion relates to a probabilistic measure of how close the roadmap is to a solution that provides the desired properties.

In order to compute the spanner, the method builds an asymptotically optimal dense graph. C-space samples are included in the spanner if they are useful for coverage or connectivity purposes or if they improve path quality relative to paths on the dense graph. The algorithm tries a unique approach in that it is based on the identification of boundaries of “visibility” regions of the spanner nodes through the dense graph nodes. It eventually guarantees that all shortest paths between boundaries of “visibility” regions on the dense graph can be represented by paths on the spanner that are at most t times longer, which leads to Eq. 1. The parameters t and Δ control the sparsity of the resulting roadmap.

Simulations with rigid bodies ($SE(2)$ and $SE(3)$) indicate that the method provides sparse roadmaps and that the rate of nodes added to the spanner decreases to 0, resulting in very efficient online query resolution times, with path solution quality significantly better than theoretical bounds.

2 Setup and Nomenclature

The C-space abstraction casts a robot's position and orientation as a point q in a d -dimensional space. The collision-free part of C is denoted as C_{free} . This paper focuses on the C-spaces of planar and rigid body configurations ($SE(2)$, $SE(3)$), but SPARS is applicable if appropriate metric and sampling functions exist.

Definition 1 (The Path Planning Problem). Given the set of free configurations $C_{free} \subset C$, initial and goal points $q_{init}, q_{goal} \in C_{free}$, find a continuous path $\pi \in \Pi = \{\rho | \rho : [0, 1] \rightarrow C_{free}\}$, $\pi(0) = q_{init}$ and $\pi(1) = q_{goal}$.

Definition 2 (Robust Feasible Paths). A path planning instance ($C_{free}, q_{init}, q_{goal}$) is robustly feasible if there is a cl -robust path that solves it, for a positive clearance $cl > 0$. A path $\pi \in \Pi$ is cl -robust, if π lies entirely in the cl -interior of C_{free} .

This work aims towards a planner that provides a compact, finite-size data structure for answering shortest-path queries in continuous spaces. Such a planner must be able to identify which C-space samples are not needed as roadmap nodes. An implicit, exhaustive graph $G(V, E)$ over C_{free} can be defined by taking all the elements of C_{free} as nodes and collision free paths between them as edges. Then, a “roadmap spanner” is a subgraph $S(V_S \subset V, E_S \subset E)$ of this implicit, exhaustive graph of the continuous space with three distinct properties:

- All nodes in G are connected with a path in C_{free} to a node on S (coverage).
- S has as many connected components as G (connectivity).
- All shortest paths on S are no longer than t times the corresponding shortest paths in G (spanner property).

The properties allow for (a) arbitrary query points to connect to the roadmap, (b) paths to exist between any query points through S that can be connected in C_{free} , and (c) asymptotic near-optimality for query points that lie on S .

Definition 3 (Asympt. Near-Optimality with Additive Cost). An algorithm is asymptotically near-optimal with additive cost if, for a path planning problem ($C_{free}, q_{init}, q_{goal}$) and cost function $c : \Pi \rightarrow \mathbb{R}^{\geq 0}$ with a cl -robust optimal path of finite cost c^* , the probability it will find a path with cost $c \leq t \cdot c^* + \varepsilon$, for a stretch factor $t \geq 1$ and additive error $\varepsilon \geq 0$, converges to 1 as the iterations approach infinity.

This paper presents an algorithm that asymptotically converges to a sparse data structure with the above property.

Definition 4 (Local Planner). Given $q, q' \in \mathcal{C}$, a *local planner* computes a *local path* $L(q, q')$ connecting both configurations q and q' in the absence of obstacles. A straight-line between q and q' in \mathcal{C} is often sufficient.

Definition 5 (Distance function). The space \mathcal{C} is endowed with a *distance function* $d(q, q')$ that returns distances between configurations in the absence of obstacles.

Definition 6 (Shortest Paths). A *cl*-robust *shortest path* in \mathcal{C}_{free} is denoted as $\pi_{\mathcal{C}}$.

The shortest path between two configurations computed through the roadmap spanner S is denoted as π_S .

3 Algorithm

SPARS builds two graphs in parallel: (a) a “sparse” graph $S(V_S, E_S)$ with the desirable properties, returned upon termination, and (b) a “dense” graph $D(V_D, E_D)$, which instead of being the exhaustive graph G , corresponds to the output of δ -PRM and asymptotically stores optimal paths. The notion of sparsity in this work deviates from its standard use in graph theory (linear number of edges as a function of the number of nodes) and is used to denote that S also contains a small number of nodes, while D asymptotically includes all \mathcal{C} -space points.

SPARS operates by sampling configurations in \mathcal{C}_{free} and adding them to D . The algorithm selects a subset of these samples as nodes in the roadmap spanner S (i.e., $V_S \subset V_D$). There are four methods for promoting a node in D to S which are tested in order: “guards”, “bridges”, “mediators”, and “shortcuts”. A *guard* is added whenever a sample cannot be connected to any node already in V_S with a collision-free path of length Δ . A *bridge* is found whenever a configuration can be connected to multiple nodes in V_S which are in disconnected components of S . “*Mediators*” are added when they reveal the boundary of a Voronoi region between two sparse nodes, which do not share an edge in E_S . “*Shortcut*” nodes are added when a path through D is discovered which is significantly shorter than paths in S .

By construction, there is no edge in E_S that has length more than $2 \cdot \Delta$, where Δ is imposed to be the visibility range of a spanner node and is provided as input to the algorithm. There is also no edge in E_D with length more than δ , the maximum radius for neighborhoods in δ -PRM. Typically $\delta \ll \Delta$.

Definition 7 (Representatives). A spanner node $v \in V_S$ will be the *representative* $rep(q)$ of sample $q \in V_D$, if they can be connected with an obstacle-free path of length less than Δ and v is the closest to q node on the spanner with this property, i.e., $rep(q) = \{v \in V_S \text{ so that } L(v, q) \in \mathcal{C}_{free}, d(v, q) < \Delta \text{ and } \operatorname{argmin}_{v \in V_S} d(v, q)\}$. The *visibility region* of a spanner node v given the set of spanner nodes V_S is defined as the set of collision-free configurations that have v as their representative. Figure 2 (left) offers an illustration of the visibility region notion.

Definition 8 (Interfaces and support). The *interface* between two spanner nodes v and v' , $i(v, v')$, is the shared boundary of their visibility regions as in Figure 2 (right).

A sample $q \in V_D$ supports the interface $i(v, v')$ of its representative $v = rep(q)$, if there is a sample $q' \in V_D$ so that $L(q, q') \in E_D$ and $rep(q') = v'$, where $v' \neq v$.

Definition 9 (Midpoint). The *midpoint* between two spanner nodes v and v' along the local path $L(v, v')$ will be denoted as $m(v, v')$:

$$m(v, v') \in L(v, v') \text{ and } d(v, m(v, v')) = d(m(v, v'), v')$$

If the local path $L(v, v')$ is obstacle-free, then the midpoint $m(v, v')$ lies on the interface $i(v, v')$ between the two spanner nodes.

Algorithm 1 outlines the operation of SPARS. The input includes (i) the maximum number M of failures to add a node in S , (ii) t , the spanner's stretch factor, and (iii) the spanner visibility range Δ and (iv) the sample visibility range δ . The algorithm initializes the two graphs and sets the number of failed attempts to zero (line 1). While the failed attempts are fewer than M (line 2), the algorithm samples obstacle-free configurations q (line 3). Every q is added to the dense graph as a node and connected with an edge to existing samples $q' \in V_D$ so that $d(q, q') < \delta$ and $L(q, q') \in C_{free}$ (line 4). Then, SPARS identifies the connections $E_q \in C_{free}$ between the sample q and spanner nodes of distance less than Δ . Among these nodes, the representative $v = rep(q)$ is found (lines 5-11).

Figure 3 describes the first three cases under which SPARS adds a sample q to V_S . If none of the existing spanner nodes can be connected to q , then the sample has to be added for coverage purposes (lines 12-13). The nodes added for C-space coverage will be called “guards”. If q can connect two disconnected components, then it is also a candidate for addition (lines 14-15). The function `Add_Bridge` first checks whether v_1 and v_2 can be connected directly with an edge. If they can, this edge is added, otherwise q is added and connected to v_1 and v_2 . If a direct edge is added, its maximum length can be $2 \cdot \Delta$.

Beyond this point, the algorithm adds samples in V_S only if they support an interface. The algorithm detects if q has neighbors in D with a different representative,

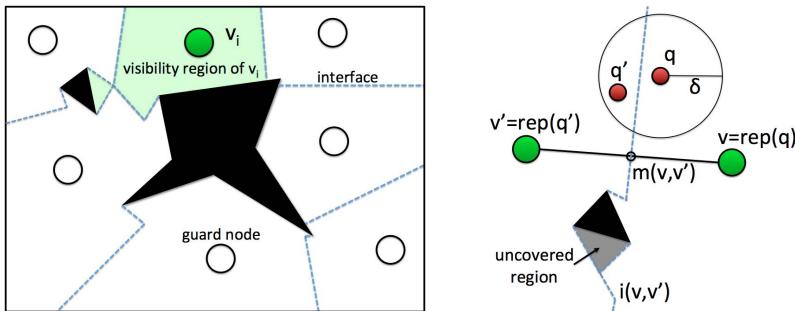


Fig. 2 (left) Visibility region of the representative v_i : the configurations that can be connected to v_i and have it as the closest node. (right) Two neighboring spanner nodes define an interface: the shared boundary of their visibility regions. Configuration q supports the interface $i(v, v')$, where v is the representative of q , if there is a configuration q' within the δ hyper-sphere of q that has a different representative ($rep(q') = v' \neq v$) and $L(q, q') \in C_{free}$.

Algorithm 1. SPARS($C_{free}, M, t, \Delta, \delta$)

```

1 failures  $\leftarrow 0$ ;  $V_D \leftarrow \emptyset$ ;  $E_D \leftarrow \emptyset$ ;  $V_S \leftarrow \emptyset$ ;  $E_S \leftarrow \emptyset$ ;
2 while failures  $< M$  do
3    $q \leftarrow \text{Sample}(C_{free})$ ;
4    $\text{Add\_Node\_}\delta\text{-PRM}(D, q, \delta)$ ;
5    $E_q = \emptyset$ ; // initialize edges from  $q$  to spanner nodes to empty
6    $v = \emptyset$ ; //  $v$  will be  $q'$ s representative
7   for  $v' \in V_S$  do // find guards that can be connected to  $q$ 
8     if  $d(v', q) < \Delta$  and  $L(v', q) \subset C_{free}$  then
9        $E_q \leftarrow E_q \cup \{L(v', q)\}$ ; // add local path
10      if  $v == \emptyset$  or  $d(v', q) < d(v, q)$  then
11         $v = v'$ ;
12      if  $E_q == \emptyset$  then // when  $q$  not visible from existing guards
13         $\text{Add\_Guard}(V_S, E_S, q)$ ;
14      else if  $q$  connects two spanner nodes  $v_1, v_2$  that were previously disconnected then
15         $\text{Add\_Bridge}(V_S, E_S, v_1, v_2, q)$ ;
16      else
17         $V' = \emptyset$ ; //  $V'$ : nodes with an interface supported by  $q$ 
18        for all  $q' \in V_D$  so that  $L(q, q') \in E_D$  do
19          if  $rep(q') \neq rep(q)$  then  $V' \leftarrow V' \cup rep(q')$ 
20        if  $V'! = \emptyset$  then
21          for  $v' \in V'$  do
22            if  $L(v, v') \notin E_S$  then
23               $\text{Add\_Mediators}(V_S, E_S, v, v', q, q')$ 
24            if  $q \notin V_S$  then
25              for  $v' \in V'$  do
26                for all  $v'' \in V_S, v'' \neq v'$  so that  $L(v, v'') \in E_S, L(v', v'') \notin E_S$  do
27                   $\Pi_S \leftarrow \{\pi_S(m(v, v'), m(v, v''))\}$ ; // midpoint paths
28                  for all  $x \in V_S, x \neq v, v', v'': L(x, v), L(x, v'') \in E_S, L(x, v') \notin E_S$  do
29                     $\Pi_S \leftarrow \Pi_S \cup \{\pi_S(m(v, v'), m(v, x))\}$ ;
30                   $\pi_S = \text{argmax}_{\pi \in \Pi_S} |\pi|$ ; // max. length path
31                  for all  $q''$  that support  $i(v, v'')$  do
32                     $\pi_D \leftarrow \text{Shortest\_Path}(D, q, q'')$ ;
33                    if  $t \cdot |\pi_D| < |\pi_S|$  then
34                       $\text{Add\_Shortcut}(V_S, E_S, V_D, E_D, \pi_D, v, v'')$ ;
35                    if  $q \notin V_S$  then  $\text{failures}++$ ;
36                    else  $\text{failures} = 0$ ;
37 return ( $V_S, E_S$ );

```

i.e., if q lies on an interface (lines 17-19). If representatives v and v' of q and q' do not share an edge, then q and q' are candidates for addition (lines 21-23). The goal is to guarantee that every two spanner nodes that share an interface will share an edge in E_S . This is handled by function Add_Mediators (Algorithm 2), which again first tries to connect v and v' directly. If $L(v, v')$ is not collision-free, then Add_Mediators will attempt to connect v and v' through the midpoint mid of q

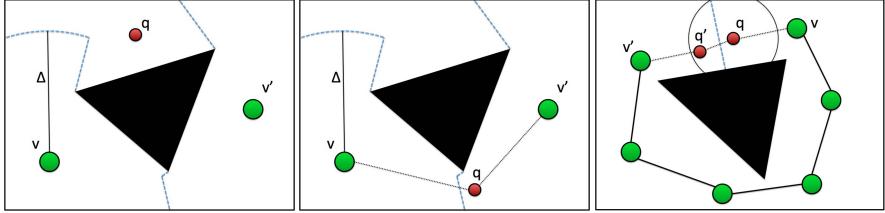


Fig. 3 (left) q added as a guard. (middle) q added as a bridge. (right) q, q' support $i(v, v')$, which does not have an edge crossing it. Samples q and q' are candidates for addition.

Algorithm 2. Add_Mediators (S, D, v, v', q, q')

```

1  $mid \leftarrow m(q, q');$ 
2 if  $L(v, v') \in C_{free}$  then
3    $E_S \leftarrow E_S \cup \{L(v, v')\};$ 
4 else if  $L(v, mid) \in C_{free}$  and  $L(v', mid) \in C_{free}$  then
5    $V_S \leftarrow V_S \cup \{mid\};$ 
6    $E_S \leftarrow E_S \cup \{L(v, mid), L(v', mid)\};$ 
7   Reduce_Interfaces( $S, D, mid$ );
8 else
9    $V_S \leftarrow V_S \cup \{q, q'\};$ 
10   $E_S \leftarrow E_S \cup \{L(v, q), L(q, q'), L(q', v')\};$ 
11  Reduce_Interfaces( $S, D, q$ );
12  Reduce_Interfaces( $S, D, q'$ );

```

and q' , i.e., $mid = m(q, q')$. If this is not possible, then both q and q' are added to S , and connected appropriately to their neighbors.

SPARS proceeds to check whether q reveals paths that are much shorter than spanner paths (lines 24-35). For all interfaces $i(v, v')$ that q supports (line 25), the algorithm considers neighbors v'' of v , so that v'' is not linked to v' (line 26). Figure 4 gives an example. For such a case, the algorithm computes the spanner path $\pi_S(m(v, v'), m(v, v''))$ (line 27) and similar spanner paths between v and neighbors x of v that do not share an edge with v' (lines 28-29). Among all of them, the algorithm stores the longest path (line 30). The algorithm aims to guarantee that all these spanner paths are less than t times longer than the shortest paths between samples along $i(v, v')$ and $i(v, v'')$ (lines 31-34).

Algorithm 3. Add_Shortcut ($V_S, E_S, V_D, E_D, \pi, v, v''$)

```

1  $prior \leftarrow v;$ 
2 for  $q \in \pi$  do
3    $V_S \leftarrow V_S \cup \{q\};$ 
4    $E_S \leftarrow E_S \cup \{L(prior, q)\};$ 
5    $prior \leftarrow q;$ 
6    $E_S \leftarrow E_S \cup \{L(prior, v'')\};$ 
7 for  $q \in \pi$  do
8   Reduce_Interfaces ( $S, D, q$ );

```

The algorithm computes the shortest path between q , which supports $i(v, v')$, and all samples that support $i(v, v'')$ and have v as their representative (lines 31-34). If the shortest path $\pi_D(q, q'')$ is shorter than t times the longest corresponding spanner path, the algorithm considers the samples along $\pi_D(q, q'')$ for addition (lines 33-34). This is accomplished by `Add_Shortcut` (Algorithm 3). If q has not been added to the spanner, the parameter *failures* is increased (lines 35-36). An efficient implementation of `Add_Shortcut` would first try to directly connect v with v'' .

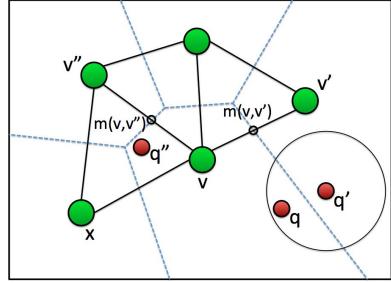


Fig. 4 q, q' support $i(v, v')$, while q'' supports $i(v, v'')$. If path $\pi_D(q, q'')$ is shorter than t times the length of spanner paths, then samples along $\pi_D(q, q'')$ are candidates for addition.

Algorithm 4. `Reduce_Interfaces` (S, D, v)

```

1 for  $v' \in V_S$  s.t.  $d(v, v') < 2 \cdot \Delta$  do
2   if  $L(v, v') \notin E_S$  then
3     for  $(q, q') \in V_D$  do
4       if  $d(q, m(v, v')) < \delta$  and  $d(q', m(v, v')) < \delta$  then
5         if  $rep(q) \neq rep(q')$  then
6           if  $L(v, rep(q')) \in C_{free}$  then
7              $E_S \leftarrow E_S \cup \{L(v, rep(q'))\};$ 
```

The method `Reduce_Interfaces` is called by all functions that add nodes to V_S . It aims to directly connect vertices which share an interface but not an edge using the local planner. Doing so prevents the algorithm from needing to add samples q and q' that support this interface.

4 Analysis

This section describes the desirable properties of SPARS. The following discussion assumes that the C-space is *expansive*, similar to related literature [5, 6].

Definition 10 (Lookout). The lookout of a set Q is the subset of configurations q in Q , such that the measure of what is visible from q outside of Q is greater than some fraction of the measure of the part of the free space outside of Q . More formally:

$$\text{LOOKOUT}_\beta(Q) = \{q \in Q \mid \omega(V(Q) \setminus Q) \geq \beta \omega(C'_{free} \setminus Q)\}$$

where C'_{free} is a C_{free} connected component and $V(Q)$ is the visibility of set Q .

Definition 11 ($\varepsilon, \alpha, \beta$ -expansiveness). A space is *expansive* if it satisfies:

- $\forall q \in C'_{free}$, where $\omega(V(q)) \geq \varepsilon \omega(C_{free})$
- $\forall Q \subseteq C'_{free}$, where $\omega(\text{LOOKOUT}_\beta(Q)) \geq \alpha \omega(Q)$

Many of the desired properties cannot be argued for poorly behaved environments. Even the existence of a finite set of nodes that achieve coverage cannot be guaranteed and depends on properties of C_{free} . Such a finite set may not always exist as is the case of environments with fractal-like boundaries.

To provide coverage, it has to be that the resulting spanner allows every collision-free configuration to be connected with a spanner node through a collision-free path.

Theorem 1 (Coverage). *Upon termination of SPARS and with probability $1 - \frac{1}{M}$, the following is true: $\forall q \in C_{free}, \exists v \in V_S$ so that $L(q, v) \in C_{free}$.*

The argument to support the above statement can be found in the presentation of the Visibility PRM [23] as SPARS is following a similar approach regarding C-space coverage and the same stopping criterion. It also relates to the property of ϵ -goodness that the *expansive space* assumption provides [6]. At a high level, each new guard inserted to S increases the coverage of C_{free} and the probability of generating configurations in non-covered regions decreases over iterations. The algorithm is then guaranteed to terminate for any finite input value M . When it stops, a probabilistic estimation of the percentage of free space not covered by spanner nodes is $\frac{1}{M}$, given uniform sampling. This means that future attempts to add spanner nodes will succeed with probability $(1 - \frac{1}{M})$. Consequently, as M goes to infinity, the resulting graph covers the entire space. This is a conservative estimate, since the algorithm assumes an artificial visibility range limit Δ for spanner nodes. The work on Visibility PRMs has shown that even for relatively complicated problems in $SE(3)$, a relatively small number of guards is needed to probabilistically cover the space. Connectivity properties of the resulting sparse roadmap spanner can be argued in a similar manner.

Theorem 2 (Connectivity). *Upon termination of SPARS and with probability 1 as M goes to infinity, for every pair of nodes $v, v' \in V_S$ that are connected with a collision-free path in C_{free} , there is a path $\pi_S(v, v')$ that connects them on S .*

The above property arises from lines 14-15 of SPARS. The algorithm adds an edge or a node every time it detects there is a way to connect two disconnected components. The probability of sampling a configuration that will connect such disconnected components of the graph depends on the environment. Narrow passages will make this connection more challenging. The probability of connecting any two disconnected components is 1, as the value of M goes to infinity, since every configuration will be eventually sampled. The combination of the last two theorems provides probabilistic completeness, at least when the algorithm samples configurations q in a uniform way.

Regarding path quality, using δ -PRM, the approach is able to construct a dense graph that provides asymptotic optimality [9]. Thus, the following lemma holds.

Lemma 1 (Asymptotic Optimality of D). *Upon termination of SPARS and with probability 1 as $M \rightarrow \infty$, for every cl-robust optimal path $\pi_C(q, q')$, there is a collision-free path $\pi_D(q, q')$ on D , so that: $|\pi_D(q, q')| \rightarrow |\pi_C(q, q')|$.*

Thus, if the length of spanner paths is bounded relative to dense paths, they will be asymptotically bounded relative to optimal C-space paths.

Lemma 2 (Coverage of Optimal Paths by S). *Consider a shortest dense path $\pi_D(q_0, q_m)$. As M goes to infinity, the probability of having a sequence of guard nodes $U = (v_1, \dots, v_n)$ with the following properties goes to 1:*

- *Each q along $\pi_D(q_0, q_m)$ belongs to the visibility region of a spanner node.*
- *v_1 is the representative of q_0 [i.e., $v_1 = rep(q_0)$] and similarly $v_n = rep(q_m)$.*
- *All the paths $L(v_i, v_{i+1})$ belong to the set E_S of the sparse roadmap spanner.*

Proof. Fig. 5 (left) illustrates the first property, which is a direct outcome of the coverage theorem. The second point holds without loss of generality. The following discussion focuses on the validity of the last point and relates to Fig. 5 (middle). Consider q_j along $\pi_D(q_0, q_m)$ that supports $i(v_i, v_{i+1})$. With probability 1 as $M \rightarrow \infty$, samples have been generated in the $\frac{cl}{2}$ neighborhood of q_j , which is guaranteed to be collision-free. Consider two samples in the $\frac{cl}{2}$ -ball centered at q_j : (a) q_j^i with spanner node v_i as its representative and (b) q_j^{i+1} with v_{i+1} as its representative.

To guarantee the algorithm will produce such samples, it has to be that the set of such configurations has non-zero measure. The region from which the sample q_j^i must be generated from is the intersection of (a) the volume of visibility paths from v_i to the optimum path π_D and (b) the $\frac{cl}{2}$ -ball centered at q_j , which has positive measure given *expansiveness* [6]. An *expansive space* can be partitioned so that the $\frac{cl}{2}$ -ball is one partition of the space. *Expansiveness* states that every subset of both partitions must have visibility of the other with positive measure. It is already true that the local path $L(v_i, q_j)$ is collision-free and contributes towards the lookout of the partition containing v_i . From the above observations, a sample q_j^i will be generated and similarly for q_j^{i+1} .

The local path $L(q_j^i, q_j^{i+1})$ must be collision-free. Furthermore, for $\delta > 0$ there are q_j^i and q_j^{i+1} for which $d(q_j^i, q_j^{i+1}) < \delta$. Then, the algorithm would have added the edge $L(q_j^i, q_j^{i+1})$ in E_D . There are two cases at this point. Either v_i and v_{i+1} are connected as desired, or the representatives are not connected. Assume v_i and v_{i+1} are not connected. Then all the requirements of the lines 17-23 of the algorithm are satisfied: (i) $q_j^i, q_j^{i+1} \in V_D$ and $L(q_j^i, q_j^{i+1}) \in E_D$, (ii) $rep(q_j^i) = v_i \neq v_{i+1} = rep(q_j^{i+1})$ (iii) $L(v_i, v_{i+1}) \notin E_S$. Thus, `Add_Mediators` would have been called. The function would either introduce the edge $L(v_i, v_{i+1})$ or it would add the samples q_j^i, q_j^{i+1} to S together with the edges $L(v_i, q_j^i)$, $L(q_j^i, q_j^{i+1})$, $L(q_j^{i+1}, v_{i+1})$. This means that $(v_1, \dots, v_i, v_{i+1}, \dots, v_n)$ can be replaced by $(v_1, \dots, v_i, q_j^i, q_j^{i+1}, v_{i+1}, \dots, v_n)$, which still covers the dense path $\pi_D(q_0, q_m)$. In this case, all the nodes between v_i and v_{i+1} are pairwise connected. Thus, if an edge doesn't exist, a new sequence can always be created where all the guards are pairwise connected. \square

A solution path $\pi_S(q_0, q_m)$ computed through the spanner will correspond to the concatenation of $L(q_0, v_1)$ and $L(v_n, q_m)$ with local paths $L(v_i, v_j)$ along the sequence $U = (v_1, \dots, v_n)$. Note that not all consecutive guard edges of the type

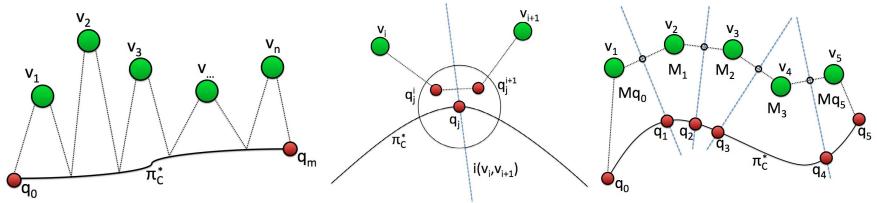


Fig. 5 (left) For every $\pi_D(q_0, q_m)$, there is a sequence of V_S nodes $\{v_1, \dots, v_n\}$ so that every q along the path is “visible” by at least one v_i . For each (v_i, v_{i+1}) there is an edge $L(v_i, v_{i+1})$ in E_S . (middle) If there is no edge $L(v_i, v_{i+1})$, the samples q_j^i and q_j^{i+1} would have been added to S . (right) The spanner path $\pi_S(q_0, q_5)$ can be split into: $\{M_{q_0}, M_1, M_2, M_3, M_{q_5}\}$. Each M_i is a path of the form $\pi_S(m(v_i, v_{i+1}), m(v_{i+1}, v_{i+2}))$. The first and last segments connect q_0 and q_5 to the spanner path.

$L(v_i, v_{i+1})$ need to be part of the shortest spanner path, as there can be short-cuts along U . Then, the path on the spanner can be decomposed into segments $M_i = \pi_S(m(v_{i-1}, v_i), m(v_i, v_{i+1}))$, which start at a midpoint between two spanner nodes and stop at another midpoint. There are also the following special segments: $M_{q_0} = (L(q_0, v_1), L(v_1, m(v_1, v_2)))$ and $M_{q_m} = (L(m(v_{n-1}, v_n), v_n), L(v_n, q_m))$. These segments connect the start or the final configuration with spanner midpoints. Figure 5 (right) provides an illustration. Overall: $\pi_S(q_0, q_m) \equiv (M_{q_0}, M_1, \dots, M_k, M_{q_m})$.

Lemma 3 (Connection Cost). *The length of M_{q_0} and M_{q_m} of a path between configurations q_0, q_m computed through the spanner is bounded by: 4Δ .*

Proof. Consider the segment M_{q_0} . The length of $L(q_0, v_1)$ cannot be longer than Δ because v_1 is the representative of q_0 . Similarly, the distance between v_1 and $m(v_1, v_2)$ cannot be more than Δ , since $m(v_1, v_2)$ is the midpoint between two spanner nodes that share an edge. Overall $|M_{q_0}| < \frac{4\Delta}{2}$. The same bound can be shown for the segment M_{q_m} . Consequently: $|M_{q_0}| + |M_{q_m}| < 4 \cdot \Delta$. \square

Lemma 4 (Spanner Property). *Segments M_i are bounded by $t \cdot |\pi_D(q_{i-1}, q_i)|$ in length, where q_i lies at the intersection of $\pi_D(q_0, q_m)$ with the interface $i(v_i, v_{i+1})$.*

Proof. The arguments presented here correspond to Fig. 6. Given lemma 2, the edges $L(v_{i-1}, v_i)$ and $L(v_i, v_{i+1})$ must belong to the set E_S , as the algorithm ensures that all interfaces will eventually have edges connecting the vertices inducing them. Assume the true optimal path goes through the visibility region of vertex v_i on sequence U . There are two cases to be considered: (a) there is no edge $L(v_{i-1}, v_{i+1})$ in E_S (Fig. 6 (left)) or (b) there is such an edge (Fig. 6 (right)).

To ensure that the spanner property is upheld in the first case, it is sufficient that the path on S from $m(v_{i-1}, v_i)$ to $m(v_i, v_{i+1})$ is shorter than t times the length of the shortest dense path between interfaces $i(v_{i-1}, v_i)$ and $i(v_i, v_{i+1})$. Lines 24-34 of SPARS, however, ensure that this property is true. When the newly sampled configuration q , is on an interface $i(v, v')$, the algorithm will search for paths from v' to all v'' where $L(v', v'') \notin E_S$. The equivalent of v in Fig. 6 (left) is v_i , v' would be v_{i+1} and v'' would be v_{i-1} or vice versa.

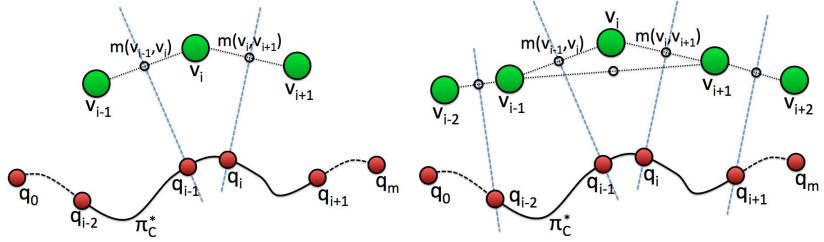


Fig. 6 (left) The optimal path between q_{i-1} and q_i is represented by the segment M_i on the spanner. The algorithm checks whether the length of M_i is less than $t \cdot |\pi_D(q_{i-1}, q_i)|$. (right) Nodes u_{i-1} and u_{i+1} are connected with an edge in this case. Thus, the spanner path will not contain the node u_i . Nevertheless, the algorithm checks whether the spanner path $[m(u_{i-2}, u_{i-1}), [u_{i-1}, m(u_{i-1}, u_{i+1})]$ is shorter than t times the shortest dense path between the interfaces $i(u_{i-2}, u_{i-1})$ and $i(u_{i-1}, u_{i+1})$. The same is true for $i(u_{i-1}, u_{i+1})$ and $i(u_{i+1}, u_{i+2})$.

In the second case, the algorithm will not check the interface $i(v_{i-1}, v_i)$ and $i(v_i, v_{i+1})$, because the nodes v_{i-1} and v_{i+1} share an edge. Nevertheless, the algorithm considers additional spanner paths when it compares against the length of dense paths. In particular, the algorithm will check whether the spanner path from $m(v_{i-2}, v_{i-1})$ to $m(v_{i-1}, v_{i+1})$ is shorter than t times the length of the shortest dense paths between the interfaces $i(v_{i-2}, v_{i-1})$ and $i(v_{i-1}, v_i)$. Notice the difference in the second vertex of the second midpoint and the second interface. In this case, node v_{i-1} in Fig. 6 (right) is equivalent to node v in lines 24-34 of SPARS, v_{i-2} is equivalent to node v' , v_i is equivalent to v'' and v_{i+1} is equivalent to node x . The algorithm will also provide the same guarantee for the spanner path $m(v_{i-1}, v_{i+1})$ to $m(v_{i+1}, v_{i+2})$ and overall: $|\pi_S(m(v_{i-2}, v_{i-1}), m(v_{i+1}, v_{i+2}))| < t \cdot (|\pi_D(q_{i-2}, q_{i-1})| + |\pi_D(q_i, q_{i+1})|) < t \cdot (|\pi_D(q_{i-2}, q_{i-1})| + |\pi_D(q_{i-1}, q_i)| + |\pi_D(q_i, q_{i+1})|)$ and the spanner property is still satisfied. If the spanner property is violated, SPARS will call function `Add_Shortcut`, which will either directly connect the corresponding vertices or add the dense path into the spanner graph. \square

Theorem 3 (Asymptotic Near-Optimality with Additive Cost). As $M \rightarrow \infty$ in SPARS and $\forall q_0, q_m \in C_{free}$: $|\pi_S(q_0, q_m)| < t \cdot |\pi_D(q_0, q_m)| + 4 \cdot \Delta$.

Proof. Lemma 2 provides the existence of a sequence U of spanner nodes that can be used to solve path planning queries for any q_0, q_m (i.e., it is possible to connect q_0 and q_m to the sequence U , and all the nodes in the sequence are pairwise connected). The combination of lemmas 3 and 4 indicates that the solution path $\pi_S(q_0, q_m)$ computed through this sequence has length that is bounded relative to the length of the dense path as follows: $|\pi_S(q_0, q_m)| < t \cdot |\pi_D(q_0, q_m)| + 4 \cdot \Delta$, where the additive cost arises from the cost of connecting the query points to the spanner nodes. Lemma 1 indicates that the paths computed through the dense graph converge asymptotically to the optimal ones. The combination of all these lemmas proves this theorem. \square

An important issue is whether SPARS adds an infinite number of nodes to S in order to provide the above path quality properties. Towards this objective, this paper argues the following result.

Theorem 4 (Termination of Node Addition). *As the number of iterations goes to infinity, the probability of SPARS adding nodes to S goes to zero.*

Proof outline: There are four ways that nodes are added in the spanner S and it is necessary to show that the probability of adding each type of node goes to zero.

Nodes for coverage are added when a sample q lies outside the visibility range of all existing nodes in V_S . Theorem 1 already argues that the probability of adding guards diminishes to zero as the number of iterations increases.

Nodes for connectivity are added when a sample q connects two disconnected components. Eventually, enough nodes for ensuring coverage will be added in S . Given an otherwise finite number of nodes, the number of samples needed to connect disconnected components is finite.

Nodes for ensuring interfaces have edges are added when a sample q has a neighbor q' in D and their representatives v and v' do not share an edge. In general, when a node is added to S , multiple new interfaces are defined, but the algorithm employs the function `Reduce.Interfaces`, which adds edges along interfaces. There are three cases for two neighboring nodes: (i) The interface lies outside C_{free} and there is no need to add an edge. (ii) $L(v, v')$ lies within C_{free} and the edge can be immediately added. (iii) $L(v, v')$ is not within C_{free} , but there exists a portion of the interface in C_{free} . In this case, new nodes need to be added, which, however, will also add new interfaces, which need to be connected by an edge. Nevertheless, the newly generated interfaces will eventually have to fall into case (i) or case (ii). Otherwise, it has to be that every time a node is added, there will exist an obstacle which blocks the local path, which violates *expansiveness assumptions*. Overall, all interfaces will be eventually connected with an edge, so the probability of adding nodes for this reason goes to 0.

Nodes for ensuring path quality are added when a sample q supports an interface and reveals a path in D that is significantly shorter than the corresponding path in S . SPARS always adds the minimal amount of nodes possible while ensuring that the path quality properties of the algorithm hold. When adding a dense path to S , SPARS will first try to smooth this path. In some cases, the path cannot be smoothed, which will result in the addition of nodes to S . The addition of new nodes could potentially always create new visibility regions where a dense path could be found that is sufficiently shorter than the spanner paths. Eventually, however, the spanner nodes which the algorithm attempts to connect will be within cl of each other and it would be guaranteed that $L(v, v')$ is collision-free and the nodes can be connected directly. Therefore, SPARS will eventually stop adding nodes in this fashion.

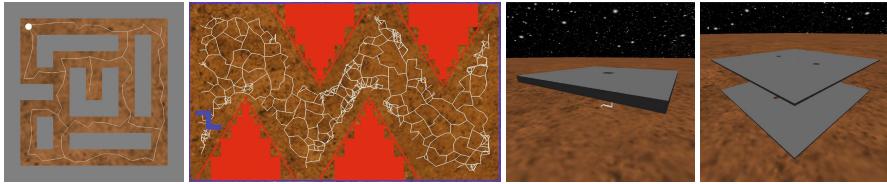


Fig. 7 Environments for testing (left to right): $SE(2)$: 2D Maze, Teeth, $SE(3)$: 3D Hole and Many Holes. The 2D Maze tests performance for constrained spaces, while the Teeth environment tests performance in large, open spaces. The output roadmap is visualized for the $SE(2)$ case.

5 Evaluation

The algorithm was implemented in a C++ simulation environment. A concern for the method is its space requirements. To improve space efficiency, information was kept to a minimum on the dense graph's nodes and edges. Dense nodes take up approximately 40 bytes of memory in $SE(2)$ examples and 72 bytes in $SE(3)$ examples. Sparse nodes use a larger amount of memory, as they retain a list of all the nodes in the dense graph which they represent, introducing a cost of 4 bytes per dense node. All edges use approximately 12 bytes of memory.

The implementation performed several optimizations compared to the algorithm described. In order to speed up convergence, connections are attempted to neighbors of newly added sparse nodes in the spanner. The neighbors considered are all nodes within $2 * \Delta$ distance, since it may take time for the algorithm to identify through sampling that two spanner nodes share an interface. Furthermore, whenever paths are added to the spanner, the algorithm always attempts to directly connect points, and if it cannot, it will try smoothing the paths. A kd-tree was used for fast nearest-neighbor query resolution. Nevertheless, it is often much more practical to consider the neighbors of nodes already in the spanner or the graph and then prune them according to their distance to the selected node. This technique was used heavily, especially for querying the dense graph.

Experiments were run in four environments in $SE(2)$ and $SE(3)$, as shown in Figure 7 for kinematic rigid bodies. The 2D Maze used a disk system, while the Teeth environment used a 'Z' shaped body. For both cases in $SE(3)$, a variation of the 'Z' robot was used which has one of its legs coming off in an orthogonal direction to the other leg. Runs in $SE(2)$ were tested with $M = 4000$, $t = 3$, and $\Delta = 20$, while runs in $SE(3)$ were tested with $M = 1400$, $t = 3$, and $\Delta = 25$. All runs were terminated if the stopping criterion was not satisfied after 60 minutes.

SPARS was compared against asymptotically optimal δ -PRM and the IRS2 approach. The algorithms were compared in terms of path quality, query resolution time, and number of nodes and edges in the final roadmaps. Results shown for δ -PRM were extracted from the dense graph D constructed by the SPARS algorithm. In this way, the stopping criterion for δ -PRM relates to the stopping condition of SPARS. Figure 8 shows an overview of the algorithms' performance.

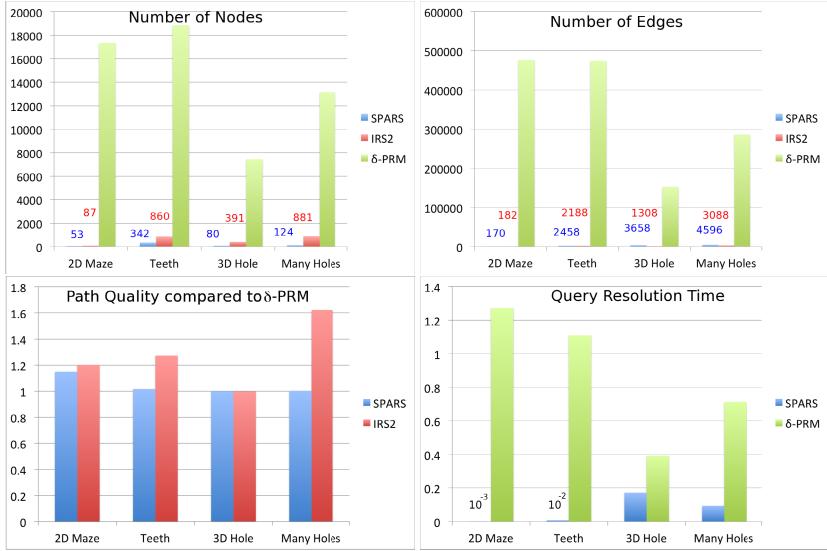


Fig. 8 A comparison of various aspects of the algorithms. (upper left) The number of nodes in the final roadmaps. (upper right) The number of edges in each roadmap. (lower left) Path length for SPARS and IRS2 as a percentage of paths from δ -PRM. (lower right) Average query resolution time for 100 queries.

SPARS is the best alternative in terms of aggressively pruning the number of nodes in the final roadmap and is able to retain many-orders of magnitude smaller roadmaps than δ -PRM. The related IRS2 method retains slightly more nodes than SPARS but returns fewer edges. Nevertheless, given that nodes have higher space requirements relative to edges, the overall space requirements of SPARS are smaller than IRS2. This had a significant effect on query resolution times which were significantly shorter for SPARS relative to δ -PRM. In terms of path length, the asymptotically near-optimal algorithms were able to return solutions which were very close to those computed by δ -PRM. Note that δ -PRM suffers from the side-effect of having multiple nodes along the solution path. Smoothing the final paths results in comparable path lengths between the various methods.

6 Discussion

This paper describes the first approach, to the best of the author's knowledge, that indicates that it is possible to build a finite size data structure, which can be used to answer path queries in continuous spaces with near-optimality guarantees. Simulations indicate that the resulting graph is orders of magnitude sparser (i.e., it has fewer nodes and edges) than graphs with asymptotic optimality guarantees. This results in significantly shorter query resolution times. The quality of paths computed

on the sparse roadmap spanner is shown to be in practice significantly better than the theoretical guarantees.

The most important direction for this research is figuring out whether S truly converges to a finite size, or if an infinite number of nodes are still needed. Future research will address the dependency of the current approach to the input parameters, such as the visibility ranges Δ , δ and the maximum allowed number of failures M . An interesting question is whether the proposed method can discover the important homotopic classes in a C-space and how does it compare against methods that aim to identify homotopic classes [7, 22]. It is also important to consider alternative methods that do not need to store during the construction of the spanner the entire graph D returned by the asymptotically optimal planner. Such a development could reduce the space and time requirements of the spanner's construction. In a similar direction, it will be helpful to introduce steps that improve computational efficiency, such as tools for reducing the cost of A* searches on the dense graph or the cost of nearest-neighbor queries.

Acknowledgements. Work by the authors has been supported by NSF CNS 0932423. Any conclusions expressed here are of the authors and do not reflect the views of the sponsors.

References

- [1] Agarwal, P.: Compact Representations for Shortest-Path Queries. Appeared at the IROS 2012 Workshop on Progress and Open Problems in Motion Planning (2011)
- [2] Amato, N.M., Bayazit, O.B., Dale, L.K., Jones, C., Vallejo, D.: OBPRM: An Obstacle-based PRM for 3D Workspaces. In: WAFR, pp. 155–168 (1998)
- [3] Baswana, S., Sen, S.: A Simple and Linear Time Randomized Algorithm for Computing Spanners in Weighted Graphs. Random Structures and Algorithms 30(4), 532–563 (2007)
- [4] Geraerts, R., Overmars, M.H.: Creating High-Quality Roadmaps for Motion Planning in Virtual Environments. In: IROS, Beijing, China, pp. 4355–4361 (2006)
- [5] Hsu, D., Kavraki, L., Latombe, J.C., Motwani, R., Sorkin, S.: On Finding Narrow Passages with Probabilistic Roadmap Planners. In: WAFR, Houston, TX (1998)
- [6] Hsu, D., Kindel, R., Latombe, J.C., Rock, S.: Randomized Kinodynamic Motion Planning with Moving Obstacles. IJRR 21(3), 233–255 (2002)
- [7] Jailllet, L., Simeon, T.: Path Deformation Roadmaps. In: Akella, S., Amato, N.M., Huang, W.H., Mishra, B. (eds.) Algorithmic Foundation Robotics VII. STAR, vol. 47, pp. 19–34. Springer, Heidelberg (2008)
- [8] Kallman, M., Mataric, M.: Motion Planning Using Dynamic Roadmaps. In: ICRA, New Orleans, LA, vol. 5, pp. 4399–4404 (2004)
- [9] Karaman, S., Frazzoli, E.: Sampling-based Algorithms for Optimal Motion Planning. IJRR 30(7), 846–894 (2011)
- [10] Kavraki, L.E., Kolountzakis, M.N., Latombe, J.C.: Analysis of Probabilistic Roadmaps for Path Planning. IEEE TRA 14(1), 166–171 (1998)
- [11] Kavraki, L.E., Svestka, P., Latombe, J.C., Overmars, M.: Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces. IEEE TRA 12(4), 566–580 (1996)

- [12] Ladd, A.M., Kavraki, L.E.: Measure Theoretic Analysis of Probabilistic Path Planning. *IEEE TRA* 20(2), 229–242 (2004)
- [13] LaValle, S.M., Kuffner, J.J.: Randomized Kinodynamic Planning. *IJRR* 20, 378–400 (2001)
- [14] Li, Y., Bekris, K.E.: Learning Approximate Cost-to-Go Metrics to Improve Sampling-based Motion Planning. In: *IEEE ICRA*, Shanghai, China (2011)
- [15] Marble, J.D., Bekris, K.E.: Asymptotically Near-Optimal is Good Enough for Motion Planning. In: *ISRR*, Flagstaff, AZ (2011)
- [16] Marble, J.D., Bekris, K.E.: Towards Small Asymptotically Near-Optimal Roadmaps. In: *IEEE ICRA*, Minnesota, MN (2012)
- [17] Nechushtan, O., Raveh, B., Halperin, D.: Sampling-Diagram Automata: A Tool for Analyzing Path Quality in Tree Planners. In: Hsu, D., Isler, V., Latombe, J.-C., Lin, M.C. (eds.) *Algorithmic Foundations of Robotics IX*. STAR, vol. 68, pp. 285–301. Springer, Heidelberg (2010)
- [18] Nieuwenhuisen, D., Overmars, M.H.: Using Cycles in Probabilistic Roadmap Graphs. In: *IEEE ICRA*, pp. 446–452 (2004)
- [19] Peleg, D., Schäffer, A.: Graph Spanners. *Journal of Graph Theory* 13(1), 99–116 (1989)
- [20] Raveh, B., Enosh, A., Halperin, D.: A Little More, a Lot Better: Improving Path Quality by a Path-Merging Algorithm. *IEEE TRO* 27(2), 365–370 (2011)
- [21] Sanchez, G., Latombe, J.C.: A Single-Query, Bi-Directional Probabilistic Roadmap Planner with Lazy Collision Checking. In: *ISRR*, pp. 403–418 (2001)
- [22] Schmitzberger, E., Bouchet, J.L., Dufaut, M., Wolf, D., Husson, R.: Capture of Homotopy Classes with Probabilistic Roadmap. In: *IEEE/RSJ IROS*, pp. 2317–2322 (2002)
- [23] Simeon, T., Laumond, J.P., Nissoux, C.: Visibility-based Probabilistic Roadmaps for Motion Planning. *Advanced Robotics Journal* 41(6), 477–494 (2000)
- [24] Varadhan, G., Manocha, D.: Star-shaped Roadmaps: A Deterministic Sampling Approach for Complete Motion Planning. *IJRR* (2007)
- [25] Xie, D., Morales, M., Pearce, R., Thomas, S., Lien, J.L., Amato, N.M.: Incremental Map Generation (IMG). In: Akella, S., Amato, N.M., Huang, W.H., Mishra, B. (eds.) *Algorithmic Foundations of Robotics*. STAR, vol. 47, pp. 53–68. Springer, Heidelberg (2008)

Toggle PRM: A Coordinated Mapping of C-Free and C-Obstacle in Arbitrary Dimension

Jory Denny and Nancy M. Amato

Abstract. Motion planning has received much attention over the past 40 years. More than 15 years have passed since the introduction of the successful sampling-based approach known as the Probabilistic RoadMap Method (PRM). PRM and its many variants have demonstrated great success for some high-dimensional problems, but they all have some level of difficulty in the presence of narrow passages. Recently, an approach called Toggle PRM has been introduced whose performance does not degrade for 2-dimensional problems with narrow passages. In Toggle PRM, a simultaneous, coordinated mapping of both \mathbb{C}_{free} and \mathbb{C}_{obst} is performed and every connection attempt augments one of the maps – either validating an edge in the current space or adding a configuration ‘witnessing’ the connection failure to the other space. In this paper, we generalize Toggle PRM to d -dimensions and show that the benefits of mapping both \mathbb{C}_{free} and \mathbb{C}_{obst} continue to hold in higher dimensions. In particular, we introduce a new narrow passage characterization, α - ε -separable narrow passages, which describes the types of passages that can be successfully mapped by Toggle PRM. Intuitively, α - ε -separable narrow passages are arbitrarily narrow regions of \mathbb{C}_{free} that separate regions of \mathbb{C}_{obst} , at least locally, such as hallways in an office building. We experimentally compare Toggle PRM with other methods in a variety of scenarios with different types of narrow passages and robots with up to 16 DOF.

1 Introduction

In robotics, planning a valid (e.g., collision-free) path for a movable object (robot) is challenging. *Motion planning*, as this is commonly called, has been thoroughly studied [23] and has been shown to be PSPACE-hard. Motion planning has broad use outside robotics in areas such as gaming/virtual reality [14] and bioinformatics [25]. Hence, it is vital to find computationally efficient solutions to this problem.

Jory Denny · Nancy M. Amato
Parasol Lab, Department of Computer Science and Engineering,
Texas A&M University, College Station, TX, USA
e-mail: {jdenny, amato}@cse.tamu.edu

Sampling-based planners [12] were a major breakthrough in motion planning. These algorithms were able to solve many previously unsolved problems, especially for high-dimensional configuration space (\mathbb{C}_{space}). Sampling-based motion planners explore \mathbb{C}_{space} and store configurations and some information about their connectivity in \mathbb{C}_{free} . One class of planners, *Probabilistic RoadMap* methods (PRMs) [12], build a roadmap (graph) representing the connectivity of \mathbb{C}_{free} . The first phase in this process, *node generation*, is where collision-free configurations are sampled and added as nodes to the roadmap. In the second phase, *node connection*, neighboring nodes are selected by a *distance metric* as potential candidates for connection. Then, simple *local planners* attempt connections between the selected nodes; successful connections are represented as roadmap edges. While these methods have been shown to be *probabilistically complete*, narrow passages, or small volumes/regions of free space (\mathbb{C}_{free}), remain difficult for them to map. In particular, it has been shown that the volume of such passages impacts the efficiency of a sampling-based planner [11]. In simple terms, the probability of generating a configuration within the narrow passage is directly related to the volume of the passage itself. Thus, the tighter the corridor is, the more difficult it is to generate samples in it. Section 2 briefly discusses the many attempts to address this weakness of sampling-based planning methods [1, 3, 26, 10].

Recently, a new sampling-based motion planning strategy that maps both free space (\mathbb{C}_{free}) and obstacle space (\mathbb{C}_{obst}) in a simultaneous, coordinated fashion has shown to be useful in 2-dimensional problem spaces for both roadmap construction (Toggle PRM [6]) and local planning (Toggle LP [7]). In particular, in Toggle PRM, coordinated learning of both \mathbb{C}_{free} and \mathbb{C}_{obst} was shown to theoretically guarantee a higher probability of sampling within narrow regions of \mathbb{C}_{space} for 2 DOF problems, while Toggle LP extended local planning attempts into a 2D triangular subspace of \mathbb{C}_{space} to yield better connectivity of roadmaps as opposed to the traditional straight-line connection [12]. The intuition behind Toggle PRM, and the main departure from traditional PRMs, is the philosophy that every connection attempt, successful or not, reveals information about connectivity in one of the spaces and hence augments one of the maps – either validating an edge in the current space or discovering a configuration in the other space ‘witnessing’ the connection failure. For example, a failed connection between two collision configurations on either side of a narrow passage would lead to the discovery of a configuration in the narrow passage. Connection attempts between connected components of a roadmap also become increasingly interesting for Toggle PRM. Ultimately, Toggle PRM might be able to induce disconnectivity for planning problems, something that is not feasible for PRMs (see Figure 1). As shown in [6], utilizing the witnesses of connection failures can greatly accelerate the discovery and mapping of narrow passages for 2 DOF problems.

This paper further develops the Toggle PRM philosophy for sampling-based motion planning and generalizes it to higher dimensions. We show that the theoretical and experimental benefits of a coordinated mapping of both \mathbb{C}_{free} and \mathbb{C}_{obst} , demonstrated in [6], continue to hold in higher dimensions. In particular, we show that Toggle PRM results in significant improvements to the effectiveness and efficiency

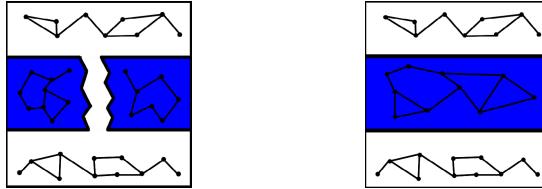


Fig. 1 A map of \mathbb{C}_{obst} can reveal important connectivity information possibly differentiating between a problem with (left) and without (right) a narrow passage

of sampling-based planners, for a particular class of narrow passages, α - ε -separable narrow passages, that we define here. Particular contributions of this work include:

- Generalizing Toggle PRM to arbitrary dimension.
- Defining a new characterization of narrow passages, α - ε -separable narrow passages, that can be mapped well by Toggle PRM.
- Providing experimental results for problems that confirm Toggle PRM's ease of mapping α - ε -separable narrow passages and demonstrate improvement over other PRM methods.

2 Preliminaries and Related Work

Motion Planning Preliminaries. A robot is a movable object whose position and orientation can be described by n parameters, or *degrees of freedom* (DOFs), each corresponding to an object component (e.g., object positions, object orientations, link angles, link displacements). Hence, a robot's placement, or configuration, can be uniquely described by a point (x_1, x_2, \dots, x_n) in an n dimensional space (x_i being the i th DOF). This space, consisting of all possible robot configurations (feasible or not) is called *configuration space* ($\mathbb{C}_{\text{space}}$) [16]. The subset of all feasible configurations is the *free space* (\mathbb{C}_{free}), while the union of the unfeasible configurations is the *blocked* or *obstacle space* (\mathbb{C}_{obst}). Thus, the motion planning problem becomes that of finding a continuous trajectory in \mathbb{C}_{free} connecting the points in \mathbb{C}_{free} corresponding to the start and the goal configurations. In general, it is intractable to compute explicit \mathbb{C}_{obst} boundaries [23], but we can often determine whether a configuration is feasible or not quite efficiently, e.g., by performing a collision detection (CD) test in the *workspace*, the robot's natural space.

PRM Variants. PRMs have known difficulty in solving narrow passage problems [11]. To address this deficit, many variants have been proposed, e.g., [21, 2, 13]. The performance of these methods is dependent on the problem instance. Below is a description of the PRM variants most related to this work.

Several methods have been proposed that attempt to generate samples on or near \mathbb{C}_{obst} surfaces. The first, Obstacle-Based PRM (OBPRM) [1], is efficient at solving problems in practice but does not sample with a known density on surfaces and thus its theoretical benefits are not understood. Recently, a new method called uniform

OBPRM (UOBPRM) [9] has been proposed that guarantees a uniform distribution of samples on \mathbb{C}_{obst} surfaces, albeit for a larger computational cost than OBPRM. Gaussian PRM [3] attempts to find configurations near obstacles by generating two samples at a Gaussian d away from each other and tests if it straddles the \mathbb{C}_{obst} boundary. Bridge Test PRM [10] is similar to Gaussian PRM. It is a filtering technique that attempts to generate two invalid configurations, and only if this is successful it tests the midpoint for validity. Hence, this test attempts to bridge the gap and generate configurations in narrow passages. However, this test may fail many times before successfully bridging a gap, depending on the narrow passage.

Medial Axis PRM (MAPRM) [26, 15] pushes randomly sampled configurations to the medial axis. It provably improves the probability of sampling in most narrow passages over uniform random sampling, but is computationally intensive even when approximations are used.

Toggle PRM [6] performs a coordinated mapping of \mathbb{C}_{free} and \mathbb{C}_{obst} . It was shown theoretically and experimentally to perform better than uniform random sampling, and most of the methods described here, in 2D \mathbb{C}_{space} . Toggle Local Planning [7] used these ideas to extend local planning into a 2-dimensional triangular subspace of \mathbb{C}_{space} , which was seen to result in improved connectivity over the traditional straight-line local planner [12]. Toggle PRM differs from Bridge Test by not only testing an entire line segment for crossing narrow passages but also by fully mapping \mathbb{C}_{obst} , including connections.

Variants Utilizing Collision Information. There have been many improvements upon PRM that use information gain or learning techniques to guide sampling towards specific regions of the environment. Many of these methods utilize information about nodes in collision, however none of them create a map of \mathbb{C}_{obst} . This section briefly describes some of these methods.

There is a class of motion planning techniques called Feature Sensitive Motion Planning [18, 20, 24, 27] that apply a divide-and-conquer approach by breaking up the environment into regions, solving each region, and combining the solutions. Although these techniques do not map \mathbb{C}_{obst} these approaches often use information from samples, both valid and invalid to find appropriate regions.

Other efforts, such as [4, 5, 19, 22], seek to model \mathbb{C}_{space} to more efficiently map \mathbb{C}_{free} . In [4, 5] a machine learning technique, locally weighted regression, is used to create an approximate model of \mathbb{C}_{space} that biases sampling toward areas which improve the model, called active sampling.

3 Toggle PRM

In this section, we provide an overview of the Toggle PRM algorithm [6] and its theoretical benefits. To generalize the algorithm to arbitrary dimension, certain aspects of higher dimensionality must be considered, i.e., connection attempts along a one-dimensional line through \mathbb{C}_{space} (e.g., straight-line local planning) are less likely to intersect a small volume of \mathbb{C}_{space} . To extend the algorithm to higher dimensions, issues such as this must be addressed. In the following, we first sketch the Toggle

Algorithm 1. *Toggle PRM.* Local planners and connectors are modified to return configurations from failed connection attempts.

Input: Sampler s

```

1: Roadmaps  $G_{free}, G_{obst} = \emptyset$ 
2: while  $\text{!done}$  do
3:   Queue  $q \leftarrow s.\text{Sample}()$ 
4:   while  $\neg q.\text{isEmpty}()$  do
5:     Configuration  $n = q.\text{dequeue}()$ 
6:     if  $n$  is valid then
7:        $G_{free}.\text{AddNode}(n)$ 
8:        $\text{collisionNodes} \leftarrow \text{TogglePRMConnect}(G_{free})$ 
9:        $q.\text{enqueue}(\text{collisionNodes})$ 
10:    else { $n$  is invalid, and local planners “validate” paths through  $\mathbb{C}_{obst}$ }
11:      Toggle Validity
12:       $G_{obst}.\text{AddNode}(n)$ 
13:       $\text{validNodes} \leftarrow \text{TogglePRMConnect}(G_{obst})$ 
14:       $q.\text{enqueue}(\text{validNodes})$ 
15:      Toggle Validity
16:    end if
17:   end while
18: end while

```

PRM algorithm, including a new connection strategy that is better at discovering and mapping certain types of “locally separable” small volume regions of \mathbb{C}_{space} . We then formalize the concept of locally separable by defining α - ϵ -separable narrow passages and proving that they can be mapped well by Toggle PRM.

Algorithm. An overview of *Toggle PRM* is shown in Algorithm 1. The major difference from traditional PRM methods is that roadmaps of both \mathbb{C}_{free} and \mathbb{C}_{obst} are constructed. The strategy is based on the fact that each successful connection attempt reveals connectivity information about one of the spaces, while each unsuccessful attempt witnesses a crossing of the opposite space. In particular, the local planning method used to verify edge connections returns a “witness” configuration if the connection attempt fails, which is used to augment the opposite spaces map.

The algorithm uses a sampler to generate nodes (e.g., uniform random sampling), and a connector, which is a combination of a local planner (e.g., straight-line) and a distance metric (e.g., Euclidean), to find the neighboring nodes to which to attempt connections. Algorithm 1 differs from the original description of Toggle PRM in the connection phase, described in Algorithm 2, which uses a heuristic particularly designed for mapping α - ϵ -separable narrow passages. We start by initializing two roadmaps, G_{free} and G_{obst} . While the map construction is not done (e.g., until the map is a certain size, a query is solved, etc.), then the following happens: samples are generated in both \mathbb{C}_{free} and \mathbb{C}_{obst} , and connections are attempted in both maps using Algorithm 2. Witnesses returned by failed connections are added to the opposite space’s roadmap. When connections are attempted in \mathbb{C}_{obst} , toggling the meaning of validity is necessary for the local planner to “validate” paths through

Algorithm 2. *TogglePRMConnect*. Local planners and connectors are modified to return configurations from failed connection attempts.

Input: Roadmap G

Output: Collision Nodes C

```

1: for all Newly added nodes  $n$  do
2:   repeat
3:     Attempt connection from  $n$  to a neighbor  $b$  in a different CC of  $G$  as  $n$ 
4:     if Connection attempt fails then
5:       Add witness of failed connection attempt to  $C$ 
6:     else
7:        $G.AddEdge(n, b)$ 
8:     end if
9:   until Connection attempt fails
10: end for
11: return  $C$ 
```

\mathbb{C}_{obst} , i.e., all configurations along a local plan must be located in \mathbb{C}_{obst} . The algorithm repeats until the stopping criterion is reached, or it is shown to be unreachable. The algorithm uses a *queue* to incrementally build the roadmap.

The major difference from the 2D version of Toggle PRM is the connection phase. While any connection algorithm could be used, we introduce a new strategy designed to do well when connections would cross narrow passages that separate \mathbb{C}_{obst} . Selecting appropriate neighbors for connection is crucial since it will either allow you to learn about the connectivity of the current space, i.e., successful connection attempts, or about the opposite space, i.e., failed connection attempts. However, if learning is biased towards the opposite space too much, we “flood” the queue, which effectively oversamples \mathbb{C}_{space} by attempting too many wasteful (and costly) connections to similar nodes within the roadmaps. Ideally, we want to connect to the current space’s roadmap *and* learn enough about the opposite space.

Toggle PRM uses a novel connection strategy shown in Algorithm 2 that attempts to connect unconnected nodes of a roadmap to new connected components. After a node merges into a connected component, connections are attempted to other connected components. This repeats until a failure is reached, i.e., Toggle PRM learns about the opposite space. In higher dimensions, we have found that connecting into the roadmap (in order of closest to furthest roadmap nodes) until a failed connection is found reveals enough connectivity information in a space without oversampling. Eventually, this connectivity information could lead to discovering disconnections in the planning space.

3.1 α - ϵ -Separable Narrow Passages

In this section, we define a type of narrow passage that can be mapped well by Toggle PRM. Let $\infty(P)$ be a function which computes the hypervolume of a polytope P . Let \mathbb{C}_X be either \mathbb{C}_{free} or \mathbb{C}_{obst} and \mathbb{C}_Y be the complement of \mathbb{C}_X . Definitions 1, 2, 3,

and 4 describe necessary geometric objects for Theorem 1 and Corollaries 1, 2, and 3 to prove that Toggle PRM performs better than uniform random sampling in these objects. Finally, Definition 5 classifies an α - ε -separable narrow passage, and Proposition 1 proves improvement over uniform random sampling.

Definition 1. A *separable polytope* (Figure 2(a)) is a polytope P that is composed of three distinct polytopes; two outer polytopes $A_1, A_2 \subseteq \mathbb{C}_X$ and one inner polytope $B \subseteq \mathbb{C}_Y$ such that any line segment between points $a_1 \in A_1$ and $a_2 \in A_2$ lies completely in P and crosses B .

Definition 2. Let $\varepsilon \in [0, 1)$ be arbitrarily close to 0. A separable polytope $S = \{A_1, B, A_2\}$ is ε -*separable* (Figure 2(b)) if $\infty(B) \leq \varepsilon \infty(S)$.

Definition 3. Let $\alpha \in (0, 1]$. An α -separable polytope P (Figure 2(c)) is a polytope composed of two distinct polytopes $A \subseteq \mathbb{C}_X$ and $B \subseteq \mathbb{C}_Y$ such that there exists a separable polytope $S \subseteq P$, such that $\infty(S) \geq \alpha \infty(P)$.

Definition 4. An α -separable polytope P is α - ε -*separable* (Figure 2(d)) if the separating polytope $S \subseteq P$ is ε -separable.

These definitions describe various shapes which could be components of \mathbb{C}_{space} . The inner sections of the polytopes typically correspond to narrow passages and outer sections are components of \mathbb{C}_{obst} . Together, they lay the foundation for defining the types of narrow passages that can be mapped well with Toggle PRM.

Theorem 1. Given two sampling attempts, Toggle PRM will sample in the inner section B of a separable polytope $S = \{A_1, B, A_2\}$ with probability

$$P = \left(1 - \left(1 - \frac{\infty(B)}{\infty(\mathbb{C}_{space})} \right)^2 \right) + \binom{2}{1} \frac{\infty(A_1) \infty(A_2)}{\infty(\mathbb{C}_{space})^2}$$

Proof. For Toggle PRM to sample in B , either one of the two samples must lie in B (i.e., uniform sampling yields a sample), the first sample lies in A_1 and the second in A_2 , or the first lies in A_2 and the second in A_1 (i.e., the connection attempt yields a sample). The probability to uniformly sample in B given two sampling attempts is

$$P_{uniform} = \left(1 - \left(1 - \frac{\infty(B)}{\infty(\mathbb{C}_{space})} \right)^2 \right)$$

The probability to get one sample in A_1 and one in A_2 is

$$P_{local\ planner} = \binom{2}{1} \frac{\infty(A_1) \infty(A_2)}{\infty(\mathbb{C}_{space})^2}$$
□

Corollary 1. Given two sampling attempts, Toggle PRM will sample in section B of an ε -separable polytope $S = \{A_1, B, A_2\}$, with $\varepsilon = 0$, with probability

$$P = \binom{2}{1} \frac{\infty(A_1) \infty(A_2)}{\infty(\mathbb{C}_{space})^2}$$

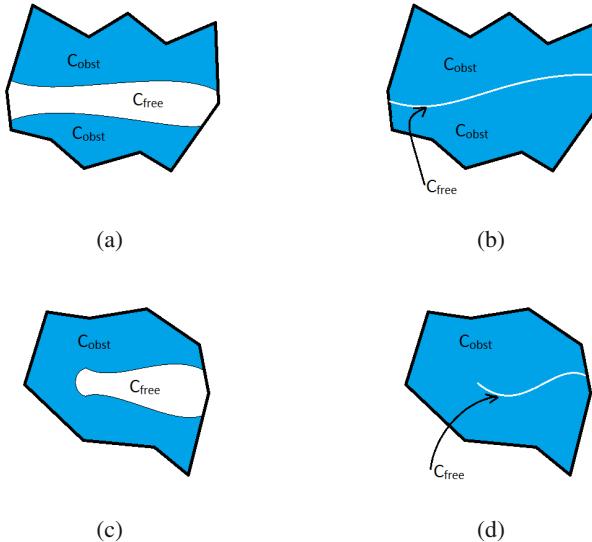


Fig. 2 Examples of various separable polytopes. (a) Separable polytope $S = \{A_1, A_2 \subset \mathbb{C}_{\text{obst}}, B \subseteq \mathbb{C}_{\text{free}}\}$. (b) ε -separable polytope $S = \{A_1, A_2 \subset \mathbb{C}_{\text{obst}}, B \subseteq \mathbb{C}_{\text{free}}\}$. (c) α -separable polytope $P = \{A \subseteq \mathbb{C}_{\text{obst}}, B \subseteq \mathbb{C}_{\text{free}}\}$. (d) α - ε -separable polytope $P = \{A \subseteq \mathbb{C}_{\text{obst}}, B \subseteq \mathbb{C}_{\text{free}}\}$.

Proof. If $\varepsilon = 0$ then $\infty(B) = 0$ and the probability to uniformly sample within B is 0. Thus, the probability from Theorem 1 collapses to

$$P = \binom{2}{1} \frac{\infty(A_1)\infty(A_2)}{\infty(\mathbb{C}_{\text{space}})^2}$$

□

Corollary 2. *Given two sampling attempts, Toggle PRM will sample in the inner section B of an α -separable polytope $Q = \{A_1, B, A_2\}$ with non-zero probability.*

Proof. Let $S = \{A_1, B, A_2\} \subseteq Q$ be the largest separable polytope within Q . Since the hypervolume of S is greater than an α ratio of the hypervolume of Q , Theorem 1 implies non-zero probability of sampling in B since α is strictly greater than 0. □

Corollary 3. *Given two sampling attempts, Toggle PRM will sample in the inner section B of an α - ε -separable polytope $Q = \{A_1, B, A_2\}$ with non-zero probability.*

Definition 5. A narrow passage N is α - ε -separable if there exists an α - ε -separable polytope $Q = \{A_1, B, A_2\}$, where $A \subseteq \mathbb{C}_{\text{obst}}$ and $B \equiv N$.

Proposition 1. *Given the same number of sampling attempts, Toggle PRM will sample in an α - ε -separable narrow passage with probability greater than that of uniform sampling.*

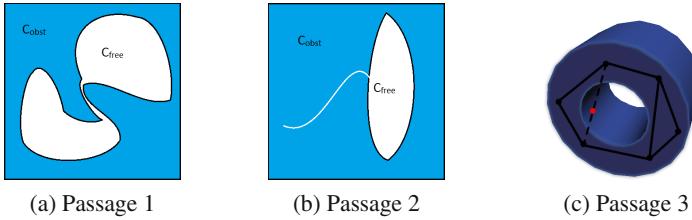


Fig. 3 Examples of α - ϵ -separable narrow passages (a and b) and non-separable narrow passages (c)

Proof. By definition, the narrow passage N is a component of a larger α - ϵ -separable polytope. By Theorem 1 and Corollaries 1, 2, and 3, Toggle PRM will sample in N with non-zero probability greater than that of uniform sampling. \square

It can be seen that the connection heuristic described in Algorithm 2 performs well in these narrow passages. For example, in a separable polytope $S = \{A_1, B, A_2\}$, if connected components develop in both A_1 and A_2 , then connections between them will generate samples in B . Additionally, there is a compounding effect in Toggle PRM that promotes sampling within narrow passages in both C_{free} and C_{obst} . For example, if the middle section of a separable polytope is in C_{obst} , then Toggle PRM may generate a sample in C_{free} and the connections from this sample to other samples in C_{free} will map the narrow region of C_{obst} .

Intuitively, α - ϵ -separable narrow passages almost separate the surrounding space. Essentially, if the narrow passage is a hyperplane of dimension $d - 1$, where d is the dimension of C_{space} , then Toggle PRM should be effective in mapping the narrow passage. Examples of α - ϵ -separable narrow passages are shown in Figure 3(a) and (b). However, Toggle PRM might have difficulty mapping problems such as a cylindrical C_{space} (3D shown in Figure 3(c)) with a hole through it (a 1D line), as it is not separable.

4 Experiments

In this section, we describe experiments conducted to test the effectiveness of Toggle PRM. First, we describe the experimental setup used in the various experiments. Then, we compare Toggle PRM to other common sampling-based methods in a variety of experiments ranging from varying narrow passage widths, surrounding obstacle volume widths, articulated linkages, and specific problems.

Experimental Setup. Toggle PRM, uniform random sampling, medial-axis sampling (MAPRM), Gaussian sampling, bridge-test sampling, and obstacle-based sampling (OBPRM) were implemented using the C++ motion planning library developed by the Parasol Lab at Texas A&M University. RAPID [8] or VCLIP [17] were used for collision detection computations. RAPID is used for higher DOF experiments, while VCLIP is used on the control experiments presented on simple

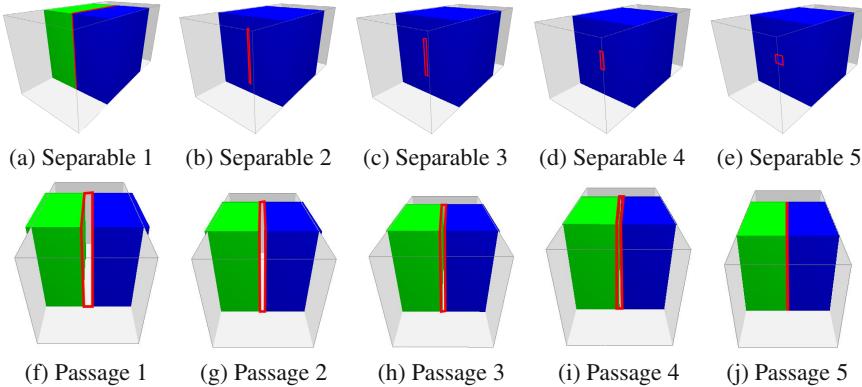


Fig. 4 Simple environments that vary narrow passage shape while keeping the volume constant (Separable 1-5) or that vary the passage volume while keeping the shape fixed (Passage 1-5). Narrow passages are highlighted (red).

environments. In these planners, connections are attempted between each node and its k -nearest neighbors according to a distance metric; here we use $k = 5$, \mathbb{C}_{space} Euclidean distance, and a simple straight-line local planner using a bisection evaluation strategy. The bisection evaluation strategy is commonly used because it usually identifies failures faster and hence reduces the amortized cost of failed connection attempts. We chose $k = 5$ to keep the computation cost of connections relatively low. During connections, only one failure connection is allowed per node, i.e., up to k connections will be attempted, but testing ceases after the first failure. This reduces overhead of connections for Toggle PRM, as well as allowing it to generalize to higher dimensions. Gaussian sampling is implemented as described in [3], with a Gaussian $d_{gauss} = \min(NP_{width}, Obst_{width})$, where NP_{width} is the narrow passage width and $Obst_{width}$ is the narrowest width of all surrounding obstacles. d_{gauss} is chosen to find samples which straddle the edges of \mathbb{C}_{obst} surrounding the narrow passage. Bridge-test sampling is implemented as described in [10], with $d_{bridge} = NP_{width} + d_{gauss}$ chosen so that samples can span the narrow passage. OBPRM is based on \mathbb{C}_{space} validity changes and implemented as in [1]. MAPRM is implemented using exact clearance information in low DOF cases and approximate clearance information in higher DOF cases as described in [15].

A few important metrics are used as a basis of comparison of the methods. Firstly, the number of free nodes sampled in the narrow passage is used to compare a sampler's ability to generate nodes in varying narrow passages. In problem solving tasks, the number of nodes in the roadmap of \mathbb{C}_{free} , number of CD calls (as a metric of efficiency of the planner), and the percentage of queries solved is compared. All results are averaged over 10 runs for all experiments, if a planner is unable to solve all problem instances, averages over only problems solved is reported.

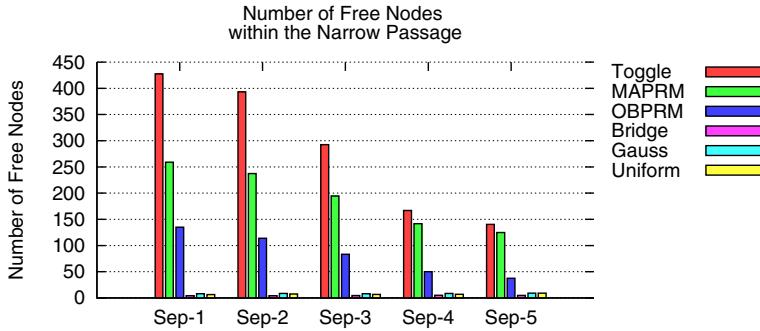


Fig. 5 The number of free nodes sampled in the narrow passages from 1000 attempts in the Separable environments for Toggle PRM (red), MAPRM (green), OBPRM (blue), bridge-test sampling (magenta), Gaussian sampling (cyan), and uniform sampling (yellow)

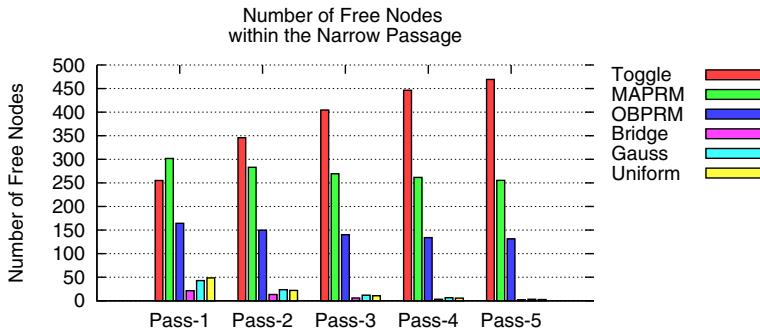


Fig. 6 The number of free nodes sampled in the narrow passages from 1000 attempts in the Passage environments for Toggle PRM (red), MAPRM (green), OBPRM (blue), bridge-test sampling (magenta), Gaussian sampling (cyan), and uniform sampling (yellow)

Controlled Narrow Passages. In this set of experiments, 1000 samples are attempted with each method (valid or invalid) in 10 simple environments that vary narrow passage volume or shape, using a point robot in 3-space.

The Separable environments (Separable 1-5 in Figure 4) have passages ranging from fully separable (Separable-1) to $\frac{1}{8}$ separable (Separable-5), by varying passage shape while keeping volume constant. The results in Figure 5 show that Toggle PRM outperforms the other methods in terms of successfully generating samples within the narrow passage. Additionally, in agreement with the analysis in Section 3.1, the effectiveness of Toggle PRM declines as the separability decreases. Indeed, this is true for all methods except for uniform, since its sampling density depends only on the volume.

The Passage environments (Passage 1-5 in Figure 4) vary the narrow passage volume from a width of 8 (Passage-1) to 0.5 (Passage-5), while keeping the shape fixed.

The results in Figure 6 show that as the passage becomes narrower (ε decreases), Toggle PRM actually generates more samples in the narrow region while all other methods declined in performance. The improved performance of Toggle PRM can be explained as follows. Recall that, unlike other methods, Toggle PRM samples in both \mathbb{C}_{free} and \mathbb{C}_{obst} , and that the total number of nodes it generates is roughly constant over the various environments. Hence, these results indicate that as a region becomes more difficult to map in one space (the passage becomes narrower), then the surrounding region is easier to map, and moreover, the coordinated mapping of it will facilitate the discovery of samples in the difficult space.

Toggle PRM for Articulated Linkages. To show how Toggle PRM generalizes to higher dimensions, we compare Toggle PRM on a series of articulated linkage problems. The environment obstacles stay the same in each query problem, in which a query must traverse the small hole, as shown in Figure 7(a). The free base linkages vary from 3 links (8 DOF) to 11 links (16 DOF), as seen in Figure 7(b-f). Planners stop when either the query is solved or 1000 nodes have been sampled in \mathbb{C}_{free} . However, not all planners were able to solve example queries every time, so the percentage of queries solved is shown in Table 1. Results for both the number of free nodes and the number of CD calls required to solve the problem are shown in Figure 8.

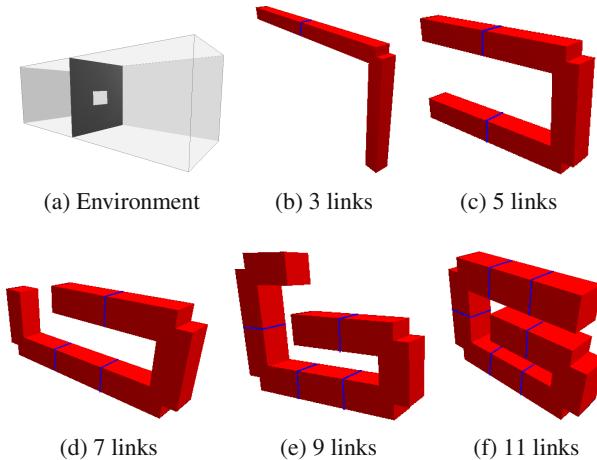
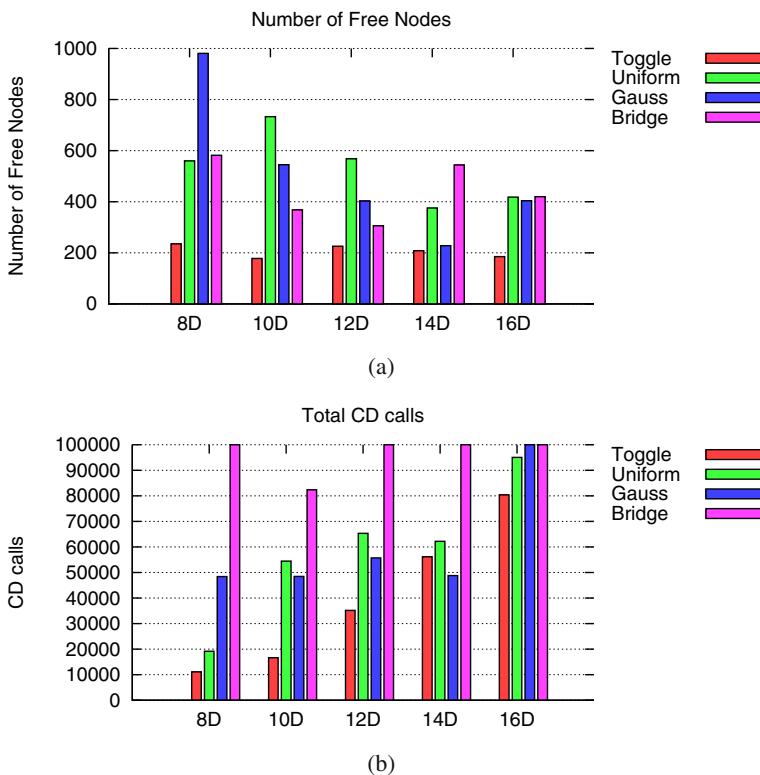


Fig. 7 Hook environment (a) along with 5 articulated linkage robots ranging from 3 links (b) to 11 links (f)

As shown in the results, Toggle PRM clearly solves more queries than the other planners. Toggle PRM typically can solve 100% of the queries, meaning its ability to solve problems efficiently, given a simple restriction such as was done in this experiment (limit of 1000 nodes), is more robust. Excluding the 14 DOF case where Gaussian sampling seems more efficient than Toggle PRM, Toggle PRM typically

Table 1 Percentage of successful planning attempts in Hook environments

| Linkage | Toggle | Uniform | Gauss | Bridge |
|---------|------------|---------|-------|--------|
| 8 DOF | 100 | 30 | 10 | 40 |
| 10 DOF | 100 | 50 | 60 | 40 |
| 12 DOF | 100 | 40 | 60 | 70 |
| 14 DOF | 100 | 60 | 40 | 80 |
| 16 DOF | 90 | 40 | 40 | 30 |

**Fig. 8** Metrics required to solve queries for the various Hook problems for Toggle PRM(red), uniform sampling (green), Gaussian sampling (blue), bridge-test sampling (magenta)

reduces the computational cost for solving such problems, even in higher DOFs. In the 14 DOF case however, Toggle PRM still outperforms Gaussian sampling in terms of nodes needed and percentage of instances solved. Many times however, Toggle PRM has half of the CD calls as the other comparable methods.

As we can see, Toggle PRM has increased sampling density within various narrow passages. Additionally, we have shown how Toggle PRM performs as the DOF increases, such as with articulated linkages. Next, we present this improvement for more general problems.

Problem Solving. In our last experiment, we emphasize the usefulness of Toggle PRM in a series of long tunnel environments, that a robot (3D rigid body - 6 DOF) must traverse. In these experiments, we limit a planner's roadmap nodes to 5000, success ratios are based upon this. The environments Maze-tunnel and Z-tunnel are shown in Figure 9. The results of these experiments are shown in Table 2.

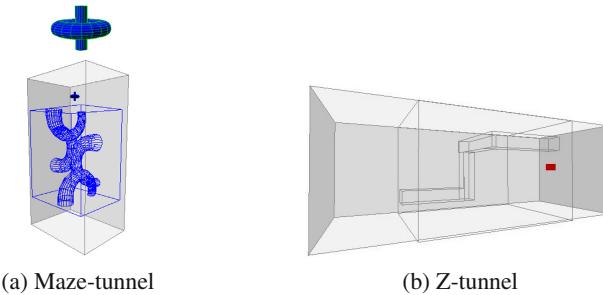


Fig. 9 Solution paths through these complicated environments must traverse the narrow passage

Table 2 Results from tunnel problems. Toggle PRM outperforms the other planners.

| Environment | Maze-tunnel | | | Z-tunnel | | | | |
|-------------------|-------------|------------------|------------|---------------|---------|------------------|------------|---------------|
| | Planner | % Queries Solved | Free Nodes | CD Calls | Planner | % Queries Solved | Free Nodes | CD Calls |
| Toggle PRM | | 100 | 420 | 1.37E5 | | 100 | 318 | 6.25E5 |
| Uniform | | 60 | 4355 | 1.62E5 | | 90 | 2500 | 6.27E5 |
| MAPRM | | 100 | 720 | 4.99E6 | | 100 | 361 | 9.07E5 |
| Gaussian | | 100 | 2436 | 8.34E5 | | 100 | 952 | 4.47E5 |
| Bridge | | 60 | 2666 | 5.2E6 | | 30 | 702 | 3.22E6 |
| OBPRM | | 100 | 749 | 2.27E6 | | 100 | 1441 | 8.1E5 |

As the results show, Toggle PRM is able to map \mathbb{C}_{free} more efficiently than the other methods, typically using fewer CD calls and fewer overall nodes for both environments. In the Z-tunnel, even though Toggle PRM does not reduce the number of collision detection calls as much as Gaussian sampling, Toggle PRM reduces the number of nodes required to solve the problem. Toggle PRM could be made more efficient in this scenario if we extend the blocked region past the bounds of the environment. Moreover, Gaussian sampling requires a definition of the d value, making tuning an issue. Regardless, these other methods have shown merit, and we believe combining the use of these samplers might bring out the best of all methods, as in using Gaussian sampling to feed samples to Toggle PRM.

5 Conclusion

In this paper, we presented a generalized Toggle PRM paradigm which extends naturally to higher dimensional problem spaces. Toggle PRM is a simple coordinated method for mapping both \mathbb{C}_{free} and \mathbb{C}_{obst} , with the added theoretical benefit of performing well for narrow passage problems. Additionally, we showed the generalized algorithm's benefit for a variety of benchmark problems. In the future, further extension of the theory behind the method should be taken to possibly show convergence rates, such as was done with PRMs. Further exploration into heuristics needs to be taken for Toggle PRM, for example, if medial-axis retractions are used in conjunction with the Toggle PRM paradigm.

Acknowledgements. This research supported in part by NSF awards CRI-0551685, CCF-0833199, CCF-0830753, IIS-096053, IIS-0917266 by THECB NHARP award 000512-0097-2009, by Chevron, IBM, by Award KUS-C1-016-04, made by King Abdullah University of Science and Technology (KAUST). Denny supported in part by an AFS Merit Fellowship, Texas A&M University.

References

1. Amato, N.M., Bayazit, O.B., Dale, L.K., Jones, C.V., Vallejo, D.: OBprm: An obstacle-based PRM for 3D workspaces. In: *Robotics: The Algorithmic Perspective*, pp. 155–168. A.K. Peters, Natick (1998); Proc. Third Workshop on Algorithmic Foundations of Robotics (WAFR), Houston, TX (1998)
2. Bohlin, R., Kavraki, L.E.: Path planning using Lazy PRM. In: Proc. IEEE Int. Conf. Robot. Autom. (ICRA), pp. 521–528 (2000)
3. Boor, V., Overmars, M.H., van der Stappen, A.F.: The Gaussian sampling strategy for probabilistic roadmap planners. In: Proc. IEEE Int. Conf. Robot. Autom. (ICRA), vol. 2, pp. 1018–1023 (1999)
4. Burns, B., Brock, O.: Sampling-based motion planning using predictive models. In: Proc. IEEE Int. Conf. Robot. Autom. (ICRA), pp. 3313–3318 (2005)
5. Burns, B., Brock, O.: Toward optimal configuration space sampling. In: Proc. Robotics: Sci. Sys. (RSS), pp. 105–112 (2005)
6. Denny, J., Amato, N.M.: Toggle PRM: Simultaneous mapping of C-free and C-obstacle - a study in 2D. In: Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS), San Francisco, California, USA, pp. 2632–2639 (2011)
7. Denny, J., Amato, N.M.: The toggle local planner for sampling-based motion planning. In: Proc. IEEE Int. Conf. Robot. Autom. (ICRA), St. Paul, Minnesota, USA, pp. 1779–1786 (2012)
8. Gottschalk, S., Lin, M.C., Manocha, D.: OBB-tree: A hierarchical structure for rapid interference detection. *Comput. Graph.* 30, 171–180 (1996); Proc. SIGGRAPH 1996
9. Yeh, H.-Y.(C.), Shawna Thomas, D.E., Amato, N.M.: Uobprm: A uniformly distributed obstacle-based prm. In: Proc. IEEE Int. Conf. Intel. Rob. Syst (IROS), Vilamoura, Algarve, Portugal (2012)
10. Hsu, D., Jiang, T., Reif, J., Sun, Z.: Bridge test for sampling narrow passages with probabilistic roadmap planners. In: Proc. IEEE Int. Conf. Robot. Autom. (ICRA), pp. 4420–4426 (2003)

11. Hsu, D., Kavraki, L.E., Latombe, J.C., Motwani, R., Sorkin, S.: On finding narrow passages with probabilistic roadmap planners. In: Proc. Int. Workshop on Algorithmic Foundations of Robotics (WAFR), pp. 141–153 (1998)
12. Kavraki, L.E., Švestka, 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. LaValle, S.M., Kuffner, J.J.: Randomized kinodynamic planning. In: Proc. IEEE Int. Conf. Robot. Autom. (ICRA), pp. 473–479 (1999)
14. Lien, J.M., Bayazit, O.B., Sowell, R.T., Rodriguez, S., Amato, N.M.: Shepherding behaviors. In: Proc. IEEE Int. Conf. Robot. Autom. (ICRA), pp. 4159–4164 (2004)
15. Lien, J.M., Thomas, S.L., Amato, N.M.: A general framework for sampling on the medial axis of the free space. In: Proc. IEEE Int. Conf. Robot. Autom. (ICRA), pp. 4439–4444 (2003)
16. Lozano-Pérez, T., Wesley, M.A.: An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM* 22(10), 560–570 (1979)
17. Mirtich, B.: V-clip: Fast and robust polyhedral collision detection. *ACM Transaction on Graphics* 17(3), 177–208 (1996)
18. Morales, M., Tapia, L., Pearce, R., Rodriguez, S., Amato, N.M.: A Machine Learning Approach for Feature-Sensitive Motion Planning. In: Erdmann, M., Overmars, M., Hsu, D., der Stappen, F. (eds.) *Algorithmic Foundations of Robotics VI*. STAR, vol. 17, pp. 361–376. Springer, Heidelberg (2005)
19. Morales, M., Pearce, R., Amato, N.M.: Metrics for analyzing the evolution of C-Space models. In: Proc. IEEE Int. Conf. Robot. Autom. (ICRA), pp. 1268–1273 (2006)
20. Morales, M., Tapia, L., Pearce, R., Rodriguez, S., Amato, N.M.: C-space subdivision and integration in feature-sensitive motion planning. In: Proc. IEEE Int. Conf. Robot. Autom. (ICRA), pp. 3114–3119 (2005)
21. Nielsen, C.L., Kavraki, L.E.: A two level fuzzy PRM for manipulation planning. In: Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS), pp. 1716–1722 (2000)
22. Pearce, R., Morales, M., Amato, N.M.: Structural improvement filtering strategy for prm. In: Proc. Robotics: Sci. Sys. (RSS) (2008)
23. Reif, J.H.: Complexity of the mover's problem and generalizations. In: Proc. IEEE Symp. Foundations of Computer Science (FOCS), San Juan, Puerto Rico, pp. 421–427 (1979)
24. Rodriguez, S., Thomas, S., Pearce, R., Amato, N.M. (RESAMPL): A Region-Sensitive Adaptive Motion Planner. In: Akella, S., Amato, N.M., Huang, W.H., Mishra, B. (eds.) *Algorithmic Foundation of Robotics VII*. STAR, vol. 47, pp. 285–300. Springer, Heidelberg (2008)
25. Singh, A.P., Latombe, J.C., Brutlag, D.L.: A motion planning approach to flexible ligand binding. In: Int. Conf. on Intelligent Systems for Molecular Biology (ISMB), pp. 252–261 (1999)
26. Wilmarth, S.A., Amato, N.M., Stiller, P.F.: MAPRM: A probabilistic roadmap planner with sampling on the medial axis of the free space. In: Proc. IEEE Int. Conf. Robot. Autom. (ICRA), vol. 2, pp. 1024–1031 (1999)
27. Wong, S.W.H., Jenkin, M.: Exploiting collision information in probabilistic roadmap planning. In: Int. Conf. on Mechatronics (2009)

On the Power of Manifold Samples in Exploring Configuration Spaces and the Dimensionality of Narrow Passages^{*}

Oren Salzman, Michael Hemmer, and Dan Halperin

Abstract. We extend our study of Motion Planning via Manifold Samples (MMS), a general algorithmic framework that combines geometric methods for the exact and complete analysis of low-dimensional configuration spaces with sampling-based approaches that are appropriate for higher dimensions. The framework explores the configuration space by taking samples that are *low-dimensional manifolds of the configuration space* capturing its connectivity much better than isolated point samples. The contributions of this paper are as follows: (i) We present a recursive application of MMS in a six-dimensional configuration space, enabling the coordination of two polygonal robots translating and rotating amidst polygonal obstacles. In the adduced experiments for the more demanding test cases MMS clearly outperforms PRM, with over 20-fold speedup in a *coordination-tight* setting. (ii) A probabilistic completeness proof for the most prevalent case, namely MMS with samples that are affine subspaces. (iii) A closer examination of the test cases reveals that MMS has, in comparison to standard sampling-based algorithms, a significant advantage in scenarios containing *high-dimensional narrow passages*. This provokes a novel characterization of narrow passages which attempts to capture their dimensionality, an attribute that had been (to a large extent) unattended in previous definitions.

Oren Salzman · Michael Hemmer · Dan Halperin
Tel-Aviv University,
Tel Aviv 69978, Israel
e-mail: {orenzalz,danha}@post.tau.ac.il,
mhsaar@googlemail.com

* This work has been supported in part by the 7th Framework Programme for Research of the European Commission, under FET-Open grant number 255827 (CGL—Computational Geometry Learning), by the Israel Science Foundation (grant no. 1102/11), and by the Hermann Minkowski–Minerva Center for Geometry at Tel Aviv University.

1 Introduction

Configuration spaces, or C-spaces, are fundamental tools for studying a large variety of systems. A point in a d -dimensional C-space describes one state (or configuration) of a system governed by d parameters. C-spaces appear in diverse domains such as graphical animation, surgical planning, computational biology and computer games. For a general overview of the subject and its applications see [9, 23, 25]. The most typical and prevalent example are C-spaces describing mobile systems (“robots”) with d degrees of freedom (*dofs*) moving in some *workspace* amongst obstacles. As every point in the configuration space \mathcal{C} corresponds to a *free* or *forbidden* pose of the robot, \mathcal{C} decomposes into disjoint sets $\mathcal{C}_{\text{free}}$ and $\mathcal{C}_{\text{forb}}$, respectively. Thus, the motion-planning problem is commonly reduced to the problem of finding a path that is fully contained within $\mathcal{C}_{\text{free}}$.

1.1 Background

C-spaces have been intensively studied for over three decades. Fundamentally, two major approaches exist:

(i) Analytic Solutions: The theoretical foundations, such as the introduction of C-spaces [27] and the understanding that constructing a C-space is computationally hard with respect to the number of *dofs* [29], were already laid in the late 1970’s and early 1980’s in the context of motion planning. Exact analytic solutions to the general motion planning problem as well as for various low-dimensional instances have been proposed in [4, 7, 8, 32] and [2, 3, 15, 27, 31], respectively. For a survey of related approaches see [33]. However, only recent advances in applied aspects of computational geometry made robust implementations for important building blocks available. For instance, Minkowski sums, which allow the representation of the C-space of a translating robot, have robust and exact two- and three-dimensional implementations [13, 14, 36]. Likewise, implementations of planar arrangements¹ for curves [35, C.30], could be used as essential components in [32].

(ii) Sampling-Based Approaches: Sampling-based approaches, such as Probabilistic Roadmaps (PRM) [20], Expansive Space Trees (EST) [17] and Rapidly-exploring Random Trees (RRT) [24], as well as their many variants, aim to capture the connectivity of $\mathcal{C}_{\text{free}}$ in a graph data structure, via random sampling of configurations. For a general survey on the field see [9, 25]. As opposed to analytic solutions these approaches are also applicable to problems with a large number of *dof*. Importantly, the PRM and RRT algorithms were shown to be probabilistically complete [18, 21, 22], that is, they are

¹ A subdivision of the plane into zero-dimensional, one-dimensional and two-dimensional cells, called vertices, edges and faces, respectively induced by the curves.

guaranteed to find a valid solution, if one exists. However, the required running time for finding such a solution cannot be computed for new queries at run-time. This is especially problematic as these algorithms suffer from high sensitivity to the so-called “narrow passage” problem, e.g., where the robot is required to move in environments cluttered with obstacles, having low clearance.

Though there are also some hybrid approaches [11, 16, 26, 37] it is apparent that the arsenal of currently available motion-planning algorithms lacks a general scheme applicable to high-dimensional problems with little or low sensitivity to narrow passages. In [30] we introduced a framework for Motion Planning via Manifold Samples (MMS), which should also be considered as a hybrid approach. In a setting considering a three-dimensional C-space it was capable of achieving twenty-fold (and more) speedup factor in running time when compared to the PRM algorithm when used for planning paths within narrow passages. We believe that the speedup presented in [30] does not present a mere algorithmic advantage for a specific implemented instance but a fundamental advantage of the framework when solving scenarios with narrow passages.

This study attempts to continue developing the MMS framework as a tool to overcome the gap mentioned in existing motion-planning algorithms. We briefly present the scheme and continue to a preliminary discussion on applying MMS in high-dimensional C-spaces, which motivates this paper.

1.2 Motion Planning via Manifold Samples

The framework is presented as a means to explore the *entire* C-space, or, in motion-planning terminology as a multi-query planner, consisting of a pre-processing stage and a query stage. The preprocessing stage constructs the *connectivity graph* \mathcal{G} of \mathcal{C} , a data structure that captures the connectivity of \mathcal{C} using *low-dimensional manifolds* as samples. The manifolds are decomposed into cells in $\mathcal{C}_{\text{free}}$ and $\mathcal{C}_{\text{forb}}$ in an analytic manner; we call a cell of the decomposed manifold that lies in $\mathcal{C}_{\text{free}}$ a *free space cell* (FSC). The FSCs serve as nodes in \mathcal{G} . Two nodes are connected by an edge if their corresponding FSCs intersect. See Fig. 1 for an illustration.

Once \mathcal{G} has been constructed it can be queried for paths between two configurations $q_s, q_t \in \mathcal{C}_{\text{free}}$ in the following manner: A manifold that contains q_s in one of its FSCs is generated and decomposed (similarly for q_t). These FSCs and their appropriate edges are added to \mathcal{G} . We compute a path γ in \mathcal{G} between the FSCs that contain q_s and q_t . If such a path is found in \mathcal{G} , it can be (rather straightforwardly) transformed into a continuous path in $\mathcal{C}_{\text{free}}$ by planning a path within each FSC in γ .

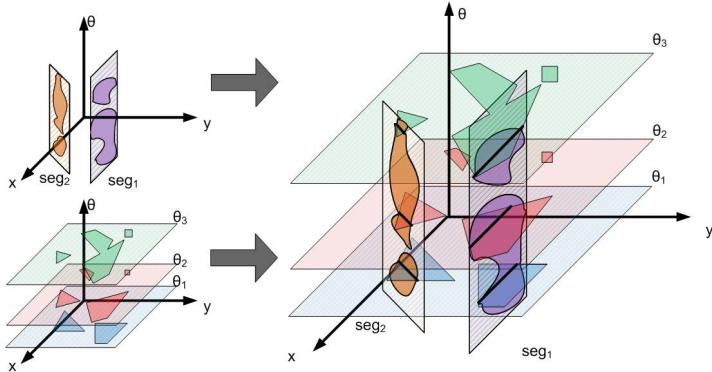


Fig. 1 Three-dimensional C-spaces: The left side illustrates two families of manifolds where the decomposed cells are darkly shaded. The right side illustrates their intersection that induces the graph \mathcal{G} . Figure taken from [30].

1.3 MMS in Higher Dimensions

The successful application of MMS in [30] to a three-dimensional C-space can be misleading when we come to apply it to higher dimensions. The heart of the scheme is the choice of manifolds from which we sample. Informally, for the scheme to work we must require that the used set of manifolds \mathcal{M} fulfills the following conditions.

- C1.** The manifolds in \mathcal{M} cover the C-space.
- C2.** A pair of surfaces chosen uniformly and independently² at random from \mathcal{M} intersect with significant probability.
- C3.** Manifolds need to be of very low dimension as MMS requires an analytic description of the C-space when restricted to a manifold. Otherwise the machinery for a construction of this description is not readily available.

For MMS to work in C-spaces of dimension d , Condition **C2** has a prerequisite that the sum of dimensions of a pair of manifolds chosen uniformly and independently at random from \mathcal{M} is at least d with significant probability. This means in particular that \mathcal{M} will consist of manifolds of dimension³ $\lceil \frac{d}{2} \rceil$. With this prerequisite in mind, there is already much to gain from using our existing and strong machinery for analyzing two-dimensional manifolds [5, 6, 12], while fulfilling the conditions above: We can solve motion-planning problems with four degrees of freedom, at the strength level that MMS offers, which is higher than that of standard sampling-based tools.

² The requirement that the choices are independent stems from the way we prove completeness of the method. It is not necessarily an essential component of the method itself.

³ The precise statement is somewhat more involved and does not contribute much to the informal discussion here. Roughly, \mathcal{M} should comprise manifolds of dimension $\lceil \frac{d}{2} \rceil$ or higher and possibly manifolds of their co-dimension.

However, we wish to advance to higher-dimensional C-spaces in which satisfying all the above conditions at once is in general impossible. We next discuss two possible relaxations of the conditions above that can lead to effective extensions of MMS to higher dimensions.

Dependent Choice of Manifolds: If we insist on using only very low-dimensional manifolds even in higher-dimensional C-spaces, then to guarantee that pairs of manifolds intersect, we need to impose some dependence between the choices of manifolds, i.e., relaxing condition **C2**. A natural way to impose intersections between manifolds is to adapt the framework of tree-based planners like RRT [24]. When we add a new manifold, we insist that it connects either directly or by a sequence of manifolds to the set of manifolds collected in the data structure (tree in the case of RRT) so far.

Approximating Manifolds of High Dimension: As we do not have the machinery to exactly analyze C-spaces restricted to manifolds of dimension three or higher, we suggest to substitute exact decomposition of the manifolds as induced by the C-space by some approximation. i.e., relaxing condition **C3**. There are various ways to carefully approximate C-spaces. In the rest of the paper we take the approach of *a recursive application of MMS*.

In Section 2 we demonstrate this recursive application for a specific problem in a six-dimensional configuration space, namely the coordination of two planar polygonal robots translating and rotating amidst polygonal obstacles. In the adduced experiments for the more demanding test cases MMS clearly outperforms PRM, with over 20-fold speedup in an especially tight setting. Section 3 provides the theoretical foundations for using MMS in a recursive fashion. In Section 4 we examine the significant advantage of MMS with respect to prevailing sampling-based approaches in scenarios containing *high-dimensional narrow passages*. This provokes a novel characterization of narrow passages which attempts to capture their dimensionality. We conclude with an outlook on further work in Section 5.

2 The Case of Two Rigid Polygonal Robots

We discuss the MMS framework applied to the case of coordinating the motion of two polygonal robots R_a and R_b translating and rotating in the plane amidst polygonal obstacles. Each robot is described by the position of its reference point $r_a, r_b \in \mathbb{R}^2$ and the amount of counter-clockwise rotation θ_a, θ_b with respect to an initial orientation. All placements of R_a in the workspace \mathcal{W} induce the three-dimensional space $\mathcal{C}^a = \mathbb{R}^2 \times S^1$. Similarly for R_b . We describe the full system by the six-dimensional C-space $\mathcal{C} = \mathcal{C}^a \times \mathcal{C}^b$.

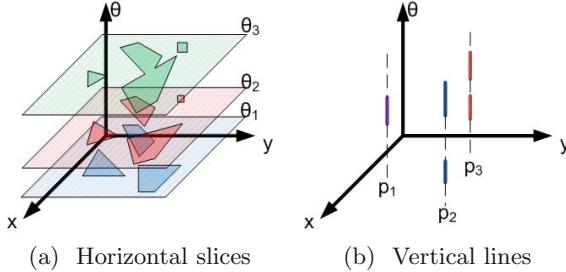


Fig. 2 Manifolds families and their FSCs. FSCs of horizontal slices are polygons while FSCs of vertical lines are intervals along the line.

2.1 Recursive Application of the MMS Framework

We first assume (falsely) that we have the means to decompose three-dimensional manifolds. Under this assumption, the application of MMS is straightforward: The set \mathcal{M} consists of two families. An element of the first family of manifolds is defined by fixing R_b at free configurations $b \in \mathcal{C}_{\text{free}}^b$ while R_a moves freely inducing the three-dimensional subspaces⁴ $\mathcal{C}^a \times b$. The second family is defined symmetrically by fixing R_a . As subspace pairs of the form $(a \times C^b, C^a \times b)$ intersect at the point (a, b) , manifolds of the two families intersect allowing for connections in the connectivity graph \mathcal{G} .

However, we do not have the tools to construct three-dimensional manifolds explicitly. Thus the principal idea is to construct *approximations* of these manifolds by another application of MMS. Since for a certain manifold one robot is fixed, we are left with a three-dimensional C-space in which the fixed robot is regarded as an obstacle. Essentially this is done by using the implementation presented in [30] but with a simpler set of manifolds (see also Fig. 2): (i) **Horizontal slices** – corresponding to a fixed orientation of the moving robot while it is free to translate (ii) **Vertical lines** – corresponding to a fixed location of the moving robot while it is free to rotate.

Since we only approximate the three dimensional subspaces we have to make sure that they still intersect. In other words the sampled positions of a robot must be covered by the approximation of its subspace. To do so we sample an initial set of angles Θ_a that is used for the first robot throughout the entire algorithm. When approximating its subspace (the second robot is fixed) we take a horizontal slice for each angle in Θ_a . Conversely, we only fix the robots position at angles in Θ_a . We do the same for the second robot and a set Θ_b . This way it is ensured that even the approximations of the three dimensional subspaces intersect.

⁴ In this paper, when discussing subspaces, we should actually use the term affine subspaces or linear manifolds. We allow ourselves this (slight) inaccuracy for ease of reading.

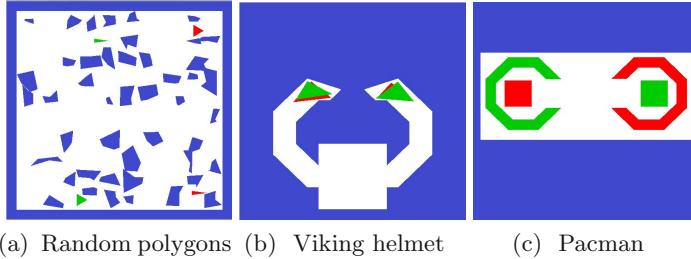


Fig. 3 Experimental scenarios. Source and target configurations are drawn in green and red, respectively.

2.2 Implementation Details

Horizontal Slices: Let R_m and R_f denote the moving and fixed robot, respectively. Θ_m denotes the set of angles that is sampled for R_m . A horizontal plane for an angle $\theta_m \in \Theta_m$ is defined by the Minkowski sum of $-R_{\theta_m}$ with all obstacles and, in addition, with the fixed robot.⁵ However, for each approximation of a three-dimensional affine subspace of robot R_m we are using the same set of angles, namely Θ_m . Only the position of robot R_f changes. Therefore, for all $\theta_m \in \Theta_m$ we precompute the Minkowski sum of robot $-R_{\theta_m}$ with all obstacles. In order to obtain a concrete slice we only add the Minkowski sum of $-R_{\theta_m}$ with R_f by a simple overlay.

Vertical Lines: Fixing the reference point of R^m to some location while it is free to rotate induces a vertical line in the three-dimensional C-space. Each vertex (or edge) of the robot in combination with each edge (or vertex) of an obstacle (or the fixed robot) gives rise to up to two critical angles on this line. These critical values mark a potential transition between $\mathcal{C}_{\text{forb}}$ and $\mathcal{C}_{\text{free}}$. Thus a vertical line is constructed by computing these critical angles and the FSCs are maximal free intervals along this line (for further details see [1]).

2.3 Experimental Results

We demonstrate the performance of our planner using three different scenarios. All scenarios consist of a workspace, obstacles, two robots and one query (source and target configurations). All reported tests were measured on a Dell 1440 with one 2.4GHz P8600 Intel Core 2 Duo CPU processor and 3GB of memory running with a Windows 7 32-bit OS. Preprocessing times presented are times that yielded at least 80% (minimum of 5 runs) success rate in solving queries. The algorithm is implemented in C++ based on CGAL [35],

⁵ $-R_{\theta_m}$ denotes R_m rotated by θ_m and reflected about the origin.

which is used for the geometric primitives, and the BOOST graph library [34], which is used to represent the connectivity graph \mathcal{G} .

Fig. 3 illustrates the scenarios where the obstacles are drawn in blue and the source and target configurations are drawn in green and red, respectively. We used an implementation of the PRM algorithm as provided by the OOPSMP package [28]. For fair comparison, we did not use cycles in the roadmap as cycles increase the preprocessing time significantly. We manually optimized the parameters of each planner over a concrete set. The parameters for MMS are: n_θ – the number of sampled angles; n_ℓ – the number of vertical lines; n_f – the number of times some robot is fixed to a certain configuration while the three-dimensional C-space of the other is computed. The parameters used for the PRM are: k – the number of neighbors to which each milestone should be connected; %st – the percentage of time used to sample new milestones. The results are summarized in Table 1.

Table 1 Comparison With PRM

| Scenario | MMS | | | | PRM | | | Speedup |
|-----------------|------------|----------|-------|--------|-----|--------|--------|---------|
| | n_θ | n_ℓ | n_f | t[sec] | k | % st | t[sec] | |
| Random polygons | 5 | 512 | 2 | 8 | 20 | 0.025 | 8 | 1 |
| Viking helmet | 20 | 16 | 10 | 6.2 | 14 | 0.0125 | 40 | 6.45 |
| Pacman | 5 | 4 | 180 | 17.6 | 20 | 0.0125 | 20 | 1.14 |

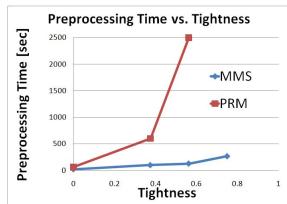


Fig. 4 Tightness Results

The Random polygons scenario⁶ is an easy scenario where little coordination is required. Both planners require the same amount of time to solve this case. We see that even though our planner uses complex primitives, when using the right parameters, it can handle simple cases with no overhead when compared to the PRM algorithm.

The Viking-helmet scenario consists of two narrow passages that each robot needs to pass through. Moreover, coordination is required for the two robots to exchange places in the lower chamber. We see that the running times of the MMS are favorable when compared to the PRM implementation. Note that although each robot is required to move along a narrow passage, the motion along this passage does not require coordination between the robots.

The Pacman scenario, in which the two robots need to exchange places, requires coordination of the robots: they are required to move into a position where the C-shaped robot, or pacman, “swallows” the square robot, the pacman is then required to rotate around the robot. Finally the two robots should move apart (see Fig. 5). We ran this scenario several times, progressively increasing the square robot size. This caused a “tightening” of the passages containing the desired path. Fig. 4 demonstrates the preprocessing time as

⁶ A scenario provided as part of the OOPSMP distribution.

a function of the tightness of the problem for both planners. A tightness of zero denotes the base scenario (Fig. 3c) while a tightness of one denotes the tightest solvable case. Our algorithm has less sensitivity to the tightness of the problem as opposed to the PRM algorithm. In the tightest experiment solved by the PRM, MMS runs 20 times faster. We ran the experiment on a tighter scenario but the PRM algorithm terminated after 5000 seconds due to lack of memory resources. We believe that behavior of the algorithms with respect to the tightness of the passage is a fundamental difference between the two algorithms and discuss its origin in Section 4.

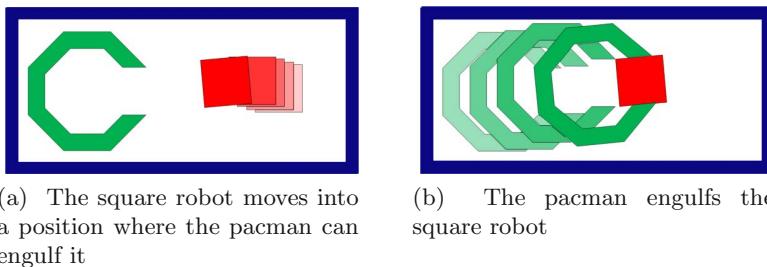


Fig. 5 Example of a path in the Pacman Scenario

3 Probabilistic Completeness of MMS

It has been shown that PRM, using point samples, is probabilistically complete (see, e.g., [9, C.7]). At first glance it may seem that if the scheme is complete for point samples then it is evidently complete when these samples are substituted with manifold samples: manifolds of dimension one or higher guarantee better coverage of the configuration space. However, there is a crucial difference between PRM and MMS when it comes to connectivity. The completeness proof for PRM relies, among others facts, on the fact that if the straight line segment in the configuration space connecting two nearby samples lies in the free space, then the nodes corresponding to these two configurations are connected by an edge in the roadmap graph. The connectivity in MMS is attained through intersections of manifolds, which may require a chain of subpaths on several distinct manifolds to connect two nearby free configurations. This is what makes the completeness proof for MMS non-trivial and is expressed in Lemma 3.2 below.

We present a probabilistic completeness proof for the MMS framework for the case where the configuration space \mathcal{C} is the d -dimensional Euclidean space \mathbb{R}^d while MMS is taking samples from two perpendicular affine subspaces, the sum of whose dimension is d . Assuming Euclidean space does not impose a real restriction as long as the actual C-space can be embedded into a Euclidean space. Also in more complex cases such as periodic parameters this only requires some minor technical modifications.

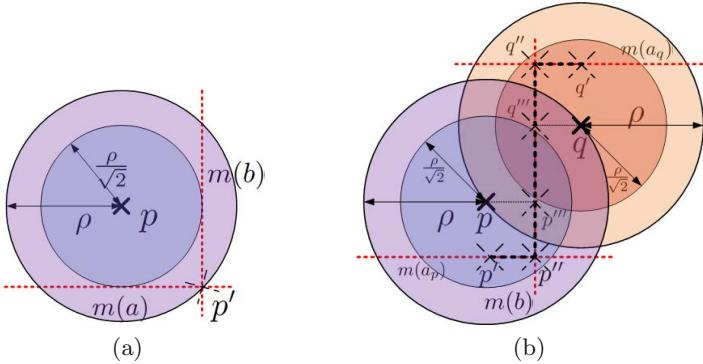


Fig. 6 Two-dimensional sketch: balls and manifolds are presented as circles and lines, respectively. (a) Intersection of two $\rho/\sqrt{2}$ -intersecting manifolds. (b) Construction of a path as defined in Lemma 3.2.

Let A and B denote such subspaces and let k and $d - k$ be their dimensions, respectively. As \mathcal{C} is decomposed into two perpendicular subspaces, a point $p = (a_1, \dots, a_k, b_1, \dots, b_{d-k}) \in \mathcal{C}$ may be represented as the pair of points (a, b) from subspaces A and B . Under this assumption, the set of manifolds \mathcal{M} consists of two families of $(d - k)$ and k -dimensional manifolds \mathcal{M}^B and \mathcal{M}^A . Family \mathcal{M}^A consists of all manifolds that are defined by fixing a point $a_0 \in A$ while the remaining $d - k$ parameters are variable, \mathcal{M}^B is defined symmetrically. Two manifolds $m(a) \in \mathcal{M}^A$ and $m(b) \in \mathcal{M}^B$ always intersect in exactly one point, i.e., $m(a) \cap m(b) = (a, b) \in \mathcal{C}$. Let $B_r^{\mathcal{C}}(p) = \{q \in \mathcal{C} \mid \text{dist}(p, q) \leq r\}$ define a ball in \mathcal{C} of radius r centered at $p \in \mathcal{C}$, where dist denotes the Euclidean metric on \mathcal{C} . Likewise, $B_r^B(b)$ and $B_r^A(a)$ denote $(d - k)$ and k -dimensional balls in B and A , respectively.

Definition 1 (ρ -intersecting). For $\rho > 0$ we term a manifold $m(a) \in \mathcal{M}^A$ ρ -intersecting for a point $p \in \mathcal{C}$ if $m(a) \cap B_{\rho}^{\mathcal{C}}(p) \neq \emptyset$, i.e., if $a \in B_{\rho}^A(p_A)$, where p_A is the projection of p into A . Similarly for manifolds in B .

A feasible path γ is a continuous mapping from the interval $[0, 1]$ into $\mathcal{C}_{\text{free}}$. The image of a path is defined as $\text{Im}(\gamma) = \{\gamma(\alpha) \mid \alpha \in [0, 1]\}$. We show that for any collision-free path $\gamma_{p,q}$ of clearance $\rho > 0$ between two configurations p and q , the probability that MMS constructs a path from p to q with distance at most ρ from $\gamma_{p,q}$ on the union of the sampled manifolds is positive. Moreover, the probability of failing to find such a path by the MMS algorithm decreases exponentially with the number of samples.

Lemma 3.1 For $p \in \mathcal{C}$ and $\rho > 0$ let $m(a) \in \mathcal{M}^A$ and $m(b) \in \mathcal{M}^B$ be two manifolds that are $\rho/\sqrt{2}$ -intersecting. Their intersection point $p' = (a, b) = m(a) \cap m(b)$ is in $B_{\rho}^{\mathcal{C}}(p)$.

For an illustration of Lemma 3.1 see Figure 6a. The proof (given in [1]) follows immediately from elementary properties of \mathbb{R}^d . Lemma 3.2 is more

important as it establishes connections. Informally, it shows that for any two points p and q , a manifold $m(b) \in \mathcal{M}^B$ that is close to both points enables a connection between two manifolds $m(a_p), m(a_q) \in \mathcal{M}^A$ that are close to p and q , respectively.

Lemma 3.2 *Let $p, q \in C$ be two points such that $\text{dist}(p, q) \leq \rho$ and let $m(a_p), m(a_q) \in \mathcal{M}^A$ be two $\rho/\sqrt{2}$ -intersecting manifolds for p and q respectively. Let $m(b) \in \mathcal{M}^B$ be a manifold that is simultaneously $\rho/\sqrt{2}$ -intersecting for p and q and let $p' = (a_p, p_B) \in B_\rho^C(p)$ and $q' = (a_q, q_B) \in B_\rho^C(q)$ be the projection of p and q on $m(a_p)$ and $m(a_q)$, respectively.*

There exists a path $\gamma_{p',q'}$ between p' and q' such that $\text{Im}(\gamma_{p',q'}) \subseteq (B_\rho^C(p) \cup B_\rho^C(q)) \cap (m(a_p) \cup m(b) \cup m(a_q))$, i.e. there is a path lying on the manifolds within the union of the balls.

Proof. Let $p'' = m(a_p) \cap m(b) = (a_p, b)$ and $q'' = m(a_q) \cap m(b) = (a_q, b)$ denote the intersection point of $m(a_p)$ and $m(a_q)$ with $m(b)$, respectively. Moreover, let $p''' = (p_A, b) \in B_\rho^C(p)$ and $q''' = (q_A, b) \in B_\rho^C(q)$ denote the projections of p and q on $m(b)$. We show that the path composed of the segments (p', p'') , (p'', p''') , (p''', q''') , (q''', q'') and (q'', q') fulfills the requirements. See Fig. 6b.

By Lemma 3.1 the intersection points p'' and q'' are inside $B_\rho^C(p)$ and $B_\rho^C(q)$, respectively. Thus, by convexity of each ball the segments $(p', p'') \subset m(q_p)$ and $(q', q'') \subset m(a_q)$ as well as the segments $(p'', p'''), (q'', q''') \subset m(b)$ are in $(B_\rho^C(p) \cup B_\rho^C(q))$.

It remains to show that $(p''', q''') \subset m(b)$ is inside $(B_\rho^C(p) \cup B_\rho^C(q))$. Recall that $\text{dist}(p, q) \leq \rho$ and therefore $\text{dist}(p''', q''') \leq \rho$. Let \bar{p} be a point on the segment (p''', q''') that, w.l.o.g, is closer to p''' . Thus $\text{dist}(\bar{p}, p''') \leq \rho/2$. The manifold $m(b)$ is $\rho/\sqrt{2}$ -intersecting, thus $\text{dist}(p, p''') \leq \rho/\sqrt{2}$. As the segments (p, p''') and (p''', \bar{p}) are perpendicular it holds:

$$\text{dist}(p, \bar{p}) = \sqrt{\text{dist}(p, p''')^2 + \text{dist}(p''', \bar{p})^2} \leq \sqrt{\rho^2/2 + \rho^2/4} < \rho.$$

□

Theorem 3.3 *Let $p, q \in \mathcal{C}_{\text{free}}$ such that there exists a collision-free path $\gamma_{p,q} \in \Gamma$ of length L and clearance ρ between p and q . Then the probability of the MMS algorithm to return a path between p and q after generating n_A and n_B manifolds from families \mathcal{M}^A and \mathcal{M}^B is:*

$$\begin{aligned} \Pr[(p, q)\text{SUCCESS}] &= 1 - \Pr[(p, q)\text{FAILURE}] \\ &\geq 1 - \left[\frac{L}{\rho} \right] [(1 - \mu_A)^{n_A} + (1 - \mu_B)^{n_B}], \end{aligned}$$

where μ_A and μ_B are some positive constants.

The constants μ_A and μ_B reflect the probability of a manifold to be $\rho/\sqrt{2}$ -intersecting for one or two nearby points, respectively; The proof for

Theorem 3.3 is rather technical. It involves using Lemma 3.2 repeatedly for points along the path $\gamma_{p,q}$ of distance less than ρ . We omit the details and refer the reader to [1] for the full proof.

Recursive Application. The proof of Theorem 3.3 assumes that the samples are taken using full high-dimensional manifolds. However, Section 2 demonstrates a recursive application of MMS where the approximate samples are generated by another application of MMS.

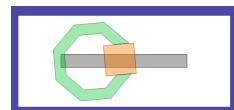
In order to obtain a completeness proof for the two-level scheme let γ be a path of clearance 2ρ . First, assume that the samples taken by the first level of MMS are exact. Applying Theorem 3.3 for γ and ρ shows that with sufficient probability MMS would find a set M' of manifolds that would contain a path γ' . Since we required clearance 2ρ but relied on the tighter clearance ρ , it is guaranteed that γ' still has clearance ρ . Now, each manifold $m' \in M'$ is actually only an approximation constructed by another application of MMS. Thus, for each $m' \in M'$ apply Theorem 3.3 to subpath $\gamma'_{m'} = \gamma' \cap m'$ which has clearance ρ . Concatenation of all the resulting subpaths concludes the argument. Of course the parameters in the inequality in Theorem 3.3 change accordingly.

We remark that the recursive approach imposes a mild restriction on the sampling scheme as the sampling and the approximation must be somewhat coordinated. Since in theory $m(a) \cap m(b) = (a, b)$ we must ensure that points that we sample from A are contained in every approximation of $m(b) \in \mathcal{M}^B$ and vice versa. In our implementation this is ensured by restricting the set of possible angles to those used to approximate $m(b) \in \mathcal{M}^B$ (see Section 2).

4 On the Dimension of Narrow Passages

Consider the pacman scenario illustrated in Fig. 3c of the experiments section. We obtain a narrow passage by increasing the size of the square-shaped robot making it harder for the pacman to swallow it. Fig. 4 shows that our approach is significantly less sensitive to this tightening than the PRM algorithm. In order to explain this, let us take a closer look at the nature of the narrow passage for the tightest solvable case.

In order to get from the start placement to the goal placement, the pacman must swallow the square, rotate around it and spit it out again. Due to symmetry it is sufficient to concentrate on the first part. The figure to the right depicts the tightest case, i.e., when the square robot fits exactly into the “mouth” of the pacman. The gray rectangle indicates the positions of the reference point of the square such that there is a valid movement of the pacman that will allow it to swallow the square robot (two-dimensional region, two parameters), the rotation angle of the square is also important (one additional parameter). The range of concurrently possible values for all three parameters is small



but does not tend to zero. The passage becomes narrow by the fact that the rotation angle of the pacman must correlate exactly with the orientation of the square to allow for passing through the mouth. Moreover, the set of valid placements for the reference point of the pacman while swallowing the square (other parameters being fixed) is a line, i.e., its x and y parameter values are coupled. Thus, the passage is a four-dimensional object as we have a tight coupling of two pairs of parameters in a six-dimensional C-space.

The PRM approach has difficulties to sample in this passage since the measure tends to zero as the size of the square increases. On the other hand, for our approach the passage is only tight with respect to the correlation of the two angles. As soon as the MMS samples an (approximated) volume that fixes the square robot such that the pacman *can* engulf it, the approximation of the volume just needs to include a horizontal slice of a suitable angle and the passage becomes evident in the corresponding Minkowski sum computation.

4.1 Definition of Narrow Passages

Intuition may suggest that narrow passages are tunnel-shaped. However, a one-dimensional tunnel in a high-dimensional C-spaces would correspond to a simultaneous coupling of *all* parameters, which is often not the case. For instance, the discussion of the pacman scenario shows that the passage is narrow but that it is still a four-dimensional volume, which proved to be a considerable advantage for our approach in the experiments. Though, some sampling based approaches try to take the dimension of a passage into account (e.g. see [10]) it seems that this aspect is not reflected by existing definitions that attempt to capture attributes of the C-space. Definitions such as ϵ -goodness [19] and expansiveness [17] are able to measure the size of a narrow passage better than the clearance [18] of a path, but neither incorporates the dimension of a narrow passage in a very accessible way. As a consequence, we would like to propose a new set of definitions that attempt to simultaneously grasp the tightness and the dimension of a passage.

We start by defining the “ordinary” clearance of a path. The characterization is based on the notion of homotopy classes of paths with respect to a set $\Gamma_{s,t}$, i.e., the set of all paths starting at s and ending at t . For a path $\gamma_0 \in \Gamma_{s,t}$ and its homotopy class $\mathcal{H}(\gamma_0)$ we define the clearance of the class as the largest clearance found among all paths in $\mathcal{H}(\gamma_0)$.

Definition 2. The **clearance** of a homotopy class $\mathcal{H}(\gamma_0)$ for $\gamma_0 \in \Gamma_{s,t}$ is

$$\sup_{\gamma \in \mathcal{H}(\gamma_0)} \{ \sup \{ \rho > 0 \mid B_\rho^d \oplus \text{Im}(\gamma) \subseteq \mathcal{C}_{\text{free}} \} \},$$

where \oplus denotes the Minkowski sum of two sets, which is the vector sum of the sets.

By using a d -dimensional ball this definition treats all directions equally, thus considering the passage of $\mathcal{H}(\gamma_0)$ to be a one-dimensional tunnel. We next

refine this definition by using a k -dimensional disk, which may be placed in different orientations depending on the position along the path.

Definition 3. For some integer $0 < k \leq d$ the **k -clearance** of $\mathcal{H}(\gamma_0)$ is:

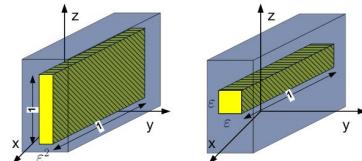
$$\sup_{\gamma \in \mathcal{H}(\gamma_0)} \{ \sup \{ \rho > 0 \mid \exists \mathbf{R} : [0, 1] \rightarrow \mathcal{R} \ \forall t \in [0, 1] : \gamma(t) \oplus \mathbf{R}(t) B_\rho^k \subseteq \mathcal{C}_{\text{free}} \} \},$$

where \mathcal{R} is the set of d -dimensional rotation matrices and B_ρ^k is the k -dimensional ball of radius ρ . In case the map \mathbf{R} is required to be continuous we talk about **continuous k -clearance**.

Clearly, the k -clearance of $\mathcal{H}(\gamma_0)$ for $k = d$ is simply the clearance of $\mathcal{H}(\gamma_0)$. For decreasing values of k , the k -clearance of a homotopy class is a monotonic increasing sequence. We next define the dimension of a passage using this sequence, that is, we set the dimension to be the first k for which the clearance becomes *significantly larger*⁷ than the original d -dimensional clearance.

Definition 4. A passage for $\mathcal{H}(\gamma_0)$ in \mathbb{R}^d of clearance ρ is called **k -dimensional** if k is the largest index such that k -clearance($\mathcal{H}(\gamma_0)$) $\gg \rho$. If for every k k -clearance($\mathcal{H}(\gamma_0)$) $\not\gg \rho$ then the passage is termed one-dimensional.

The figure on the right illustrates two three-dimensional C-spaces consisting of a narrow passage (yellow) surrounded by obstacles (blue). Both passages have a measure of ε^2 thus for a PRM like planner, sampling in either passage is equally hard as the probability of a uniform point sample to lie in either one of the narrow



passages is proportional to ε^2 . However, the two passages are fundamentally different. The passage depicted on the right-hand side is a one-dimensional tunnel corresponding to a tight coupling of the three parameters. The passage depicted on the left-hand side is a two-dimensional flume which is much easier to intersect by a probabilistic approach that uses manifolds as samples. Our new definitions formally reveal this difference. For k equals 3, 2 and 1 the k -clearance of the right passage is ε , $\sqrt{2}\varepsilon$ and larger than 1, respectively. For the left passage this sequence is ε^2 for $k = 3$ and larger than 1 for $k = 2, 1$ which characterizes the passage as two-dimensional.

4.2 Discussion

We believe that the definitions introduced in Section 4.1, can be an essential cornerstone for a formal proof that shows the advantage of manifold samples over point samples in the presence of high-dimensional narrow passages. We

⁷ We leave this notion informal as it might depend on the problem at hand.

sketch the argument briefly. Let $\mathcal{C}_{\text{free}}$ contain a narrow passage of dimension k , that is, the passage has clearance ρ and k -clearance λ , where $\lambda \gg \rho$. This implies that it is possible to place discs of dimension k and radius $\lambda \gg \rho$ into the tight passage. The main argument is that for a random linear manifold of dimension $d - k$ the probability to hit such a disc is proportional to λ , which is much larger than ρ . The probability also depends on the angle between the linear subspace containing the disc and the linear manifold. However, by choosing a proper set of manifold families it should be possible to guarantee the existence of at least one family for which the angle is bounded.

5 Further Work

The extension of MMS [30] presented here is part of our on-going efforts towards the goal of creating a general scheme for exploring high-dimensional C-spaces that is less sensitive to narrow passages than currently available tools. As discussed in Section 1.3 the original scheme imposes a set of conditions that in combination restrict an application of MMS to rather low dimensions. In this paper we chose to relax condition **C3**, namely by computing only approximations of three-dimensional manifolds. An alternative path is to relax condition **C2**, namely by not sampling the manifolds uniformly and independently at random. This would enable the use of manifolds of low dimension as it allows to enforce intersection. Following this path we envision a single-query planer that explores a C-space in an RRT-like fashion.

Another possibility is to explore other ways to compute approximative manifold samples, for instance, the (so far) exact representations of FSCs could be replaced by much simpler (and thus faster) but conservative⁸ approximations. This is certainly applicable to manifold samples of dimension one or two and should also enable manifold samples of higher dimensions. We remark that the use of approximations should not harm the probabilistic completeness as long as it is possible to refine approximations such that they converge to the exact results (equivalent to increased number of samples).

Using these extensions we wish to apply the scheme to a variety of difficult problems including assembly maintainability (part removal for maintenance [38]) by employing a single-query variant of the scheme. Additionally, we intend to extend the scheme to and experiment with motion-planning problems for highly-redundant robots as well as for fleets of robots, exploiting the symmetries in the respective C-space.

For supplementary material, omitted here for lack of space, the reader is referred to our web-page <http://acg.cs.tau.ac.il/projects/mms>.

⁸ Approximated FSC are contained in $\mathcal{C}_{\text{free}}$.

References

1. Salzman, O., Hemmer, M., Halperin, D.: On the Power of Manifold Samples in Exploring Configuration Spaces and the Dimensionality of Narrow Passages. CoRR abs/1202.5249 (2012), <http://arxiv.org/abs/1202.5249>
2. Aronov, B., Sharir, M.: On translational motion planning of a convex polyhedron in 3-space. SIAM J. Comput. 26(6), 1785–1803 (1997)
3. Avnaim, F., Boissonnat, J.D., Faverjon, B.: A practical exact motion planning algorithm for polygonal object amidst polygonal obstacles. In: Proceedings of the Workshop on Geometry and Robotics, pp. 67–86. Springer, London (1989)
4. Basu, S., Pollack, R., Roy, M.F.: Algorithms in Real Algebraic Geometry, 2nd edn. Algorithms and Computation in Mathematics. Springer (2006)
5. Berberich, E., Fogel, E., Halperin, D., Kerber, M., Setter, O.: Arrangements on parametric surfaces II: Concretizations and applications. MCS 4, 67–91 (2010)
6. Berberich, E., Fogel, E., Halperin, D., Mehlhorn, K., Wein, R.: Arrangements on parametric surfaces I: General framework and infrastructure. MCS 4, 45–66 (2010)
7. Canny, J.F.: Complexity of Robot Motion Planning (ACM Doctoral Dissertation Award). The MIT Press (1988)
8. Chazelle, B., Edelsbrunner, H., Guibas, L.J., Sharir, M.: A singly exponential stratification scheme for real semi-algebraic varieties and its applications. Theoretical Computer Science 84(1), 77–105 (1991)
9. Choset, H., Burgard, W., Hutchinson, S., Kantor, G., Kavraki, L.E., Lynch, K., Thrun, S.: Principles of Robot Motion: Theory, Algorithms, and Implementation. MIT Press (2005)
10. Dalibard, S., Laumond, J.P.: Linear dimensionality reduction in random motion planning. I. J. Robotic Res. 30(12), 1461–1476 (2011)
11. De Berg, M., Cheong, O., van Kreveld, M., Overmars, M.: Computational Geometry: Algorithms and Applications, 3rd edn. Springer (2008)
12. Fogel, E., Halperin, D., Wein, R.: CGAL Arrangements and their Applications. Springer, Heidelberg (2012)
13. Fogel, E., Halperin, D.: Exact and efficient construction of Minkowski sums of convex polyhedra with applications. CAD 39(11), 929–940 (2007)
14. Hachenberger, P.: Exact Minkowski sums of polyhedra and exact and efficient decomposition of polyhedra into convex pieces. Algorithmica 55(2), 329–345 (2009)
15. Halperin, D., Sharir, M.: A near-quadratic algorithm for planning the motion of a polygon in a polygonal environment. Disc. Comput. Geom. 16(2), 121–134 (1996)
16. Hirsch, S., Halperin, D.: Hybrid Motion Planning: Coordinating Two Discs Moving Among Polygonal Obstacles in the Plane. In: Boissonnat, J.-D., Burdick, J., Goldberg, K., Hutchinson, S. (eds.) Algorithmic Foundation Robotics. STAR, vol. 7, pp. 239–255. Springer, Heidelberg (2004)
17. Hsu, D., Latombe, J.C., Motwani, R.: Path planning in expansive configuration spaces. Int. J. Comp. Geo. & App. 4, 495–512 (1999)
18. 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)

19. Kavraki, L.E., Latombe, J.C., Motwani, R., Raghavan, P.: Randomized query processing in robot path planning. *JCSS* 57(1), 50–60 (1998)
20. Kavraki, L.E., 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)
21. Kuffner, J.J., LaValle, S.M.: RRT-Connect: An efficient approach to single-query path planning. In: *ICRA*, pp. 995–1001 (2000)
22. Ladd, A.M., Kavraki, L.E.: Generalizing the analysis of PRM. In: *ICRA*, pp. 2120–2125. IEEE Press (2002)
23. Latombe, J.C.: Robot Motion Planning. Kluwer Academic Publishers, Norwell (1991)
24. LaValle, S.M.: Rapidly-exploring random trees: A new tool for path planning. In Computer Science Dept., Iowa State University Tech. Rep., pp. 98–11 (1998)
25. LaValle, S.M.: Planning Algorithms. Cambridge University Press, Cambridge (2006)
26. Lien, J.M.: Hybrid motion planning using Minkowski sums. In: *RSS 2008* (2008)
27. Lozano-Perez, T.: Spatial planning: A configuration space approach. *MIT AI Memo* 605 (1980)
28. Plaku, E., Bekris, K.E., Kavraki, L.E.: OOPS for motion planning: An online open-source programming system. In: *ICRA*, pp. 3711–3716. IEEE (2007)
29. Reif, J.H.: Complexity of the mover’s problem and generalizations. In: *FOCS*, pp. 421–427. IEEE Computer Society, Washington, DC (1979)
30. Salzman, O., Hemmer, M., Raveh, B., Halperin, D.: Motion Planning via Manifold Samples. In: Demetrescu, C., Halldórsson, M.M. (eds.) *ESA 2011*. LNCS, vol. 6942, pp. 493–505. Springer, Heidelberg (2011)
31. Schwartz, J.T., Sharir, M.: On the “piano movers” problem: I. The case of a two-dimensional rigid polygonal body moving amidst polygonal barriers. *Commun. Pure Appl. Math.* 35, 345–398 (1983)
32. Schwartz, J.T., Sharir, M.: On the “piano movers” problem: II. General techniques for computing topological properties of real algebraic manifolds. *Advances in Applied Mathematics* 4(3), 298–351 (1983)
33. Sharir, M.: Algorithmic Motion Planning, *Handbook of Discrete and Computational Geometry*, 2nd edn. CRC Press, Inc., Boca Raton (2004)
34. Siek, J.G., Lee, L.Q., Lumsdaine, A.: *The Boost Graph Library: User Guide and Reference Manual*. Addison-Wesley Professional (2001)
35. The CGAL Project: *CGAL User and Reference Manual*, 3.7 edn. CGAL Editorial Board (2010), <http://www.cgal.org/>
36. Wein, R.: Exact and Efficient Construction of Planar Minkowski Sums Using the Convolution Method. In: Azar, Y., Erlebach, T. (eds.) *ESA 2006*. LNCS, vol. 4168, pp. 829–840. Springer, Heidelberg (2006)
37. Yang, J., Sacks, E.: RRT path planner with 3 DOF local planner. In: *ICRA*, pp. 145–149 (2006)
38. Zhang, L., Huang, X., Kim, Y.J., Manocha, D.: D-plan: Efficient collision-free path computation for part removal and disassembly. *Journal of Computer-Aided Design and Applications* (2008)

Sampling Extremal Trajectories for Planar Rigid Bodies

Weifu Wang and Devin Balkcom

Abstract. This paper presents an approach to finding the time-optimal trajectories for a simple rigid-body model of a mobile robot in an obstacle-free plane. Previous work has used Pontryagin’s Principle to find strong necessary conditions on time-optimal trajectories of the rigid body; trajectories satisfying these conditions are called *extremal trajectories*. The main contribution of this paper is a method for sampling the extremal trajectories sufficiently densely to guarantee that for any pair of start and goal configurations, a trajectory can be found that (provably) approximately reaches the goal, approximately optimally; the quality of the approximation is only limited by the availability of computational resources.

1 Introduction

Consider a single rigid body in an otherwise empty plane. The body can translate in various directions that are described relative to a frame rigidly attached to the body, or can rotate around various rotation centers whose locations are also fixed relative to the body frame. Let there also be some bounds on the velocities of these translations and rotations. This paper attacks the problem of finding the shortest or fastest trajectory to move the body from one configuration to another.

The system studied is quite specific relative to systems for which general-purpose motion planning algorithms have been successfully applied, and the algorithm we present is much weaker than algorithms derived from exact analytical descriptions of time-optimal trajectories for specific systems (*e.g.* Dubins [4], Reeds-Shepp [16], and others [18, 17, 11, 2, 1, 20, 9]). However, the theorems and results in this paper show, perhaps for the first time, that *provably* optimal motion planning is computationally feasible for a somewhat-general model of mobile robots in the plane.

Motion planning algorithms that take either a sampling approach (*e.g.* search-based planners [3], RRT-based planners [12], probabilistic roadmaps [10]) or an

Weifu Wang · Devin Balkcom
Dartmouth College, Dartmouth, NH
e-mail: Weifu.Wang@dartmouth.edu

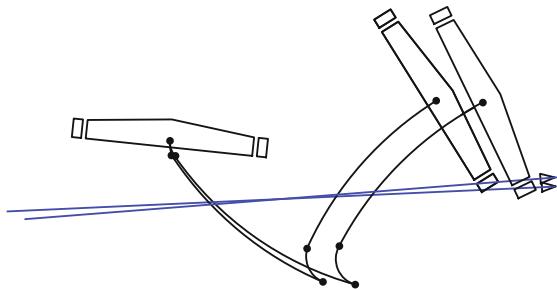


Fig. 1 Two control lines, and a corresponding extremal trajectory for each

optimization approach [15, 14, 5] have been very effective in solving very generally-stated kino-dynamic motion-planning problems in the presence of obstacles. However, because these algorithms do not take advantage of the specific details of any particular robot model, these algorithms do not provably return trajectories that meet some optimality criteria, and in some cases, cannot guarantee that a solution that reaches the goal will be found, even if one exists.

Over the past few years, we have been working on generalizing the strong analytical results that have been found for various vehicles. In previous work, we have been able to show some strong geometric necessary conditions on optimal trajectories for a general model of planar rigid bodies [6]. This rigid body might represent a simple model of a mobile robot (for example, the steered cars studied by Dubins, Reeds and Shepp, and others [4, 16, 18, 11, 19], differential drives[2], or omnidirectional vehicles [1, 20], or an object being manipulated in the plane by a robot arm, studied by *e.g.* Lynch [13]).

Essentially, every time-optimal trajectory can be characterized by a line in the plane, called the *control line*; motion of the body is essentially determined by the placement of this line, though the motion of the body is somewhat more complex than driving directly along the line. For example, figure 1 shows two characteristic control lines, and corresponding extremal trajectories; the vehicle is an asymmetric differential-drive robot – the maximum speed for the left wheel is 1.8 times the maximum speed for the right wheel. The problem of solving for an optimal trajectory would seem to be straightforward: given a starting configuration and a goal configuration, solve for the line placement that causes the vehicle to drive to the goal.

Solving for the placement of the control line is an inverse kinematics problem. For some simple robot geometries (*e.g.*, steered cars, differential drives, symmetric omnidirectional robots), an analytical solution can be found [4, 16, 2, 1, 20], but for other interesting variations of robot designs, we do not expect to be able to find analytical solutions.

An approach to solving an inverse kinematics problem is to sample a forwards kinematics problem. Sample the placements of the control line relative to the starting configuration, determine the structure of each trajectory, and choose a line whose corresponding trajectory reaches the goal. In [8], we tried this approach with

apparently good results, but because for some configurations small changes in the placement of the line can dramatically change the resulting trajectory, we were unable to prove that the sampling of line placements could be made sufficiently dense to find trajectories close to the optimal.

The main contribution of this paper is an alternate sampling strategy, and a collection of theorems that prove that this sampling strategy is sufficient to find approximately time-optimal trajectories that approximately reach any goal configuration. By *approximately reach*, we mean that a trajectory is found that passes no more than some small distance ε from any particular configuration, where ε can be chosen arbitrarily. By *approximately optimal*, we mean that this trajectory has a time cost that is within δ of the optimal time cost, where δ can be chosen to be arbitrarily small, up to limits determined by computational precision and effort. Based on this collection of theorems, we sketch an algorithm that densely samples the space of trajectories, building a complete map of trajectories from a starting configuration (canonically chosen to be at the origin) to every other configuration that can be reached within some time bound t_{\max} .

The basic idea is that for every candidate trajectory structure, there is a segment whose duration is most sensitive to the location of the control line, called the *most-sensitive segment*. We sample the duration of this most-sensitive segment densely, and use this sampled duration to compute some constraints on the placement of the control line; specifically, the value of the Hamiltonian, H . This H value is characteristic of a trajectory, and if the identity of successive controls in the trajectory are known, fully determines the structure of the rest of the extremal trajectory. Further sampling the duration of the first control in the trajectory fully determines the placement of the control line.

2 Model and Problem Statement

We now state the problem and model more formally. Let there be a frame attached to the body. The configuration of the body is described by the position and orientation of this frame relative to some world frame, $q = (x, y, \theta)$.

Let the control of the body be a Lebesgue-integrable vector function $u(t) \in \mathbf{R}^3$ that describes the translational and rotational velocities of the body at each time. For example, a constant function $u(t) = (1, 0, -1)$ would indicate that the body should translate with velocity one in the direction of the first axis of the body frame, and rotate with angular velocity -1 (a rotation in the clockwise direction).

The trajectory of the body in the world frame is determined by integrating the generalized velocity in the world frame. The controls can be transformed into velocities in the world frame by a 3×3 matrix R that is formed by replacing the upper left block of a 3×3 identity matrix with a 2×2 rotation matrix:

$$q(t) = q(0) + \int_0^t R(\theta(\tau))u(\tau) \quad (1)$$

Let the vector function $u(t)$ be constrained within a polyhedron. Polyhedral control sets appear frequently in models of mobile robots or rigid bodies being pushed stably within the plane. For example, bounds on steering angle and translational velocity for a steered car, or bounds on wheel speeds for differential-drive or three-wheeled omnidirectional robot, lead to linear (and thus polyhedral) constraints on controls.

The formal problem statement is, given a pair of start and goal configurations for a rigid body, as well as the polyhedron bounding the velocity controls, find a time-optimal trajectory between the two configurations.

3 Necessary Conditions for Time Optimality

In previous work, we have used Pontryagin's Maximum Principle to study necessary conditions on time-optimal trajectories for the rigid-body system. We call the trajectories satisfying the PMP *extremal*, and only a subset of extremal trajectories are optimal.

The PMP states that the Hamiltonian value of the system is the product of the control and an adjoint function, and during an optimal trajectory the control must maximize the Hamiltonian value, which is further a constant during this trajectory.

For the rigid body system we study, we have shown in previous work [7] that the Hamiltonian can be written in the following way:

$$H = k_1 \dot{x} + k_2 \dot{y} + \dot{\theta}(k_1 y - k_2 x + k_3), \quad (2)$$

where k_1 , k_2 and k_3 are constants of integration, and $(\dot{x}, \dot{y}, \dot{\theta}) = R(\theta(t))u(t)$ almost everywhere. Along any optimal trajectory, the control $u(t)$ must maximize equation 2, for some choice of constants. We can thus think of the Hamiltonian maximization equation as a control law that describes the evolution of extremal trajectories.

Initial configuration of the body and the particular choice of constants essentially determine the structure of the trajectory. For example, if $k_1 = k_2 = 0$, and $k_3 > 0$, then the Hamiltonian reduces to $H = k_3 \dot{\theta}$, and the controls must constantly maximize angular velocity. In previous work, we have labelled trajectories of this type *whirl*, and found a complete analytical solution method. Choosing different constants leads to different trajectories; the main problem is then to find the choice of constants that generate a trajectory to a particular desired goal.

If $k_1^2 + k_2^2 \neq 0$, it is convenient to scale the constants so that $k_1^2 + k_2^2 = 1$. Then there is a nice geometric interpretation of the results of the Maximum Principle. For a particular choice of constants, we can draw a line in the plane with heading along the vector (k_1, k_2) , a (signed) distance from the origin k_3 . Changing the constants moves this *control line* and gives different trajectory structures (figure 1). If we attach a frame to the line, with first axis in the direction of (k_1, k_2) and second axis in the direction $(-k_2, k_1)$, then the Hamiltonian can be written more simply as:

$$H = \dot{x}_L + y_L \dot{\theta}, \quad (3)$$

with subscript L indicating coordinates measured in the control line frame.

For some choices of control line and starting configuration, the evolution of the trajectory is completely determined by PMP and the control line; we call such trajectories *generic*, and they are the focus of this paper. Although there are non-generic (singular and whirl) trajectories, the problem of finding the fastest non-generic trajectory was previously solved analytically in [6].

3.1 Discrete Control Set

The maximization condition implies that only controls at vertices of the polyhedral control space occur in time-optimal generic trajectories [6]; thus we only need to consider a discrete control set. We denote the available controls (vertices of the original polyhedral control space) using \mathbf{u}_i , for $i \in \{1, 2, \dots, m\}$. For example, the control $\mathbf{u}_1 = (1, 0, 0)$, would correspond to driving forwards, and the control $\mathbf{u}_2 = (0, 0, 1)$ would correspond to spinning counterclockwise.

A trajectory structure is a sequence of controls. Denote the trajectory structure by \mathbf{s} . For example, $\mathbf{s} = (1, 3, 2, 1, 3, 2, \dots)$. The length of \mathbf{s} is n , indicating the trajectory has n segments. It is often necessary to refer to the control at the i th segment in a trajectory. For convenience, we define a function $U(s_i)$, such that $U(s_i) = \mathbf{u}_{s_i}$.

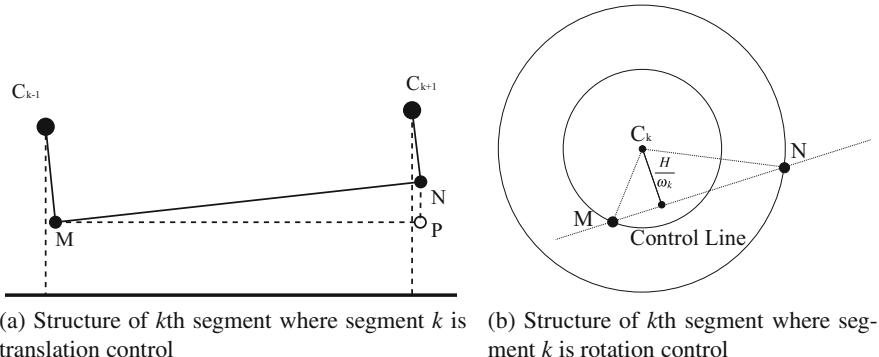
4 Sampling the Space of Control Lines

This section presents the main results. Theorems 1 and 2 show that for a particular extremal trajectory structure, choosing the durations of the first segment and one other segment is sufficient to determine the placement of the control line, and thus the duration of every other segment except the last. Sampling these two durations sufficiently finely guarantees that extremal trajectories are sampled densely enough that a trajectory can be found that is in some sense *close enough* to the time-optimal in both time (theorem 3) and distance (theorem 4).

Although the paper is technically self-contained, proofs of the theorems rely strongly on geometric interpretations of the motion of the rigid body relative to the control line; these geometric interpretations are introduced more gently in [6].

Theorem 1. *Given the duration of the k th segment, t_k , and the identity of three consecutive controls, $U(s_{k-1})$, $U(s_k)$, $U(s_{k+1})$, where $k \in \{2, 3, \dots, n-1\}$, the Hamiltonian value H can be uniquely calculated.*

Proof. First consider the case where s_k is a translation control; let the velocity be v_k . Figure 2a shows an example. Let M be the location of the body reference point at the switch *to* control $U(s_k)$, and let N be the location of this point at the switch *from* control $U(s_k)$. Let P be the point in the control-line frame with the same horizontal coordinate as N and the same vertical coordinate as M .

**Fig. 2** Figures for proof of theorem 1

From equation 3, the magnitude of the projection of the translation velocity vector onto the control line is H . The magnitude of the velocity vector perpendicular to the control line is therefore $\sqrt{v_k^2 - H^2}$, and we can compute the length $\|N - P\|$:

$$\|N - P\| = t_k \sqrt{v_k^2 - H^2}. \quad (4)$$

We know the identity of the previous and next control; we can use this and H to compute the distance of the rotation centers (C_{k-1} and C_{k+1}) to the control line. Specifically, C_{k-1} is a distance of H/ω_{k-1} from the control line, and C_{k+1} is H/ω_{k+1} from the control line. The radii of curvature corresponding to those controls are R_{k-1} and R_{k+1} .

The distance $\|N - P\|$ can also be determined from these two rotation controls, since the translation must be tangent to the circles centered at each rotation center with corresponding radii. We have:

$$\|N - P\| = \left| \left(\frac{H}{\omega_{k+1}} - \frac{HR_{k+1}}{v_k} \right) - \left(\frac{H}{\omega_{k-1}} - \frac{HR_{k-1}}{v_k} \right) \right|. \quad (5)$$

For notational brevity, denote $R_{k+1} - R_{k-1}$ by $R_{k+1,k-1}$, and $1/\omega_{k+1} - 1/\omega_{k-1}$ by $\omega_{k+1,k-1}$. Combining equations 4 and 5, and solving for H ,

$$H = \frac{t_k v_k}{\sqrt{t_k^2 + \omega_{k+1,k-1}^2 + \left(\frac{R_{k+1,k-1}}{v_k}\right)^2 - 2\omega_{k+1,k-1} \frac{R_{k+1,k-1}}{v}}}. \quad (6)$$

Now, consider the case where $U(s_k)$ is a rotation, shown in figure 2b. In previous work [6], we proved that for any switch from $U(s_m)$ to $U(s_n)$, there exists a point rigidly attached to the robot, such that at the switch between these controls, the point must be on the control line. Call switch $U(s_{k-1}) \rightarrow U(s_k)$ switch $k-1$, and call switch $U(s_k) \rightarrow U(s_{k+1})$ switch k .

Let M be the location of the switching point corresponding to the switch $k-1$, at the time of this switch. Let N be the location of the switching point corresponding to the switch k , at the time of this switch. The distances from C_k to N and from C_k to M are known [6]. The distance from C_k to the control line is H/ω_k .

Some triangle geometry allows a solution for H . Specifically, denote the line passing through M and N by $y\cos\beta - x\sin\beta + c = 0$. Denote the coordinates of M by (x_1, y_1) , the coordinates of N by (x_2, y_2) , and the coordinates of C_k by (x_k, y_k) . Then, we have:

$$y_1 \cos\beta - x_1 \sin\beta + c = 0 \quad (7)$$

$$y_2 \cos\beta - x_2 \sin\beta + c = 0 \quad (8)$$

$$\frac{H}{\omega_k} = |y_k \cos\beta - x_k \sin\beta + c| . \quad (9)$$

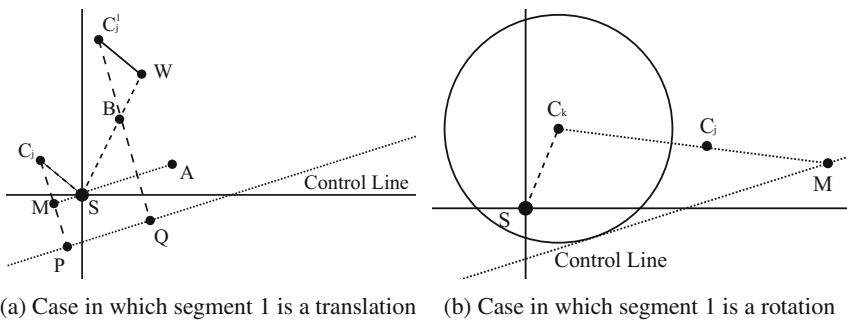
Solving the equation, we have:

$$\beta = \text{atan2}(y_2 - y_1, x_2 - x_1) \quad (10)$$

$$c = y_1 \cos\beta - x_1 \sin\beta \quad (11)$$

$$H = \omega_k |y_k \cos\beta - x_k \sin\beta + c| . \quad (12)$$

□



(a) Case in which segment 1 is a translation

(b) Case in which segment 1 is a rotation

Fig. 3 Figures for proof of theorem 2

Theorem 2. Given the identity of the first control, the Hamiltonian value, and the time for the first segment, t_1 , the angle of the control line (α) and the distance to the origin (r) can be calculated; there may be zero, one, or two solutions.

Proof. If $U(s_1)$ is translation, take figure 3a as an example. Denote the orientation of the control line by α . Denote the first control $U(s_1)$ by $(\dot{x}, \dot{y}, 0)$. Consider a unit vector \mathbf{SA} pointing along $(\cos \alpha, \sin \alpha)$, and a unit vector \mathbf{SB} pointing in the direction of (\dot{x}, \dot{y}) , where $\dot{x}^2 + \dot{y}^2 = v_1^2$. Denote the angle between these two vectors by γ ; then $\cos \gamma = H/v_1$. Denote the location of the body reference point at the time of the first switch by W . We can then derive:

$$\mathbf{SB} = (\dot{x}, \dot{y}); \quad \mathbf{SA} = (\cos \alpha, \sin \alpha) \quad (13)$$

$$\|S - W\| = v_1 t_1; \quad H/v_1 = (\dot{x} \cos \alpha + \dot{y} \sin \alpha)/v_1 \quad (14)$$

$$\tan^2 \frac{\alpha}{2} (H + \dot{x}) - 2\dot{y} \tan \frac{\alpha}{2} + (H - \dot{x}) = 0 \quad (15)$$

$$\Rightarrow \tan \frac{\alpha}{2} = (\dot{y} + \sqrt{v_1^2 - H^2})/(H + \dot{x}). \quad (16)$$

There are up to two solutions for α . For each α , to determine r , find a control j with the largest distance to a line aligned with vector $(\cos \alpha, \sin \alpha)$. Denote that distance by d_j , the control j by $\mathbf{u}_j = (\dot{x}_j, \dot{y}_j, \theta_j)$, the angular velocity by ω_j , where $\omega_j = \theta_j$, and denote the radius by $R_j = \sqrt{(\frac{\dot{x}_j}{\theta_j})^2 + (\frac{\dot{y}_j}{\theta_j})^2}$. We know that after time t_1 , that control will become the one maximizing the Hamiltonian value, so we have:

$$\|C_j - P\| = d_j; \quad \|M - P\| = r \quad (17)$$

$$d_j + t_1 \sqrt{v^2 - H^2} = \frac{H}{\omega_j} = \|C_j^1 - Q\| \quad (18)$$

$$R_j \cos(\text{atan} \frac{\dot{y}_j}{\dot{x}_j} + \text{acos} \frac{H}{v_1}) + r = d_j. \quad (19)$$

If the first control is a rotation (figure 3b), we then know where the first rotation center C_1 is. At the same time, using C_1 and t_1 , the duration of the first segment, we will know the configuration of the robot at the first switch.

For the first segment, we know C_1 , and we know H . There exists a circle centered at C_1 with radius $\frac{H}{\omega_1}$. At the first switch, we can find all the switching points for switches from C_1 to every other control. The control line must pass a switching point M and tangent to the circle, as shown in figure 3b. At the same time, the next control $U(s_2)$ must satisfy the necessary conditions. Since this is not a singular trajectory, at each switch, there are at most two controls that could maximize the Hamiltonian value, so the next control is unique. Therefore, we can calculate the α and r .

Denote the first control $U(s_1)$ by $(\dot{x}_1, \dot{y}_1, \dot{\theta}_1)$. Consider the switching point for switch 1: $U(s_1)$ to $U(s_2)$ ($U(s_2)$ can be any possible control), denote this switching point by M . Since we know the first control and the duration of the first segment

t_1 , we can then calculate the coordinates of M at first switch, denote by (x_s, y_s) . And we know C_1 has world frame coordinates $(-\frac{\dot{y}_1}{\theta_1}, \frac{\dot{x}_1}{\theta_1})$. Denote the control line by $y \cos \alpha + x \sin \alpha + r = 0$. Since the control line passes through (x_s, y_s) and has distance H/ω_1 to C_1 , we have:

$$y_s \cos \alpha - y_s \sin \alpha + r = 0 \quad (20)$$

$$\frac{H}{\omega_1} = \frac{\dot{x}_1}{\theta_1} \cos \alpha - \frac{-\dot{y}_1}{\theta_1} \sin \alpha + r. \quad (21)$$

We can then test if the second control could maximize the Hamiltonian value, and discard any solutions that violate PMP. Then, we can calculate α and r by solving the above system of equations. \square

4.1 Approximation Theorems

The two theorems stated above relate the position of the control line to the duration of different segments of an extremal trajectory. Assume we have two trajectories with the same structure (control sequence), starting at S . If the durations for all corresponding segments are similar, then we can see that at any time, the locations reached by the two trajectories will be close (as measured by Euclidean distance).

Lemma 1. *Consider two trajectories Y and Y' with identical structure, equal duration for all segments but one translation segment k , and a small distance δ . The corresponding end points are G and G' . For any translation control k in Y with duration t_k , and the same control k in Y_1 with duration t'_k , if $|t_k - t'_k| < \frac{\delta}{v_k}$, $\|G - G'\| < \delta$.*

Proof. Translation is commutative. \square

Now, let us consider how changing the duration of a rotation control can change the end point. Given some small angle δ and a vector v , assume we can rotate around any point on v . We want to know how much the end point can move. Given two points on the vector P and Q , if P is further from the end point, then rotating around P can change the end point more than rotating around Q . In the following lemma, we will prove that the upper bound of the movement of the end point is $v\delta$.

Lemma 2. *Consider a trajectory Y with end point G , a small angle δ , and two points P and Q on the trajectory, with $\|P - G\| > \|Q - G\|$. Form a new trajectory Y_1 with end point G_1 by rotating the trajectory from P to G around P by δ , and form another trajectory Y_2 with end point G_2 by rotating the trajectory from Q to G by δ . Then, $\|G_2 - G\| < \|G_1 - G\|$. What is more, for any trajectory Y_k (end point G_k) achieved by rotating the trajectory Y around a series of points along the trajectory with δ angle in total, $\|G_k - G\|$ is upper bounded by only rotating δ around the furthest point from G on Y .*

Proof. The trajectory from P to G can be viewed as a vector pointing from P to G . So can the trajectory from Q to G . Rotating the trajectory from P to G is equivalent to rotating a vector \mathbf{PQ} around P . Then

$$\|G_1 - G\| = 2 \sin \frac{\delta}{2} \|P - G\| \quad (22)$$

$$\|G_2 - G\| = 2 \sin \frac{\delta}{2} \|Q - G\| \quad (23)$$

Since $\|P - G\| > \|Q - G\|$, it follows that $\|G_2 - G\| < \|G_1 - G\|$.

Also, the combination of rotating around P and Q by a total angle of δ cannot move the end point further than $\|G_1 - G\|$. Denote $\|P - G\|$ by d_1 , and denote $\|Q - G\|$ by d_2 . Consider a rotation around P by δ_1 , followed by a rotation around Q by $\delta - \delta_1$. Denote this new trajectory by Y_3 , with end point G_3 . By the triangle inequality $\|G_3 - G\| \leq 2 \sin \frac{\delta_1}{2} d_1 + 2 \sin \frac{\delta - \delta_1}{2} d_2$. Let $d_2 = kd_1$ where $0 < k < 1$.

$$\sin\left(\frac{\delta_1}{2}\right) + k \sin\left(\frac{\delta - \delta_1}{2}\right) \leq \sin \frac{\delta}{2} \quad (24)$$

The upper bound is achieved when $\delta_1 = \delta$. Therefore, the change of the end point with any combination of rotations around points along trajectory with δ angle in total can be upper bounded by rotating around the furthest point from G with δ . \square

Note that, for any given start and goal, we can find a upper time bound t_{\max} for time-optimal trajectory by planning any (not necessarily optimal) path between the start and goal (Furtuna [6] gives a simple universal planner that can serve this purpose). At the same time, we can find a maximum speed over all controls and denote the absolute value of its speed by v_{\max}^C (compared to the fastest translation v_{\max}^T), and denote the absolute value of the fastest rotation by ω_{\max} . Based on these two quantities, we can then calculate the furthest point from the goal the trajectory can reach (because the distance between the reference and the goal may not always decreasing) $d_{\max} = \frac{t_{\max} v_{\max}^C}{2}$. Now, we will prove the following theorem.

Theorem 3. Consider two trajectories Y' and Y with the same structure (control sequence), both starting at S and having time cost t' and t . Denote the point that Y' passes through at t' by G' , and denote the point that Y passes through at t by G . For any $\varepsilon > 0$, there exists $\delta > 0$ such that if $|t' - t| < \delta$, $|G' - G| < \varepsilon$. Specifically, let δ be the minimum of $\frac{\sqrt{(\omega_{\max} t_{\max} v_{\max}^C / 2 + v_{\max}^C)^2 + 2\omega_{\max} v_{\max}^C \varepsilon} - (\omega_{\max} t_{\max} / 2 + 1)v_{\max}^C}{\omega_{\max} v_{\max}^C}$ and $\frac{\varepsilon}{v_{\max}^T}$.

Proof. Based on Lemma 1 and 2, we can derive an upper bound on how much a change in the duration of a segment can affect the location of the point a trajectory passes through at a certain time. The segment can either be a rotation or a translation. First let us consider translation. We have:

$$\|G^* - G\| < \varepsilon \quad (25)$$

$$\varepsilon \leq \delta v_{\max}^T \quad (26)$$

$$\delta \geq \frac{\varepsilon}{v_{\max}^T}. \quad (27)$$

Now consider the rotation case. According to Lemma 2, changing the duration of the rotation around the furthest point from G may lead to the biggest difference between trajectories in \mathbf{R}^2 . At the same time, for a small time δ , any rotation cannot change the orientation of the body more than $\delta\omega_{\max}$. Then Y can be rotated at most $\delta\omega_{\max}$ from Y' . Denote the upper time bound for Y' by t_{\max} . The upper time bound for Y can then be represented by $t_{\max} + \delta$. Denote the furthest point by P for both trajectories by translating one of the trajectories such that the two trajectories overlap at P . Without loss of generality, $\|P - G'\| < \|P - G\|$. We have:

$$\|G' - G\| < \varepsilon \leq 2 * \frac{\sin \omega_{\max} \delta}{2} \|P - G\| + \delta v_{\max}^C \quad (28)$$

$$\|P - G\| \leq \frac{v_{\max}^C(t+\delta)}{2} \quad (29)$$

$$\Rightarrow \varepsilon < \omega_{\max} \delta \frac{v_{\max}^C(t+\delta)}{2} + \delta v_{\max}^C \quad (30)$$

$$\Rightarrow \delta \geq \frac{\sqrt{(\omega_{\max} t_{\max} v_{\max}^C / 2 + v_{\max}^C)^2 + 2 \omega_{\max} v_{\max}^C \varepsilon} - (\omega_{\max} t_{\max} / 2 + 1) v_{\max}^C}{\omega_{\max} v_{\max}^C}. \quad (31)$$

We require the difference between Y and Y' to be less than ε ; therefore, choose the smaller δ . \square

Finally, we need to prove that small changes in t_1 and t_k cannot change the duration of an extremal trajectory too much.

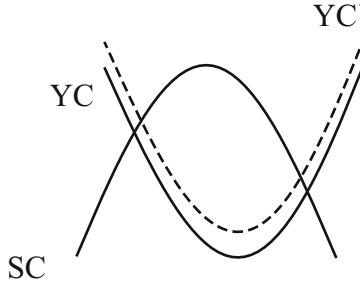


Fig. 4 The intersection between the same control (with different H) with switching curves in $y - \theta$ space

Theorem 4. Consider two extremal trajectories Y and Y' with the same trajectory structure. Let the duration of the first segments be t_1 and t'_1 . There exists a most-sensitive segment k , satisfying the properties below. Let the durations of segments k be t_k and t'_k . If $|t_1 - t'_1| < \Delta t$ and $|t_k - t'_k| < \Delta t$, then $|\sum_{i=1}^{n-1} t_i - \sum_{i=1}^{n-1} t'_i| < n\Delta t$.

Proof. Given a trajectory structure of one extremal trajectory, we will show that there exists a most sensitive translation control p and a most sensitive control q . A most sensitive segment k can be derived from control p and control q .

For the first $n - 1$ segments, we want to prove that even if we change t_1 and t_k by Δt_1 and Δt , the total duration of those segments does not change more than $n\Delta t$. Then, we need to prove that for each segment, changing t_1 by Δt_1 and changing t_k by Δt does not change the duration by more than Δt .

The trajectory is represented by H and α . First let us consider α . As we have seen in the previous section, given H , changing α only affects the duration of the first segment. So, as long as we change the duration of the first control t_1 by $\Delta t_1 < \Delta t$, we can be assured that changing t_1 by Δt_1 will affect the total duration of the trajectory no more than Δt .

Now, let us consider the Hamiltonian value. For a small, fixed change of H , the duration of each segment $t_k, k = 2, 3, \dots, n - 1$ may change differently. We will prove that for this change ΔH , there exists a segment k such that the duration of k th segment (the most sensitive one) changes the most. We find the most sensitive segment k base on the most sensitive rotation control and the most sensitive translation control.

First, let us consider rotation. Consider the switching points in the control line frame. At each switch, we can observe the signed distance of the body from the control line y_L , and the angle of the body relative to the control line θ_L . Imagine fixing each switching point on the control line and rotating the robot around the switching point. We then get a set of sinusoids indicating the relative position of y_L and θ_L at a switch for different H values. If we plot these sinusoids in a frame where y_L is the second axis while θ_L is the first axis, we call these sinusoids *switching curves*, and this coordinate system $y - \theta$ space.

Now, consider generic trajectories in the $y - \theta$ space; take figure 4 as an example. Curves YC and YC' follow the same control sequences, but have different H values. Let the difference in H values be ΔH . To calculate how much the duration of this control changes, we need to calculate how much the θ changes at the intersection. For a unit change of θ , the y can change no more than $\max \frac{dy}{d\theta} * \Delta \theta$, which is denoted by Δy . We choose a suitable scale of θ and y such that the switching curve can be represented by $\sin \theta$ and the control curve TC (the control has angular velocity ω_i and the radius is R_i) is represented by $y_c = A \sin(\gamma\theta + \beta) + B$. We have the following

$$A = \frac{H}{\omega_i} \quad (32)$$

$$\Delta H \leq \Delta y + y'_c * \Delta \theta \quad (33)$$

$$\Delta y \leq \theta \quad (34)$$

$$\Delta \theta * y'_c \leq \Delta \theta \left(\frac{H}{\omega_k} - R_k \right) \quad (35)$$

$$\Delta t_k = \frac{2\Delta \theta}{\omega_k} \leq \frac{2\Delta H}{\omega_k + H} \leq \frac{2\Delta H}{\omega_k}. \quad (36)$$

So, the most sensitive rotation control (MSRC) is the one with the smallest absolute angular velocity. For unit change of duration of each segment, the ΔH for the most sensitive rotation control is the smallest. So, if we change the duration of the

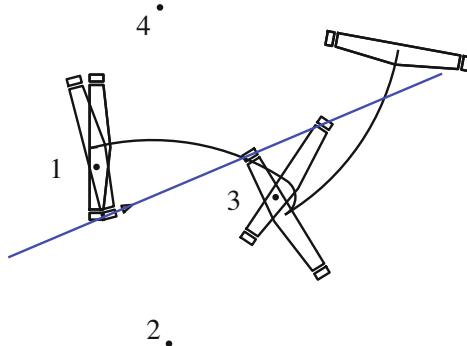


Fig. 5 An approximate time-optimal motion for the asymmetric differential-drive derived using the sampling algorithm, with a labelled sequence of rotation centers corresponding to each control

most sensitive control by Δt , the duration of all the other segments changes by less than Δt .

Now, let us consider the translation case. Two translations cannot be adjacent in an optimal trajectory [6]. So, a translation $U(s_k)$ (with speed v_k) must follow a rotation $U(s_{k-1})$. The translation is followed by a rotation $U(s_{k+1})$. Since we know the trajectory structure, we then can calculate the distance the translation need to cover in control line frame's y direction, take figure 2a as an example. We know $\|N - P\|$; therefore, we know the duration of the translation: $\frac{\|N - P\|}{\sqrt{v_k^2 - H^2}}$. Then,

$$\Delta t_k = |\omega_{k-1, k+1}| \left(\frac{H + \Delta H}{\sqrt{v_k^2 - (H + \Delta H)^2}} - \frac{H}{\sqrt{v_k^2 - H^2}} \right). \quad (37)$$

The translation with the smallest v_k is therefore the most sensitive translation control (MSTC). If we change the duration of that segment by Δt , then the ΔH is small enough to ensure that the duration change of all the other segments are smaller than Δt . Function $\frac{H + \Delta H}{\sqrt{v_k^2 - (H + \Delta H)^2}} - \frac{H}{\sqrt{v_k^2 - H^2}}$ is increasing with respect to H .

We can then compare the Δt_k calculated from rotation and translation to find the most sensitive segment k such that for any unit change of H , the duration of segment k changes the most. Therefore, if we change the duration of that segment by Δt , the duration of all the other segments will not change more than Δt , and the difference between t and t' is guaranteed to be bounded within $n\Delta t$. \square

5 A Sampling Algorithm

Based on the above theorems, we now sketch an algorithm that can be used to sample the space of placements of the control line. The inputs to the sampling algorithm are: the control set \mathbf{u} , an upper bound on the total duration of the trajectory t_{\max} , and

the tolerances ε and δ . The output is a list of (α, r) values describing control line placements. One of these placements is guaranteed to contain an extremal of the correct structure and duration to approximately reach any particular goal configuration closer than t_{\max} , with time no worse than the optimal time plus δ .

1. Based on t_{\max} , calculate the maximum possible segment count, n_{\max} . Periods have a bounded number of segments. In [7], generic extremal trajectories were divided into two types: *roll* and *shuffle* trajectories. It was further shown that shuffle trajectories cannot be longer than one period. For roll trajectories, a period requires orientation of the robot change 2π , which take certain amount of time. Therefore, t_{\max} can be used to bound the number of periods that a roll trajectory can include.
2. Calculate step size Δt . In theorem 3, we proved that for any trajectory, if the total duration of the trajectory is changed by some δ , no point on the trajectory will move more than ε . So, for any given ε , we can calculate a satisfying δ . We can then use theorem 4 to compute a sampling step size Δt , with $\Delta t \leq \delta/n_{\max}$.
3. Calculate every possible extremal trajectory structure given the set of controls, using the approach outlined in [7].
4. For each trajectory structure, find a most sensitive segment k , using the method outline in the proof of theorem 4.
5. For each trajectory structure, sample the duration, t_k , of the most sensitive segment (using Δt as step size) and compute the corresponding value for H using theorem 1. If segment k is rotation, an upper bound on t_k is $2\pi/\omega_k$; if translation, the upper bound is t_{\max} .
6. For each trajectory structure, for each H value, consider each control in the structure as a possible first control. Sample the first control duration t_1 using Δt . Use t_1 , H , the identity of the first control, and theorem 2, to compute the location of the control line, (α, r) . Save each (α, r) pair to a list.
7. For each computed control line placement, sample the total duration of the trajectory, and use the control line to generate (simulate) the trajectory. For each sampled total duration, calculate the Δq and build the mapping between the trajectory (control line) and Δq .

The algorithm described generates a trajectory suitably close (specified by ε and δ) to an optimal trajectory between any pair of start and goal configurations that are not too distant from one another (specified by t_{\max}).

First, between a particular q_s and q_g , there exists a time-optimal trajectory; call it Y^* . This optimal trajectory must be extremal, has a particular structure S^* , and has particular durations for each segment. The algorithm considers each possible trajectory structure, and must therefore find a trajectory with structure S^* . Further, S^* has a most sensitive segment s_k^* , with duration t_k^* . A sampled trajectory is considered with duration t_k such that $|t_k^* - t_k| < \Delta t$, and because s_k is the most sensitive segment, every other segment duration must be within Δt of the optimal.

6 Implementation Results

We implemented a slight variation on the algorithm described above in C code. The variation we implemented is perhaps somewhat easier to implement, since it avoids generating trajectory structures explicitly (step 3 above), but algorithm correctness is somewhat more challenging to explain. The primary difference between the two algorithms is that instead of looping over trajectory structures, the variation loops instead over possible trajectory substructures $U(s_{k-1})$, $U(s_k)$ and $U(s_{k+1})$. To ensure that all trajectory structures are considered, it is also required to add a set of certain critical values of H to the generated list of H values.

We used the implementation first to test the equations described in theorems 1 and 2 for computing r and α given t_1 and t_k , against known analytical results for the differential robot and the omnidirectional robot that compute all segment durations for a given r and α .

We also used the sample implementation to generate the set of (α, r) values for the asymmetric differential-drive robot drive with one wheel that can drive 1.8 faster than the other, with $\varepsilon = .1$, where the distance between wheels is 2. Figure 5 shows an example near-optimal trajectory. On a standard desktop machine, generating the complete set of about 10, 000, 000 (α, r) pairs required about nine minutes; using this set to generate (simulate) a trajectory for each (α, r) pair (with $t_{\max} = 10$) required a further nine minutes. Though computationally expensive, it should be pointed out that this solution finds the complete mapping from a starting configuration to all goals reachable within 10 seconds.

Further testing and simulation of other systems are a current goal, as well as techniques to improve precision and run-time.

7 Conclusion

The algorithm we presented is fairly specific to a limited class of mobile robots. However, the approach highlights some ideas that we hope will prove a starting point for work at the intersection of optimal control and the design of planning algorithms. First, strong geometric conditions on optimal trajectories derived analytically may allow representation of optimal trajectories using only a few parameters. Second, searching this space of parameters directly may miss certain trajectory structures. However, an indirect sampling method may allow construction of a set of sampled parameters that do allow guarantees on optimality.

The algorithm presented, in order to guarantee near-optimality, samples trajectories to all configurations in some region – an “all-pairs” approach to the motion planning problem. Once constructed, this mapping may allow both an analysis of the structure of optimal trajectories of the configuration space, and may be useful for fast queries about trajectories between particular pairs of configurations.

A related interesting problem that we have not considered in the present paper is finding a trajectory between a particular pair of configurations, more precisely and efficiently than finding the complete mapping. Theoretical results about the

relationship between parameters that describe trajectories and the duration of trajectory segments may allow more sophisticated branch-and-bound or local-search algorithms that are still provably good approximations.

This work was supported in part by NSF CAREER grant IIS-0643476; we also thank Andrei Furtuna for his advice and suggestions.

References

1. Balkcom, D.J., Kavathekar, P.A., Mason, M.T.: Time-optimal trajectories for an omnidirectional vehicle. *International Journal of Robotics Research* 25(10), 985–999 (2006)
2. Balkcom, D.J., Mason, M.T.: Time optimal trajectories for differential drive vehicles. *International Journal of Robotics Research* 21(3), 199–217 (2002)
3. Barraquand, J., Latombe, J.-C.: Nonholonomic multibody mobile robots: Controllability and motion planning in the presence of obstacles. In: *IEEE International Conference on Robotics and Automation*, Sacramento, CA, pp. 2328–2335 (1991)
4. Dubins, L.E.: On curves of minimal length with a constraint on average curvature and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics* 79, 497–516 (1957)
5. Fernandes, C., Gurvits, L., Li, Z.X.: A variational approach to optimal nonholonomic motion planning. In: *IEEE International Conference on Robotics and Automation*, pp. 680–685 (April 1991)
6. Furtuna, A.A.: Minimum time kinematic trajectories for self-propelled rigid bodies in the unobstructed plane. Dartmouth PhD thesis, Dartmouth College (2004)
7. Furtuna, A.A., Balkcom, D.J.: Generalizing Dubins curves: Minimum-time sequences of body-fixed rotations and translations in the plane. *International Journal of Robotics Research* 29(6), 703–726 (2010)
8. Furtuna, A.A., Lu, W., Wang, W., Balkcom, D.J.: Minimum-time trajectories for kinematic mobile robots and other planar rigid bodies with finite control sets. In: *IROS*, pp. 4321–4328 (2011)
9. Hayet, J.-B., Esteves, C., Murrieta-Cid, R.: A Motion Planner for Maintaining Landmark Visibility with a Differential Drive Robot. In: Chirikjian, G.S., Choset, H., Morales, M., Murphey, T. (eds.) *Algorithmic Foundation of Robotics VIII. STAR*, vol. 57, pp. 333–347. Springer, Heidelberg (2009)
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 and Automation* 12(4), 566–580 (1996)
11. Laumond, J.-P.: Feasible trajectories for mobile robots with kinematic and environment constraints. In: *International Conference on Intelligent Autonomous Systems*, pp. 346–354 (1986)
12. LaValle, S.M.: *Planning Algorithms*. Cambridge Press (2006), also freely available online at <http://planning.cs.uiuc.edu>
13. Lynch, K.M.: The mechanics of fine manipulation by pushing. In: *IEEE International Conference on Robotics and Automation*, Nice, France, pp. 2269–2276 (1992)
14. Oberle, H.J., Grimm, W.: BNDSCO: a program for the numerical solution of optimal control problems (1989)
15. Ratliff, N., Zucker, M., Andrew (Drew) Bagnell, J., Srinivasa, S.: CHOMP: Gradient optimization techniques for efficient motion planning. In: *IEEE International Conference on Robotics and Automation*, ICRA (May 2009)

16. Reeds, J.A., Shepp, L.A.: Optimal paths for a car that goes both forwards and backwards. *Pacific Journal of Mathematics* 145(2), 367–393 (1990)
17. Souères, P., Boissonnat, J.-D.: Optimal Trajectories for Nonholonomic Mobile Robots. In: Laumond, J.-P. (ed.) *Robot Motion Planning and Control*. LNCIS, vol. 229, pp. 93–170. Springer, Heidelberg (1998)
18. Sussmann, H., Tang, G.: Shortest paths for the Reeds-Shepp car: a worked out example of the use of geometric techniques in nonlinear optimal control. SYCON 91-10, Department of Mathematics, Rutgers University, New Brunswick, NJ 08903 (1991)
19. Vendittelli, M., Laumond, J.P., Nissoux, C.: Obstacle distance for car-like robots. *IEEE Transactions on Robotics and Automation* 15(4), 678–691 (1999)
20. Wang, W., Balkcom, D.J.: Analytical time-optimal trajectories for an omni-directional vehicle. In: *IEEE International Conference on Robotics and Automation* (2012)

Convex Hull Asymptotic Shape Evolution*

Maxim Arnold, Yuliy Baryshnikov, and Steven M. LaValle

Abstract. The asymptotic properties of Rapidly exploring Random Tree (RRT) growth in large spaces is studied both in simulation and analysis. The main phenomenon is that the convex hull of the RRT reliably evolves into an equilateral triangle when grown in a symmetric planar region (a disk). To characterize this and related phenomena from flocking and swarming, a family of dynamical systems based on incremental evolution in the space of shapes is introduced. Basins of attraction over the shape space explain why the number of hull vertices tends to reduce and the shape stabilizes to a regular polygon with no more than four vertices.

1 Introduction

Rapidly exploring Random Trees (RRTs) [11] have become increasingly popular as a way to explore high-dimensional spaces for problems in robotics, motion planning, virtual prototyping, computational biology, and other fields. The experimental successes of RRTs have stimulated interest in their theoretical properties. In [12], it was established that the vertex distribution converges in probability to the sampling distribution. It was also noted that there is a “Voronoi bias” in the tree growth because the probability that a vertex is selected is proportional to the volume of its Voronoi region. This causes aggressive exploration in the beginning, and gradual refinement until the region is uniformly covered. The RRT was generalized so that uniform random samples are replaced by any dense sequence to obtain deterministic convergence guarantees that drive dispersion (radius of the largest empty ball) to zero [13]. The lengths of RRT paths and their lack of optimality was studied in two recent works. Karaman and Frazzoli prove that RRTs are not asymptotically optimal

Maxim Arnold · Yuliy Baryshnikov · Steven M. LaValle

University of Illinois, Urbana, IL 61801, USA

MA: Institute for Information Transmission Problems, Moscow, Russia

e-mail: {mda, ymb, lavalle}@uiuc.edu

* Supported by grants from AFOSR (FA9550-10-1-0567) and ONR (N00014-11-1-0178).

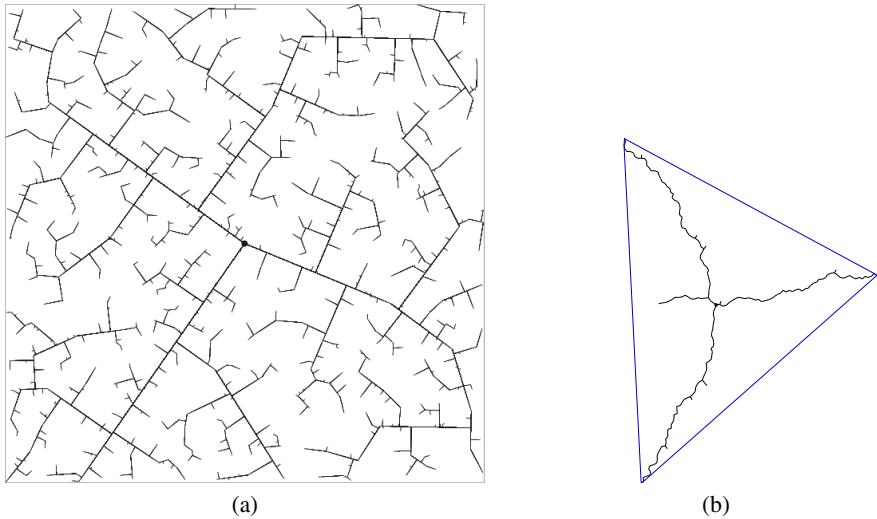


Fig. 1 (a) An RRT after 390 iterations. (b) An RRT grown from the center of a “large” disc, shown with its convex hull.

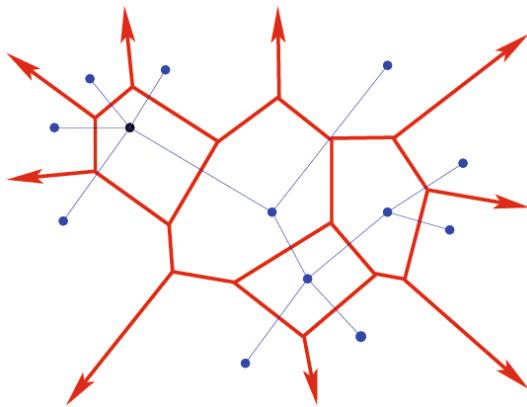


Fig. 2 The Voronoi diagram of RRT vertices contains interior Voronoi regions, which are bounded by other Voronoi regions, and exterior Voronoi regions, which extend to the boundary of the space

and propose RRT*, which is a variant that asymptotically yields optimal path lengths [6]. Nechushtan, Raveh and Halperin developed an automaton-based approach to analyzing cases that lead to poor path quality in RRTs [14]. More broadly, there has also been interest in characterizing convergence rates for other sampling-based planning algorithms that use random sampling [4, 5, 10].

In this paper, we address one of the long-standing questions regarding RRT behavior. Figure 1(a) shows how an RRT appears in a square region after 390 iterations. What happens when the size of the search space is increased? In the limiting case in which an RRT sequence is generated in a large disc in the plane, simulations reliably produce three straight tree branches, roughly 120 degrees apart. See Figure 1(b). In terms of Voronoi bias, the exterior Voronoi regions completely dominate because the probability that random samples fall into interior Voronoi regions shrinks to zero; see Figure 2. Thus, the RRT spends virtually all of its time expanding, rather than refining in areas it has already explored.

It is remarkable that the convex hull of the RRT throughout the process is approximately an equilateral triangle with random initial orientation. This has been consistently observed through numerous simulations, and it known for over a decade with no rigorous analysis. How can this behavior be precisely characterized? Why does it occur? This motivates our introduction of *convex hull asymptotic shape evolution*, CHASE, which is a family of dynamical systems that includes the RRT phenomenon just described.

In addition to understanding RRTs, we believe that the study of CHASE dynamics encompasses a broader class of problems. Another, perhaps comparable in importance, motivation for our research comes from the general desire to understand self-organization in large, loosely interacting collectives of agents, whether natural or artificial. Although the literature on flocking, swarming, and general dynamics of large populations is immense, note that it is predominantly concentrated on *locally interacting* agents. Recently, a new trend has emerged, dealing with agents interacting over arbitrary distances. As examples, [16] deals with scale-invariant rendezvous protocols in swarms and naturalists provide us with the evidence that scale-invariant (*topological*, in their parlance) interactions exists in the animal world [2].

2 Dynamics of Shapes

2.1 Rapidly Exploring Trees

The *Rapidly exploring Random Tree* (RRT) is an incremental algorithm that fills a bounded, convex region $X \subset \mathbb{R}^d$ in the following way. Let $T(V, E)$ denote a rooted tree embedded in X so that $V \subset X$ and every $e \in E$ is a line segment (with $e \subseteq X$). An infinite sequence $(T_0 \subset T_1 \subset T_2 \subset \dots)$ of trees is constructed as follows. For $T_0(V_0, E_0)$, let $E_0 = \emptyset$ and $V_0 = \{x_{\text{root}}\}$ for any chosen $x_{\text{root}} \in X$, designated as the *root*. Each T_i is then constructed from T_{i-1} . Let $\varepsilon > 0$ be a fixed *step size*. Select a point x_{rand} uniformly at random in X and let x_{near} denote the nearest point in T_{i-1} (in the union of all vertices and edges). If $x_{\text{near}} \in V_{i-1}$ and $\|x_{\text{near}} - x_{\text{rand}}\| \leq \varepsilon$, then T_i is formed by $V_i := V_{i-1} \cup \{x_{\text{rand}}\}$ and $E_i := E_{i-1} \cup \{e_{\text{new}}\}$ in which e_{new} is the segment that connects x_{near} and x_{rand} . If $\|x_{\text{near}} - x_{\text{rand}}\| > \varepsilon$, then an edge of length ε is formed instead, in the direction of x_{rand} , with x_{near} as the leaf vertex.

If $x_{\text{near}} \notin V_{i-1}$, then it must appear in the interior of some edge $e \in E$. In this case, e is split so that both of its endpoints connect to x_{near} . Recall Figure 1(a), which shows a sample RRT for $X = [0, 1]^2$. The RRT described here uses the entire swath for selection, rather than vertices alone, as described in [11]. For a discussion of how these two variants are related, see [13]; the asymptotic phenomena are independent of this distinction.

2.2 An Isotropic Process

Consider the case in which the starting point x_{root} is at the origin, and the region to be explored (and the sampling density) is *rotationally invariant*. In the limit of small ε , the RRT growth is governed by the direction towards the x_{rand} (far away), which can be assumed to be uniformly distributed on the unit circle or directions. In this case, the dynamics of the *convex hulls* of the successive RRTs can be described as follows.

Let \mathbf{P}_n denote the convex hull of the points $x_1, \dots, x_n \in \mathbb{R}^2$. The random polygon \mathbf{P}_{n+1} depends on x_1, \dots, x_n only through \mathbf{P}_n via the following iterative process:

Algorithm 1. Dynamics of the convex hulls of early RRTs in the limit of small step sizes.

```

 $n \leftarrow 0, \mathbf{P}_0 \leftarrow \mathbf{0}$ 
loop
   $l \leftarrow \text{rand}(S^1)$ 
   $v \leftarrow \text{argmax} \langle l, x \rangle, x \in \mathbf{P}_n$ 
   $\mathbf{P}_{n+1} \leftarrow \text{conv}(\mathbf{P}_n \cup \{v + \varepsilon l\})$ 
   $n \leftarrow n + 1$ 
end loop

```

This algorithm defines an increasing family of (random) convex polygons. One might view it as a (growing) Markov process on the space of convex polygons in \mathbb{R}^2 starting with the origin \mathbf{P}_0 .

Note that as the Markov chain evolves, the number of vertices in \mathbf{P}_n can change. Some vertices can disappear, being swept over by the convex hull of the newly added point. This new point does not eliminate the vertex v_l , where the functional $\langle \cdot, l \rangle$ attains its maximum if and only if the vector $-l$ does not belong to the cone $T_{v_l} \mathbf{P}_n$, the tangent cone to \mathbf{P}_n at v_l . This, in turn, can happen only if the angle at v_l is acute (less than $\pi/2$).

The Enigma of the Symmetry Breaking

Although the distribution of this growth process is manifestly rotationally invariant, our simulations show that after long enough time the polygons \mathbf{P}_n are not becoming round. On the contrary, it was observed that the polygons \mathbf{P}_n become close

(in *Hausdorff metric*¹) to large equilateral triangles. (It should be noted that the polygons are actually triangles only for a fraction of the time; typically some highly degenerate - with an interior angle close to π - vertices are present.)

Although we still do not understand completely the mechanisms of symmetry breakage and of the formation of the asymptotic shapes, there are several asymptotic results and heuristic models that shed enough light on the process to at least make plausible explanations of the observed dynamics. This paper addresses with these results and models.

2.3 Sources of Difficulties

The biggest problem emerging when one attempts to analyze the Markov process $\{\mathbf{P}_n\}_{n=1,2,\dots}$ is the lack of a natural parameterization of the space of the polygons: An attempt to account for all convex planar polygons at once leads immediately to an infinite dimensional (and notoriously complicated) space of convex support functions (with C^0 norm-induced topology). Indeed, starting with any polygon, the Markov process will have a transition, with positive probability, changing the collection of vertices forming the convex polygon \mathbf{P}_n .

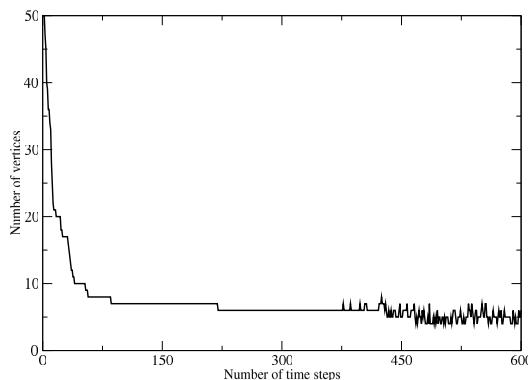


Fig. 3 Typical behavior of the number of vertices of \mathbf{P}_n , starting with a regular 50-sided polygon

While the experimental evidence suggests that the polygons \mathbf{P}_n become close to equilateral triangles in Hausdorff metric, there is little hope to understand the process of the *vertices* of \mathbf{P}_n because even the number of vertices changes with high frequency (see Fig. 3)

¹ A Hausdorff metric between two compact subsets of a metric space can be defined as $d_H(A, B) = \max_{a \in A} \min_{b \in B} d(a, b) + \max_{b \in B} \min_{a \in A} d(a, b)$.

2.4 Outline of the Results

For the reason above, we adopt a circumstantial route here. In lieu of addressing the complicated dynamics defined by Algorithm 1 throughout, we concentrate here on the time intervals over which the number of the vertices of \mathbf{P}_n remains constant. During each such interval, the evolution of the convex polygons \mathbf{P} can be described as a relatively straightforward (time-homogeneous) Markov process on the space of the k -sided polygons.

If the number of vertices of polygons \mathbf{P}_n were to stabilize, then the size (perimeter) of these polygons would grow linearly in time. This suggests considering a Markov process on the space of *shapes* of k -sided polygons. To obtain a *shape* from the original polygon one could fix a direction of the one of the sides and scale the whole polygon to have a fixed circumference. To be consistent one should apply the same scaling also on the incremental steps of the original Markov process. We define the state space of this new process, which is the space of shapes of k -sided polygons, in Section 2.5).

Since increments of the new process are scaled at each time-step of the algorithm, it cannot be time-homogeneous anymore. Since size of the original polygon grows linearly with time we approximate scaled process on the space of shapes by the similar one with the increments exactly of size $1/n$.

Markov processes with diminishing step sizes are quite familiar in the theory of stochastic approximation, where we represent the Markov process as a shift along a vector field (obtained by the *averaging*) perturbed by small random noise.

We will describe this averaged dynamical system (which we will refer to as CHASE, for Convex Hull Asymptotic Shape Evolution) in Section 3. Interestingly, it can be interpreted (up to a time reparameterization) as the gradient field for the circumference of the polygonal chain (see proposition 2 for the details): CHASE dynamics tries to make the polygonal chain longer the fastest way!

It is well known that for small steps, the stochastically perturbed processes are tracing closely their deterministic, averaged counterparts for long times. More to the point, classical results (see [9, 15]) on stochastic approximations imply that *perturbed dynamical system remains trapped with probability one near an exponentially stable equilibrium of the averaged dynamics*, if deviations of the stochastic perturbation scale such as $1/n$ (or any other scale ε_n such that $\sum \varepsilon_n = \infty$ and $\sum \varepsilon_n^2 < \infty$).

Thus, the first natural step to try to explain the lack of polygonal shapes other than equilateral triangles in the long runs of \mathbf{P}_n would be to analyze the averaged dynamics of our scaled Markov processes on the polygonal chains, locating their equilibria and studying their stability.

As it turned out, for k -sided polygonal chains, the only nondegenerate fixed points of CHASE are the regular polygons². Moreover, we prove that these equilibria are unstable for $k \geq 5$.

² CHASE can be extended to non-convex polygons; the regular polygons therefore can be non-convex ones as well.

Our averaging procedure hardly relies on the fact that the number of vertices of the polygon (equivalently, the dimension of the corresponding space of shapes) remains fixed. Thus, for k -sided polygons with obtuse internal angles, the scaled Markov chain \mathbf{P}_n is well-approximated by CHASE. In general, for polygons with acute internal angles, \mathbf{P}_n has non-zero probability of changing the number of vertices. From that it follows that an averaging procedure cannot be applied without modifications for the triangles and quadrangles. However we strongly believe that one can modify the averaging procedure in such a way, that generalized CHASE dynamics could be extended to that cases. In Section 4.5 we present some results that heuristically lend support to the natural conjecture that the regular triangles are stable shapes under any generalized averaged dynamics.

We conclude the paper with a short description of limit of CHASE for the obtuse k -sided polygons, as $k \rightarrow \infty$.

2.5 A Space of Shapes

We begin with the formal description of the space of shapes.

The Construction

A *configuration* of k points (or k -*configuration*) is a collection of distinguishable (uniquely labeled) points in \mathbb{R}^2 , not all coincident. Traditionally, the *shape* of a configuration is understood as its class under the equivalence relation on the configurations defined by the Euclidean motions and the change of scale. More specifically, two configurations $\{x_1, \dots, x_k\}$ and $\{x'_1, \dots, x'_k\}$ are said to have the same shape if there exists some rotation $U \in SO(2)$, translation vector $v \in \mathbb{R}^2$ and scaling constant $\lambda > 0$ so that $x'_i = \lambda U x_i + v$ for all i from 1 to k . Factorings by the scale and the displacement admit sections: One can assume that the center of gravity of the configuration is at the origin and its moment of inertia is 1 (or that $\sum_i |x_i|^2 = 1$) – this is where the condition that not all points coincide is important. For a natural interpretation of the space shape in terms of symplectic reduction, see [8].

Complex Projective Spaces

Perhaps most explicit way to think about the space of shapes of planar k -configurations is the following: Viewing \mathbb{R}^2 as \mathbb{C} , we can interpret the k -configuration with the center of mass at the origin as a point in $\mathbb{C}^{k-1} - \{0\}$. Factoring by rotations and rescaling is equivalent to factoring by \mathbb{C}_* , the multiplicative group of complex numbers, whence the space of shapes is isomorphic (as a Riemannian manifold) to \mathbb{CP}^{k-2} , the $(k-2)$ -dimensional complex projective space.

The simplest nontrivial case $k = 3$ is very instructive: \mathbb{CP}^1 is isometric to the Riemannian sphere S^2 . The poles of the sphere can be identified with the equilateral triangles (North or South depending on the orientation defined by the cyclic order of x_1, x_2, x_3); the equator corresponds to the collinear triples, and the parallels to the level sets of the (algebraic) area, considered as the function of the configurations

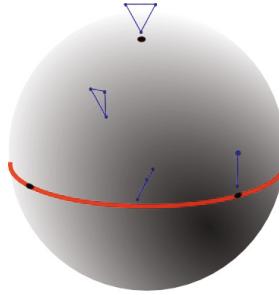


Fig. 4 The sphere of shapes of triangles. The equator consists of degenerate triangles (three points collinear). The poles correspond to equilateral triangles.

rescaled to unit moment of inertia (see Figure 4). For related shape space constructions, see [7] and an application to robotics in which probability mass is placed over the shape space [3].

Tame Configurations and the k -Gon Dynamics

The space of planar k -configurations contains an open subset consisting of *tame* ones, which we define here as the configurations for which every two consecutive sides form a positively oriented frame (in particular, meaning all points of the configuration are distinct) and the angle between these vectors is acute:

$$(x_i - x_{i-1}) \times (x_{i+1} - x_i) > 0 \text{ and } (x_i - x_{i-1}) \cdot (x_{i+1} - x_i) > 0,$$

assuming modular indexing. The space of tame configurations is invariant with respect to rotations and dilatations, and its image in the shape space $\text{conf}_k = \mathbb{CP}^{k-2}$ is an open contractible set (for $k = 3$ it is the upper hemisphere of S^2). We will denote both the space of tame k -configurations and the corresponding subset of the shape space as conf_k^+ .

Note that if $\mathbf{P}_n \in \text{conf}_k^+$, then there exists a step size ε small enough so that $\mathbf{P}_{n+1} \in \text{conf}_k^+$. Equivalently, the Markov dynamics on sufficiently large and tame configurations preserves tameness, for at least a while.

Define the *tamed Markov process* $\mathbf{P}_n^{(k)}$ by the algorithm:

Algorithm 2. Tamed dynamics

```

 $n \leftarrow 0, \mathbf{P}_0^{(k)} = (v_1, \dots, v_k) \in \text{conf}_k$ 
loop
   $l \leftarrow \text{rand}(S^1)$ 
   $i \leftarrow \text{argmax} \langle l, v_i \rangle, v_i \in \mathbf{P}_n^{(k)}$ 
   $v_i \leftarrow v_i + \varepsilon l$ 
   $n \leftarrow n + 1$ 
end loop

```

Algorithm 2 is just a formal generalisation of Algorithm 1 to the space of tamed configurations. While the resulting dynamics is different one can still notice that Markov processes \mathbf{P}_n and $\mathbf{P}_n^{(k)}$ are coupled on tame configurations. As one of our goals is to *disprove* that the Markov process \mathbf{P}_n can ever converge to a $k \geq 5$ -sided polygon, it is sufficient to disprove this for $\mathbf{P}_n^{(k)}$.

3 Tamed Dynamics

The tamed Markov process possesses several features that simplify its treatment considerably, compared to the original Markov process. First and foremost, the tamed dynamics stays in the space of k -sided polygonal chains, isomorphic to \mathbb{R}^{2k} . Furthermore, the fact that the number of the “growth points” is bounded directly implies that the size of the polygonal chain grows with $\Theta(t)$.

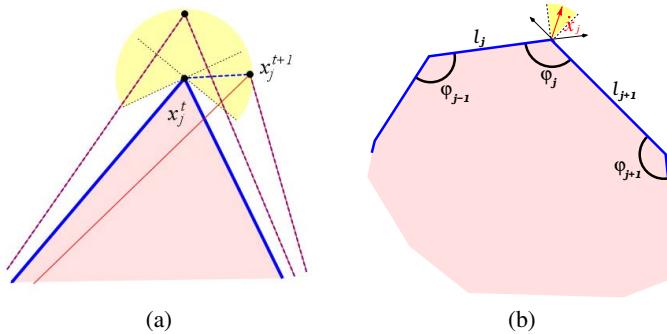


Fig. 5 The original dynamics of the RRT convex hulls and tamed dynamics coincide for any obtuse angle and are coupled for small enough increments for acute enough angles. (a) Tamed dynamics (in red) versus original dynamics (dashed blue). (b) For obtuse angles, the cone of possible directions (yellow) is spanned by the lines $(x_j - x_{j-1})^\perp$ and $(x_{j+1} - x_j)^\perp$.

Denote by $\psi(\mathbf{P})$ the perimeter of the (not necessarily convex) polygon \mathbf{P} .

Lemma 1. *The expected perimeter of $\mathbf{P}_n^{(k)}$ grows linearly: There exists some $c, C > 0$ for which*

$$\mathbb{P}\left(\psi(\mathbf{P}_n^{(k)}) < cn\right) \leq \exp(-Ct).$$

(Note that a linear upper bound on the perimeter of $\mathbf{P}_n^{(k)}$ and of \mathbf{P}_n is immediate.)

3.1 Continuous Dynamics

The stochastic process $\mathbf{P}^{(k)}$ can be viewed as a deterministic flow on the space $V_{\mathbf{P}}^k$ of k -sided polygons, subject to small noise. Indeed, by Lemma 1, the size of the polygons $\mathbf{P}_n^{(k)}$ grows linearly while the relative size of the steps decreases.

Given a polygonal k -chain \mathbf{P} , consider the expected increment

$$\Delta \mathbf{P} := \mathbb{E}(\mathbf{P}^{(k)}_{n+1} | \mathbf{P}^{(k)}_n = \mathbf{P}) - \mathbf{P}$$

(we use here the linear structure on $V_{\mathbf{P}}^k$). The following is immediate:

Lemma 2. *The function Δ commutes with rotations and is homogeneous of degree 0.*

The linear growth of the sizes of polygons $\mathbf{P}^{(k)}$ together with the homogeneity of Δ indicate that asymptotically, the expected increments are small compared to the size of $\mathbf{P}^{(k)}$. In such a situation, an approximation of the discrete stochastic dynamics by a continuous deterministic one is a natural step. This motivates the following

Definition 1. *The vector field*

$$\dot{\mathbf{P}} = \Delta \mathbf{P}$$

on $V_{\mathbf{P}}^k$ is called the CHASE dynamics.

In coordinates, the CHASE dynamics is given as follows (see Fig 5b):

Corollary 1. *Assume that \mathbf{P} consists of the vertices (x_1, \dots, x_k) , $x_i \in \mathbb{R}^2$ in cyclic order, and $(x_{i_1}, x_{i_2}, \dots, x_{i_c})$ are the vertices of \mathbf{P} on its convex hull. Then*

$$\dot{x}_{i_l} = \frac{x_{i_l} - x_{i_{l+1}}}{|x_{i_l} - x_{i_{l+1}}|} + \frac{x_{i_l} - x_{i_{l-1}}}{|x_{i_l} - x_{i_{l-1}}|}, \quad (1)$$

and $\dot{x}_i = 0$ if x_i is not on the convex hull of \mathbf{P} .

3.2 Theorem: The Discrete-Time Process Converges to the Continuous-Time System

The intuitive proximity of the $\mathbf{P}^{(k)}$ dynamics and the trajectories of CHASE flow can be made rigorous, using some standard tools. Specifically, we have shown:

Theorem 1. *Let $\mathbf{P} \in V_{\mathbf{P}}^k$, and $\mathbf{P}^{(k)}_n, 0 \leq n \leq C\lambda$ be the (random) trajectory of the tamed Markov dynamics with the initial configuration $\lambda \mathbf{P}$. Similarly, let $\mathbf{P}(n), 0 \leq n \leq \lambda^C$ be the trajectory of the CHASE dynamics starting with the same initial point $\lambda \mathbf{P}$. Then for any $C > 0$, the trajectory $\mathbf{P}^{(k)}_n / \lambda$ converges to $\mathbf{P}(n) / \lambda$ in probability in C_0 norm, as $\lambda \rightarrow \infty$.*

The proof requires some relatively extensive setup and will not be presented here. However, it is conceptually transparent: The deviations of the stochastic dynamics from the continuous one has linearly growing quadratic variations, and, by martingale large deviation results (using, for example, Azuma's inequality), one can prove the proximity of the trajectories for times of order λ , with probability exponentially close to 1. In other words, the shape of the tamed process follows the CHASE dynamical system.

The approach is close to many classical treatments of stochastically perturbed dynamical systems, see e.g.[9]. It should be noted that if one rescales the polygons $\mathbf{P}^{(k)}$ by n (to keep their linear sizes approximately constant), then the step sizes become c/n , which is the typical scale of the algorithms of stochastic approximation (compare to [15]).

4 Properties of CHASE Flow

We outline several properties of the CHASE dynamics and then analyze their equilibria.

4.1 Symmetries

High symmetries is one of the attractive features of the CHASE dynamics. Lemma 2 implies:

Proposition 1. *The CHASE dynamics defines a field of directions (a vector field defined up to point-dependent rescaling) on the shape space $V_{\mathbf{P}}^k$.*

(We will be retaining the name for the field of direction on $V_{\mathbf{P}}^k$ obtained by the projection of CHASE.)

This means that we can track the shapes in $V_{\mathbf{P}}^k$. In particular, if there would exist an exponentially stable critical point of the reduction of the CHASE dynamics to the shape space, then the tamed process would have a positive probability of converging to the corresponding shape. However, as we will show below, there are no stable equilibria in the space $V_{\mathbf{P}}^k$ for $k \geq 5$.

4.2 CHASE Is the Gradient Flow

Another interesting property of CHASE is the fact that it is a *gradient* flow, with respect to the natural Euclidean metric on $V_{\mathbf{P}}^k$:

Proposition 2. *The CHASE dynamics is the gradient of the function $\psi(\mathbf{P})$ taking a polygonal chain to the perimeter of its convex hull.*

The proof is direct: The variation of the length of the side $[x_{i_l}, x_{i_{l+1}}]$, if x_{i_l} changes infinitesimally, is the scalar product with

$$\frac{x_{i_l} - x_{i_{l+1}}}{|x_{i_l} - x_{i_{l+1}}|};$$

therefore, the claim follows.

Note that if one chooses the *other bisector*

$$\dot{x}_{i_l} = \frac{x_{i_l} - x_{i_{l+1}}}{|x_{i_l} - x_{i_{l+1}}|} - \frac{x_{i_l} - x_{i_{l-1}}}{|x_{i_l} - x_{i_{l-1}}|}, \quad (2)$$

as the direction of the velocity, then the resulting dynamics would *preserve* the perimeter and is useful in building customized Birkhoff's billiard tables; see [1].

4.3 Equilibria

According to Proposition 1, CHASE defines a vector field on V_p^k , and one would like to understand which shapes are preserved by these dynamics. Clearly, the *regular* k -polygons are preserved. Three more classes of shapes are preserved as well (see Figure 6): Trapezoids, rhombi, and *drops*, which are obtained from a regular $2k$ -polygon by extending a pair of next-to-opposite sides.

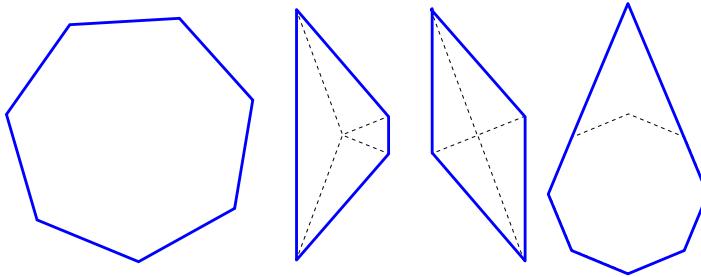


Fig. 6 From left to right, the only shapes preserved by the CHASE dynamics: regular polygons, trapezoids, rhombi, and drops. For the rhombus and trapezoid, the dashed lines show the bisectors.

As it turned out, these classes exhaust all possible shapes that are invariant under the CHASE dynamics:

Theorem 2. *The equilibrium points of CHASE in conf consist of the polygonal chains for which their convex hulls (having $i \leq k$ vertices) are regular i -sided polygons.*

Proof. Via straightforward trigonometry (using the notation shown in Figure 5b), we derive the rate of change of the angle φ_j as given by

$$\dot{\varphi}_j = \frac{\sin \varphi_{j+1} - \sin \varphi_j}{\ell_j} + \frac{\sin \varphi_{j-1} - \sin \varphi_j}{\ell_{j-1}}, \quad (3)$$

and, similarly, the rates for the lengths ℓ_j :

$$\dot{\ell}_j = 2 + \cos \varphi_j + \cos \varphi_{j-1}. \quad (4)$$

If the shape of the polygon remains invariant, then the angles are constant. This implies a system of linear equations on $s_j := \sin \varphi_j$'s. It is immediate that this system is identical to the conditions on stationary probabilities on a continuous time Markov chain on the circular graph with transition rates $1/\ell_j$. As this auxiliary Markov chain is manifestly ergodic, such stationary probabilities are unique and therefore the solutions are spanned by the vectors of $(s_j)_j = (1, 1, \dots, 1)$. Hence, all $\sin \varphi_j$ are equal, and that therefore for some φ , all of φ_j are equal to either φ or $\pi - \varphi$.

Note that this condition implies that one of the following holds: 1) There are three acute angles, forcing the polygon to be a regular triangle; 2) there are two acute angles equal to $\varphi < \pi/2$, and two complementary angles of $\pi - \varphi$, forcing the polygon to be either a trapezoid or a rhombus; 3) there is one acute angle φ , and all the remaining angles are equal to $\pi - \varphi$; 4) all angles are obtuse.

Quadrangles are realizable with the angle φ being a continuous shape parameter. For the last two cases, the preservation of the shape implies that the ratios of the lengths ℓ_j/ℓ_{j+1} remains constant, and therefore equal to the ratios of $\dot{\ell}_j/\dot{\ell}_{j+1}$. Combining this with (4) implies that the ratios of the lengths ℓ_i/ℓ_j can take only the following values (in the case of one acute angle):

$$1, \frac{1 \mp \cos \varphi}{1 \pm \cos \varphi}, (1 \pm \cos \varphi)^{\pm 1}.$$

This immediately leads to the conclusion that only the drop shape can satisfy this condition.

For the case in which all the angles are equal, the equality of all side lengths is obvious. \square

4.4 Regular Polygons Are Unstable

As discussed in Section 2.5, the space of shapes of k -sided polygonal chains is naturally isomorphic to $k - 2$ -dimensional complex projective space and has therefore (real) dimension $2k - 4$.

A natural coordinate frame on a (everywhere dense) chart in this space is given by the *lengths of the sides* of the k -gon, normalized to $\sum \ell_j = 1$ (this gives $k - 1$ coordinates) and the collection of angles φ_j satisfying the condition $\sum \varphi_j = (k - 2)\pi$. The condition of closing the polygon implies that only $(k - 3)$ of these angles are independent.

Under the CHASE dynamics, the vertices of the k -sided polygon move according to (2). The next theorem establishes that the regular k -sided polygon, although a fixed point of CHASE dynamics, is *unstable* in the linear approximation.

Note that the vector field on the shape space is defined only up to multiplication by a positive smooth function. However, one can readily verify that the linearization of the vector field at an equilibrium point is unaffected by this ambiguity.

Theorem 3. For the linearization of the CHASE dynamics near the k -regular polygonal chain, the $2k - 4$ -dimensional space of k -sided shapes has exactly one $(k - 3)$ -dimensional stable subspace (tangent to the manifold $\{\varphi = \text{const}\}$) and one $(k - 1)$ -dimensional invariant subspace, unstable for $k \geq 5$.

Proof. One can check immediately that under the CHASE evolution, the sides of the k -gons with all angles equal (to $\pi(1 - 2/k)$) move parallel to themselves. This proves the invariance of $\{\varphi = \text{const}\}$ and also the fact that (when considered in the space of k -sided polygons) the lengths are all growing linearly with the same speed. Therefore, their ratios asymptotically tend to 1, proving the first claim.

It follows that in the (ℓ, φ) frame on conf , the linearization of the CHASE vector field is block-upper-triangular:

$$J = \begin{pmatrix} J_\ell & * \\ 0 & J_\varphi \end{pmatrix}.$$

Using (3) we compute J_φ , to obtain a cyclic tri-diagonal matrix:

$$J_\varphi = -\frac{2\cos(\frac{(k-2)\pi}{k})}{\ell} \begin{pmatrix} 2 & -1 & 0 & \cdots & 0 & -1 \\ -1 & 2 & -1 & 0 & \cdots & 0 \\ 0 & -1 & 2 & -1 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & -1 & 2 & -1 \\ -1 & 0 & \cdots & 0 & -1 & 2 \end{pmatrix};$$

therefore,

$$\text{spec}(J_\varphi)_j = -\frac{4\cos(\frac{(k-2)\pi}{k})}{\ell} \left(1 - \cos\left(\frac{2\pi j}{k}\right) \right),$$

which belongs to $(0, \infty)$ if $k \geq 5$. This proves the claim. \square

4.5 Stability of Triangles

A corollary of the proof of Theorem 3 implies that the regular triangles, under the CHASE dynamics, are stable. This result is, however, relatively irrelevant, because the Markov chain \mathbf{P}_n is rather different from the tamed dynamics (which is approximated by CHASE): the triangles acquire extra vertices, and, although the growth of the resulting polygons might or might not be slowed down compared to the tamed process, we are currently unable to analyze it.

For this reason we present a simple analysis that is somewhat more general than CHASE dynamics, in which the averaged speed of an apex at j -th point ($j = 1, 2, 3$) has an outward velocity $v = v(\varphi_i/2)$ directed along the bisector, but having a general form (not necessarily $2\cos\varphi/2$, as in the case of CHASE).

We write $\alpha = \varphi_1/2$, $\beta = \varphi_2/2$, $\gamma = \varphi_3/2$. As above, we find that

$$\dot{\alpha} = \frac{1}{\sin \beta} (v(\gamma) \sin \gamma - v(\alpha) \sin \alpha) + \frac{1}{\sin \gamma} (v(\beta) \sin \beta - v(\alpha) \sin \alpha). \quad (5)$$

Denote $g(\alpha) = v(\alpha) \sin \alpha$. Then, the condition on the stationary point obtains the form

$$g(\alpha) = g(\beta) = g(\gamma).$$

Differentiating (5) with respect to α and β , we obtain linear stability conditions on stationary point. Let $J = (J_{i,j})$ be the linear part of (5). The linear stability conditions then have the form

$$0 \geq \text{tr}(J) = - \left(\frac{g'(\alpha)(\sin \beta + \sin \gamma)}{\sin \beta \sin \gamma} + \frac{g'(\beta)(\sin \alpha + \sin \gamma)}{\sin \alpha \sin \gamma} + \frac{g'(\gamma)(\sin \beta + \sin \alpha)}{\sin \beta \sin \alpha} \right)$$

and

$$0 \leq \det(J) = \frac{(g'(\alpha)g'(\beta) + g'(\beta)g'(\gamma) + g'(\gamma)g'(\alpha))(\sin \alpha + \sin \beta + \sin \gamma)}{\sin \alpha \sin \beta \sin \gamma}.$$

In particular, if $g'(\pi/3) > 0$, then the regular triangles are stable.

5 Conclusions

5.1 Summary

Summarizing, the results we presented lend theoretical support to the experimentally observed phenomena: The CHASE dynamics, which is approximating the original Markov chain well for large polygons near k -sided regular ones for $k \geq 5$, is unstable. Although we do not know what is the correct approximation for polygons close (in some sense) to large triangles, a reasonable CHASE-like approximation is stable near regular triangles. The problem of *proving* the asymptotic symmetry breaking remains open.

5.2 Higher Dimensions

One can look at the analogous problem in higher-dimensional setting. Experiments show that the convex hulls of RRTs in \mathbb{R}^d form approximately a regular d -simplex. We do not have, yet, any results similar to our planar case.

5.3 Case $k \rightarrow \infty$

One can try to approximate the evolution of convex hull of the RRT near the acute angle in a more refined way by considering the limiting case of CHASE for the infinitely many vertices for the initial shape. Without going into details, we just remark here, that in a natural parametrization, the CHASE evolution is described in this limit by the one-dimensional Boussinesq equation

$$\dot{\varphi} = (\varphi^2)''.$$

References

1. Baryshnikov, Y., Zharnitsky, V.: Sub-Riemannian geometry and periodic orbits in classical billiards. *Math. Res. Lett.* 13(4), 587–598 (2006)
2. Ballerini, M., et al.: Interaction ruling animal collective behavior depends on topological rather than metric distance: Evidence from a field study. *Proc. National Academy of Sciences* 105, 1232–1237 (2008)
3. Glover, J., Rus, D., Roy, N., Gordon, G.: Robust models of object geometry. In: Proceedings of the IROS Workshop on From Sensors to Human Spatial Concepts, Beijing, China (2006)
4. Hsu, D., Kavraki, L.E., Latombe, J.-C., Motwani, R., Sorkin, S.: On finding narrow passages with probabilistic roadmap planners. In: Agarwal, P., et al. (eds.) *Robotics: The Algorithmic Perspective*, pp. 141–154. A.K. Peters, Wellesley (1998)
5. Hsu, D., Latombe, J.-C., Motwani, R.: Path planning in expansive configuration spaces. *International Journal Computational Geometry & Applications* 4, 495–512 (1999)
6. Karaman, S., Frazzoli, E.: Sampling-based algorithms for optimal motion planning. *International Journal of Robotics Research* 30(7), 846–894 (2011)
7. Kendall, D.G., Barden, D., Carne, T.K., Le, H.: *Shape and Shape Theory*. John Wiley and Sons (1999)
8. Knutson, A.: The symplectic and algebraic geometry of Horn’s problem. *Linear Algebra Appl.* 319(1-3), 61–81 (2000)
9. Kushner, H.J., Yin, G.G.: *Stochastic Approximation and Recursive Algorithms and Applications*. Springer (2003)
10. Lamiraux, F., Laumond, J.-P.: On the expected complexity of random path planning. In: Proceedings IEEE International Conference on Robotics & Automation, pp. 3306–3311 (1996)
11. LaValle, S.M.: Rapidly-exploring random trees: A new tool for path planning. TR 98-11, Computer Science Dept., Iowa State University (October 1998)
12. LaValle, S.M.: Robot motion planning: A game-theoretic foundation. *Algorithmica* 26(3), 430–465 (2000)
13. LaValle, S.M.: *Planning Algorithms*. Cambridge University Press, Cambridge (2006), also available at <http://planning.cs.uiuc.edu/>
14. Nechushtan, O., Raveh, B., Halperin, D.: Sampling-Diagram Automata: A Tool for Analyzing Path Quality in Tree Planners. In: Hsu, D., Isler, V., Latombe, J.-C., Lin, M.C. (eds.) *Algorithmic Foundations of Robotics IX. STAR*, vol. 68, pp. 285–301. Springer, Heidelberg (2010)
15. Robbins, H., Monro, S.: A stochastic approximation method. *Ann. Math. Statistics* 22, 400–407 (1951)
16. Yu, J., Liberzon, D., LaValle, S.M.: Rendezvous Wihtout Coordinates. *IEEE Transactions on Automatic Control* 57(2), 421–434 (2012)

Efficient Collision Checking in Sampling-Based Motion Planning

Joshua Bialkowski, Sertac Karaman, Michael Otte, and Emilio Frazzoli

Abstract. Collision checking is generally considered to be the primary computational bottleneck in sampling-based motion planning algorithms. We show that this does not have to be the case. More specifically, we introduce a novel way of implementing collision checking in the context of sampling-based motion planning, such that the amortized complexity of collision checking is negligible with respect to that of the other components of sampling-based motion planning algorithms. Our method works by storing a lower bound on the distance to the nearest obstacle of each normally collision-checked point. New samples may immediately be determined collision free—without a call to the collision-checking procedure—if they are closer to a previously collision-checked point than the latter is to an obstacle. A similar criterion can also be used to detect points inside of obstacles (i.e., points that are in collision with obstacles). Analysis proves that the expected fraction of points that require a call to the normal (expensive) collision-checking procedure approaches zero as the total number of points increases. Experiments, in which the proposed idea is used in conjunction with the RRT and RRT* path planning algorithms, also validate that our method enables significant benefits in practice.

1 Introduction

Sampling-based algorithms are a popular and general approach for solving high-dimensional motion planning problems in robotics, computer graphics, and synthetic biology [1, 6, 10]. The main idea is to construct collision-free trajectories by joining points sampled from the state space, thus avoiding computations based on an explicit representation of the obstacles. The main components of sampling-based algorithms are: (i) a sampling scheme; (ii) a collision-checking function that determines if a point is in collision, given an obstacle set; (iii) a proximity search function

Joshua Bialkowski · Sertac Karaman · Michael Otte · Emilio Frazzoli
Massachusetts Institute of Technology, 77 Massachusetts Ave., Cambridge MA 02139
e-mail: {jbialk, sertac, ottemw, frazzoli}@mit.edu

that returns “neighbors” to a point, given a point set; and (iv) a local planning function that returns a trajectory between two given points.

The vast body of literature on sampling-based motion planning contains many algorithms that implement the aforementioned four components in different ways. Arguably, the main paradigms are the Probabilistic RoadMap (PRM) [9] and the Rapidly-exploring Random Tree (RRT) [12] algorithms, aimed respectively to multi- and single-query problems. Both PRM and RRT are known to be probabilistically complete, i.e., if a (robust) solution exists, then a solution will almost surely be found as the number of samples increases. Other algorithms use variations on PRM and RRT to improve performance, e.g., [7, 5] bias sampling based on collision-checking results, [4] delay collision checks until needed (LazyPRM), and [8] provide asymptotic optimality guarantees on path quality (PRM* and RRT*).

Collision checking is widely considered the primary computational bottleneck of sampling-based motion planning algorithms (e.g., see [11]). Our main contribution is to show that this does not have to be the case. We introduce a novel collision checking implementation that has negligible amortized complexity vs. the proximity searches that are already at the core of sampling-based motion planning. As a consequence, proximity searches are identified as the main determinant of complexity in sampling-based motion planning algorithms, rather than collision checking.

Traditionally, collision checking is handled by passing a point query to a “Boolean black box” collision checker that returns either true or false depending on if the point is in collision with obstacles or not, respectively. In this paper, we place a stronger requirement on the collision-checking procedure, which is now assumed to return a lower bound on the minimum distance to the obstacle set. Although computing such a bound is harder than checking whether a point is in collision or not, leveraging this extra information allows us to skip explicit collision checks for a large fraction of the subsequent samples. Indeed, the probability of calling the explicit collision checking procedure, and hence the amortized computational complexity due to collision checking, converges to zero as the number of samples increases.

The rest of this paper is organized as follows: In Section 2 we introduce notation and state our problem and main results. In Section 3 we present our new collision checking method. In Section 4 we analyze the expected fraction of samples that will require a collision distance query and calculate the expected runtime savings of the proposed approach (e.g., for RRT, PRM, and many of their variants). In Section 5 we demonstrate performance improvement when used with RRT and RRT*. In Section 6 we draw conclusions and discuss directions for future work.

2 Notation, Problem Statement, and Main Results

Let $X = (0, 1)^d$ be the *configuration space*, where $d \in \mathbb{N}$, $d \geq 2$ is the dimension of the space. Let X_{obs} be the *obstacle region*, such that $X \setminus X_{\text{obs}}$ is an open set, and denote the *obstacle-free space* as $X_{\text{free}} = \text{cl}(X \setminus X_{\text{obs}})$, where $\text{cl}(\cdot)$ denotes the closure of a set. The *initial condition* x_{init} is an element of X_{free} , and the *goal region*

X_{goal} is an open subset of X_{free} . A path planning problem is defined by a triplet $(X_{\text{free}}, x_{\text{init}}, X_{\text{goal}})$. For the sake of our discussion, we will assume that sampling-based algorithms take as input a path planning problem $(X_{\text{free}}, x_{\text{init}}, X_{\text{goal}})$, and an integer $n \in \mathbb{N}$, and return a graph $G_n = (V_n, E_n)$, where $V_n \subset X_{\text{free}}$, $\text{card}(V_n) \leq n$, and $E_n \subset V_n \times V_n$. The solution of the path planning problem (if found) can be recovered from such a graph, e.g., using standard shortest-path algorithms. Sampling-based algorithms map a sequence of sampled points ω to a sequence of graphs. Let $G_n(\omega) = (V_n(\omega), E_n(\omega))$ denote the output of the algorithm, given ω .

The problem we address in this paper is how to reduce the complexity of sampling-based motion planning algorithms, i.e., the (expected) number of operations needed to compute G_n . In particular, we consider incremental complexity—the expected difference in the total number of operations needed to compute $G_{n+1}(\omega)$ and $G_n(\omega)$. In the sequel, we characterize complexity based on the number of samples n and the environment description X_{obs} , while keeping the dimension d of the space a constant. For convenience, we assume stochastic sampling that draws uniformly and independently from X . Our results extend to deterministic and/or non-uniform sampling procedures, as long as the sampling procedure satisfy the technical conditions required for probabilistic or resolution completeness [11].

The computational complexity of a sampling-based algorithm can be decomposed in terms of the complexity of its primitive operations. The complexity of drawing a sample is bounded by a constant c_{sample} . The complexity of a collision check is bounded by $c_{\text{cc}}(X_{\text{obs}})$ a function that depends only on the description of the environment—note that the expected complexity of checking collisions with n_{obs} convex obstacles is $O(\log(n_{\text{obs}})^d)$ [13]. The complexity of (approximate) proximity searches that return the $O(\log n)$ nearest elements to a query point from a set of cardinality n is $O(\log n)$ [13]. The latter applies to k -nearest neighbor searches (k fixed or scaling as $\log n$), and range searches among uniform random samples for those points within a ball of volume scaling as $\log(n)/n$.

The complexity of the local planner is bounded by a constant c_{plan} that does not depend on the environment description. For convenience we also include in c_{plan} all other constant time bookkeeping operations (cost updates, graph rewiring, etc.) that are performed for each new edge. Each candidate path generated by the local planner must also be checked for collisions, with complexity bounded by $c_{\text{path}}(X_{\text{obs}})$ a function that depends only on the description of the environment.

The expected incremental complexity of sampling-based motion planning algorithms can be evaluated by examining the work induced by each new sample:

- RRT: Find the nearest neighbor, generate a new point, check the new point for collision, and connect the new point to the nearest neighbor.
- k -PRM: Collision check the sample, find and connect to its k -nearest neighbors.
- RRT*, PRM*: Collision check the sample, find the neighbors within a ball of volume scaling as $\log(n)/n$ —or, equivalently, the $(k \log n)$ -nearest neighbors, for a fixed k —and connect to them.

Table 1 Asymptotic bounds on the expected incremental complexity of some standard sampling-based motion planning algorithms

| Algorithm | Proximity Search | Point Collision Checking | Local Planning | Path Collision Checking |
|------------|------------------|--------------------------|----------------------|-------------------------------|
| RRT | $O(\log n)$ | $c_{cc}(X_{obs})$ | c_{plan} | $c_{path}(X_{obs})$ |
| k -PRM | $O(\log n)$ | $c_{cc}(X_{obs})$ | $k c_{plan}$ | $k c_{path}(X_{obs})$ |
| RRT*, PRM* | $O(\log n)$ | $c_{cc}(X_{obs})$ | $O(\log n) c_{plan}$ | $O(\log n) c_{path}(X_{obs})$ |

Table 1 summarizes the main contributions to the expected incremental complexity of these algorithms, in their standard implementation, based on collision-checking queries. Note that the local planning and path checking only occur if the new sample is not in collision with obstacles.

It is important to note that the asymptotic bounds may be misleading. In practice, the cost of collision checking the new sample and a new candidate connection $c_{cc}(X_{obs}) + c_{path}(X_{obs})$ often effectively dominates the incremental complexity, even for very large n . Hence, collision checking is typically regarded as the computational bottleneck for sampling-based motion planning algorithms. The cost of collision checking is even more evident for algorithms such as RRT* and PRM*, where collision checks are carried out for each of the $O(\log n)$ candidate connections at each iteration. Moreover, the ratio of the incremental complexity of RRT* to that of RRT depends on the environment via $c_{path}(X_{obs})$, and is potentially large, although constant with respect to n .

In our approach, we replace standard collision checks with approximate minimum-distance computations. The complexity of computing a non-trivial lower bound on the minimum distance of a point from the obstacle set is also bounded by a function that depends on the description of the environment. In particular, we will require our collision checking procedure to return a lower bound \bar{d} on the minimum distance d^* that satisfies $\alpha d^* \leq \bar{d} \leq d^*$, for some $\alpha \in (0, 1]$. The computational cost of such a procedure will be indicated with $c_{dist}(X_{obs})$. While (approximate) distance queries are more expensive than collision checking, the ratio $c_{dist}(X_{obs})/c_{cc}(X_{obs})$ is bounded by a constant, independent of n .

The output of the minimum-distance computations is stored in a data structure that augments the standard graph constructed by sampling-based algorithms. Using this information, it is possible to reduce the number of explicit collision checks and minimum-distance computations that need to be performed. As we will later show, using our approach to modify the standard algorithms results in the asymptotic bounds on expected incremental complexity summarized in Table 2, where p_{cc} is a function such that $\limsup_{n \rightarrow \infty} p_{cc}(n) = 0$. Again, the local planning and path checking only occur if the new sample is not in collision with obstacles.

Inspection of table 2 reveals that the incremental complexity of our version of RRT and k -PRM is less sensitive to the cost of collision checking. Moreover, the asymptotic ratio of the incremental complexity of our version of RRT* and RRT is a constant that does not depend on the environment.

Table 2 Asymptotic bounds on the expected incremental complexity of some sampling-based motion planning algorithms, modified according to the proposed approach; p_{cc} is a function, such that $\limsup_{n \rightarrow \infty} p_{cc}(n) = 0$

| Mod. Algorithm | Prox. Search | Collision Checking | Local Planning | Path Collision Checking |
|----------------|--------------|---|----------------------------|---|
| RRT | $O(\log n)$ | $O(\log n) + p_{cc}(n)c_{cc}(X_{\text{obs}})$ | c_{plan} | $p_{cc}(n)c_{\text{path}}(X_{\text{obs}})$ |
| k -PRM | $O(\log n)$ | $O(\log n) + p_{cc}(n)c_{cc}(X_{\text{obs}})$ | kc_{plan} | $k p_{cc}(n)c_{\text{path}}(X_{\text{obs}})$ |
| RRT*, PRM* | $O(\log n)$ | $O(\log n) + p_{cc}(n)c_{cc}(X_{\text{obs}})$ | $O(\log n)c_{\text{plan}}$ | $O(\log n)p_{cc}(n)c_{\text{path}}(X_{\text{obs}})$ |

3 Proposed Algorithm

Before describing our proposed modification to the collision checking procedure in Section 3.1, we first formalize its primitive procedures and data structure.

Sampling: Let $\text{Sample} : \omega \mapsto \{\text{Sample}_i(\omega)\}_{i \in \mathbb{N}} \subset X$ be such that the random variables Sample_i , $i \in \mathbb{N}$, are independent and identically distributed (i.i.d.). The samples are assumed to be from a uniform distribution, but results extend naturally to any absolutely continuous distribution with density bounded away from zero on X .

Nearest Neighbors: Given a finite point set $S \subset X$ and a point $x \in X$, the function $\text{Nearest} : (S, x) \mapsto s \in S$ returns the point in S that is closest to x ,

$$\text{Nearest}(S, x) := \underset{s \in S}{\operatorname{argmin}} \|x - s\|,$$

where $\|\cdot\|$ is a metric (e.g., Euclidean distance—see [12] for alternative choices). A set-valued version is also considered, $\text{kNearest} : (S, x, k) \mapsto \{s_1, s_2, \dots, s_k\}$, returns the k vertices in S that are nearest to x with respect to $\|\cdot\|$. By convention, if the cardinality of S is less than k , then the function returns S .

Near Vertices: Given a finite point set $S \subset X$, a point $x \in X$, and a positive real number $r \in \mathbb{R}_{>0}$, the function $\text{Near} : (S, x, r) \mapsto S_{\text{near}} \subseteq S$ returns the vertices in S that are inside a ball of radius r centered at x ,

$$\text{Near}(S, x, r) := \{s \in S : \|s - x\| \leq r\}.$$

Set Distance: A closed set $S \subset X$ and a point $x \in X$, SetDistance returns a non-trivial lower bound on the minimum distance from x to S , i.e., for some $\alpha \in (0, 1]$,

$$\alpha \min_{s \in S} \|s - x\| \leq \text{SetDistance}(S, x) \leq \min_{s \in S} \|s - x\|.$$

Segment Collision Test: Given points $x, y \in X$, $\text{CFreePath}(x, y)$ returns **True** if the line segment between x and y lies in X_{free} , i.e., $[x, y] \subset X_{\text{free}}$, and **False** otherwise.

Data Structure: We achieve efficient collision checking by storing additional information in the graph data structure that is already used by most sampling-based

algorithms. Specifically, we use the “augmented graph” $AG = (V, E, C_{\text{free}}, C_{\text{obs}}, \text{Dist})$, where $V \subset X_{\text{free}}$ and $E \subset V \times V$ are the usual vertex and edge sets. The sets $C_{\text{free}} \subset X_{\text{free}}$ and $C_{\text{obs}} \subset X_{\text{obs}}$ are sets of points for which an explicit collision check has been made. For points in C_{free} or C_{obs} the map $\text{Dist} : C_{\text{free}} \cup C_{\text{obs}} \rightarrow \mathbb{R}_{\geq 0}$ stores the approximate minimum distance to the obstacle set or to the free space, respectively. The vertices V and edges E are initialized according to the particular sampling-based algorithm in use. The sets C_{free} and C_{obs} are initialized as empty sets.

3.1 Modified Collision Checking Procedures

Using the augmented graph data structure allows us to modify the collision checking procedures as shown in Algorithms 1 and 2, respectively for points and paths. For convenience, we assume that the augmented data structure AG can be accessed directly by the collision checking procedures, and do not list it as an input.

Point Collision Test: Given a point $x \in X$, the function $\text{CFreePoint}(x)$ returns `True` if $x \in X_{\text{free}}$, and `False` otherwise. When a new sample q is checked for collision, we first check if we can quickly determine whether it is in collision or not using previously computed information, lines 1.1-1.6. In particular, we use the vertex $v_{\text{near}} \in C_{\text{free}}$ that is nearest to q (found on line 1.1). If q is closer to v_{near} than the v_{near} is to an obstacle, then clearly q is collision free, lines 1.3-1.4. Otherwise, we find the vertex $o_{\text{near}} \in C_{\text{obs}}$ that is nearest to q , line 1.2. If q is closer to o_{near} than o_{near} is to the free space, then clearly q is in collision, line 1.5-1.6.

If these two checks prove inconclusive, then a full collision check is performed using set distance computations. First, one can compute the approximate minimum distances from q to the obstacle set, and to the free set, respectively indicated with d_{obs} and d_{free} , lines 1.7-1.8. If $d_{\text{obs}} > 0$, q is in X_{free} and is added to the set C_{free} of vertices that have been explicitly determined to be collision-free, lines 1.9-1.10. Furthermore, $\text{Dist}(q)$ is set to be equal to d_{obs} , line 1.11. Otherwise, q is in X_{obs} , and is added to the set C_{obs} , with $\text{Dist}(q) = d_{\text{free}}$, lines 1.13-1.15.

Path Set Collision Test: Given a vertex set $S \subseteq V$ and point x , the function $\text{BatchCFreePath}(S, x)$ returns a set of edges $H \subseteq (V \cup \{x\}) \times (V \cup \{x\})$ such that for all $h = (x, y) \in H$, $\text{CollisionFreePath}(x, y)$ evaluates to `True`. The form of S depends on the particular planning algorithm that is being used, e.g.,

- RRT: $S = \{\text{Nearest}(V, x)\}$ is a singleton containing the nearest point to x in V .
- k -PRM: $S = k\text{Nearest}(V, x, k)$ contains up to k nearest neighbors to x in V .
- RRT* and PRM*: S contains $O(\log n)$ points—either the points within a ball of volume scaling as $\log(n)/n$ centered at x , or the $(k \log N)$ -nearest neighbors to x .

We assume $\text{CFreePoint}(x)$ is called prior to $\text{BatchCFreePath}(S, x)$, and thus x is collision-free. For each pair (s, x) , $s \in S$, the first step is to check whether the segment $[s, x]$ can be declared collision-free using only the information already available in AG , lines 2.2-2.5. Let $v_{\text{near}} \in C_{\text{free}}$ be the vertex in C_{free} that is nearest to x , line 2.2. If both s and x are closer to v_{near} than v_{near} is to the obstacle set, then clearly the segment $[s, x]$ is collision free, lines 2.4-2.5 ($\|x - v_{\text{near}}\| \leq \text{Dist}(v_{\text{near}})$

Algorithm 1: CFreePoint(q)

```

1  $v_{\text{near}} \leftarrow \text{Nearest}(C_{\text{free}}, q);$ 
2  $o_{\text{near}} \leftarrow \text{Nearest}(C_{\text{obs}}, q);$ 
3 if  $\|q - v_{\text{near}}\| \leq \text{Dist}(v_{\text{near}})$  then
4   return True
5 else if  $\|q - o_{\text{near}}\| \leq \text{Dist}(o_{\text{near}})$ 
     then
6   return False
7  $d_{\text{obs}} \leftarrow \text{SetDistance}(X_{\text{obs}}, q);$ 
8  $d_{\text{free}} \leftarrow \text{SetDistance}(X_{\text{free}}, q);$ 
9 if  $d_{\text{obs}} > 0$  then
10   $C_{\text{free}} \leftarrow C_{\text{free}} \cup \{q\};$ 
11   $\text{Dist}(q) \leftarrow d_{\text{obs}};$ 
12  return True
13 else
14   $C_{\text{obs}} \leftarrow C_{\text{obs}} \cup \{q\};$ 
15   $\text{Dist}(q) \leftarrow d_{\text{free}};$ 
16  return False

```

Algorithm 2: BatchCFreePath(S, x)

```

1  $H \leftarrow \emptyset;$ 
2  $v_{\text{near}} \leftarrow \text{Nearest}(C_{\text{free}}, x);$ 
3 foreach  $s \in S$  do
4   if  $\|s - v_{\text{near}}\| \leq \text{Dist}(v_{\text{near}})$  then
5      $H \leftarrow H \cup \{(s, x)\};$ 
6   else if CFreePath( $s, x$ ) then
7      $H \leftarrow H \cup \{(s, x)\};$ 
8 return  $H$ 

```

Algorithm 3:ApproxCDistPolytope(A, c, q)

```

1  $d \leftarrow (Aq - c);$ 
2  $d_{\min} = -\|d\|_{\infty};$ 
3  $d_{\max} = \|d\|_{\infty};$ 
4 if  $d_{\max} > 0$  then
5   return (False,  $d_{\max}$ );
6 else
7   return (True,  $-d_{\min}$ );

```

is automatic, given that CFreePoint(x) has already been called). If this check is inconclusive, then a full collision check is performed, lines 2.6-2.7.

Approximate Set Collision Distance: The set distance computation method depends on the choice of an obstacle index. For two dimensional polygonal obstacles, a segment Voronoi diagram is an index for which set distance computation is efficient and exact. For obstacles which are polytopes represented as the set $\{x : Ax \leq c\}$, Algorithm 3 is sufficient for calculating a lower bound on the set distance.

Efficient Collision Checking: We now illustrate how to improve the efficiency of standard sampling-based motion planning algorithms. As an example, Algorithms 4 and 5 show modified versions of the RRT and PRM* (pseudo-code based on [8]). AG is initialized on lines 4.1/5.1. Standard point collision checks are replaced with CFreePoint (i.e., Algorithm 1), lines 4.4/5.4. The code generating neighbor connections from a new sample is modified to generate those connections in a batch manner via a single call to BatchCFreePath (i.e., Algorithm 2), lines 4.6/5.7.

4 Analysis

Suppose ALG is a sampling-based motion planning algorithm. Let $I_{\text{point}}(n)$ denote the indicator random variable for the event that the SetDistance procedure in Algorithm 1 is called by ALG in the iteration in which the n -th sample is drawn. Similarly, define $I_{\text{path}}(n)$ as the indicator random variable for the event that the

| Algorithm 4: Modified RRT | Algorithm 5: Modified PRM* |
|---|---|
| <pre> 1 $V \leftarrow \{x_{\text{init}}\}; E \leftarrow \emptyset; C_{\text{free}} \leftarrow \emptyset;$ $C_{\text{obs}} \leftarrow \emptyset;$ 2 for $i = 1, \dots, n - 1$ do 3 $x_{\text{rand}} \leftarrow \text{Sample}_i(\omega);$ 4 if $\text{CFreePoint}(x_{\text{rand}})$ then 5 $x_{\text{nearest}} \leftarrow \text{Nearest}(V, x_{\text{rand}});$ 6 $H \leftarrow \text{BatchCFreePath}($ $\{x_{\text{nearest}}\}, x_{\text{rand}});$ 7 if $H \neq \emptyset$ then 8 $V \leftarrow V \cup x_{\text{rand}};$ 9 $E \leftarrow E \cup H;$ 10 return $G = (V, E);$ </pre> | <pre> 1 $V \leftarrow \{x_{\text{init}}\}; E \leftarrow \emptyset; C_{\text{free}} \leftarrow \emptyset;$ $C_{\text{obs}} \leftarrow \emptyset;$ 2 for $i = 1, \dots, n - 1$ do 3 $x_{\text{rand}} \leftarrow \text{Sample}_i(\omega);$ 4 if $\text{CFreePoint}(x_{\text{rand}})$ then 5 $V \leftarrow V \cup \{x_{\text{rand}}\};$ 6 foreach $v \in V$ do 7 $S \leftarrow \text{Near}(V, v, \gamma_{\text{PRM}}(\log(n)/n)^{1/d});$ 8 $H \leftarrow \text{BatchCFreePath}(S \setminus \{v\}, v);$ 9 $E \leftarrow E \cup H;$ return $G = (V, E);$ </pre> |

CFreePath procedure in Algorithm 2 is called by ALG in the iteration in which the n -th sample is drawn. Define $I(n) := I_{\text{point}}(n) + I_{\text{path}}(n)$, and let $p_{\text{cc}}(n) = \mathbb{E}[I(n)]$. We prove our main result for algorithms satisfying the following assumptions:

- (A1) ALG calls the CFreePoint procedure $O(1)$ times per iteration;
- (A2) At the n -th iteration, ALG calls the batch CFreePath procedure with $O(\log n)$ paths, and all such paths lie inside a ball of radius $o(n^{-1/(d+1)})$ as $n \rightarrow \infty$.

Our main result is stated in the following theorem.

Theorem 1. *Suppose ALG satisfies Assumptions (A1) and (A2). Then ALG , implemented using the proposed collision-checking algorithm, has zero asymptotic expected incremental set-distance complexity, i.e.,*

$$\limsup_{n \rightarrow \infty} p_{\text{cc}}(n) = 0.$$

Note that all algorithms in Table 2 satisfy Assumptions (A1) and (A2). Hence, the results summarized in Table 2 can be deduced from Theorem 1.

The rest of this section is devoted to the proof of Theorem 1. The key idea behind the proof is to divide up the state space into several cells. This particular discretization allows us to compute a bound on the expected number of samples that require Algorithm 1 to call the set-distance procedure. We show that this number grows slower than n , which implies Theorem 1.

Consider a partitioning of the environment into cells, where the size and number of cells used in the partition grows with the number of samples n . More precisely, divide up the configuration space $[0, 1]^d$ into openly-disjoint hypercube cells with edge length $l_n := n^{-\frac{1}{d+1}}$. Given $\mathbf{z} \in \mathbb{Z}^d$, we define the cell with index \mathbf{z} as the L^∞ ball of radius l_n centered at $l_n \mathbf{z}$, i.e., $C_n(\mathbf{z}) := \{x' \in X : \|x' - l_n \mathbf{z}\|_\infty \leq l_n\}$, where $l_n \mathbf{z}$ is the scalar-vector multiplication of \mathbf{z} by l_n , and $\|\cdot\|_\infty$ is the infinity norm. Let $\mathbf{Z}_n \subset \mathbb{Z}^d$

denote the smallest set for which $\mathbf{C}_n := \{C_n(\mathbf{z}) : \mathbf{z} \in \mathbf{Z}_n\}$ covers the configuration space, i.e., $X \subseteq \bigcup_{\mathbf{z} \in \mathbf{Z}_n} C_n(\mathbf{z})$. Clearly, \mathbf{Z}_n is a finite set, since X is compact.

Let γ_u denote the maximum distance between two points in the unit hypercube using the distance metric $\|\cdot\|$ employed in the `SetDistance` procedure. For instance, if $\|\cdot\|$ is the usual Euclidean metric (L_2 norm), then $\gamma_u = \sqrt{d}$; if $\|\cdot\|$ is the L_∞ norm, then $\gamma_u = 1$.

We group the cells in \mathbf{C}_n into two disjoint subsets, namely the *boundary cells* \mathbf{B}_n and the *interior cells* \mathbf{I}_n . Let \mathbf{B}'_n denote the set of all $C_n(\mathbf{z}) \subset \mathbf{C}_n$ that include a part of the obstacle set boundary, i.e., $C_n(\mathbf{z}) \cap \partial X_{\text{obs}} \neq \emptyset$. Then \mathbf{B}_n contains exactly those cells that are within a cell-distance of at most $2(\lceil(1/\alpha)\gamma_u\rceil + 1)$ to some cell in \mathbf{B}'_n ,

$$\mathbf{B}_n := \left\{ C_n(\mathbf{z}) : \|\mathbf{z} - \mathbf{z}'\|_\infty \leq 2(\lceil(1/\alpha)\gamma_u\rceil + 1) \text{ for some } \mathbf{z}' \text{ with } C_n(\mathbf{z}') \in \mathbf{B}'_n \right\},$$

where α is the constant in the constant-factor lower bound in the computation of the `SetDistance` procedure. Finally, \mathbf{I}_n is defined as exactly the set of those cells that are not boundary cells, i.e., $\mathbf{I}_n := \mathbf{C}_n \setminus \mathbf{B}_n$. The reason behind our choice of the number $2(\lceil(1/\alpha)\gamma_u\rceil + 1)$ will become clear shortly.

Let $\lambda(\cdot)$ denote the Lebesgue measure in \mathbb{R}^d . The total number of cells, denoted by K_n , can be bounded for all n as follows¹:

$$K_n \leq \frac{\lambda(X)}{\lambda(C_n(\mathbf{z}))} = \frac{\lambda(X)}{\left(n^{-\frac{1}{d+1}}\right)^d} = \lambda(X) n^{\frac{d}{d+1}} \quad (1)$$

Notice the number of cells, K_n , is an increasing and sub-linear function of n .

Let B_n denote the number of boundary cells.

Lemma 1. *There exists a constant $c_1 > 0$ such that $B_n \leq c_1(K_n)^{1-1/d}$ for all $n \in \mathbb{N}$.*

Proof. This result follows from the fact that the obstacle boundary can be covered by $N^{(d-1)/d} = N^{1-1/d}$ cells, where N is the number of equal-size cells that cover the configuration space.

Thus the fraction of cells that are boundary cells can be bounded by

$$\frac{c_1 B_n}{K_n} = \frac{c_1 (K_n)^{1-1/d}}{K_n} = c_1 (K_n)^{-1/d} \leq c_1 (\lambda(X))^{-1/d} n^{-\frac{1}{d+1}} = c_2 n^{-\frac{1}{d+1}},$$

where c_2 is a constant.

We now bound the number of calls to the collision-distance procedure by examining the number of such calls separately for samples that fall into interior cells or boundary cells, respectively. Recall that C_{free} and C_{obs} are defined as the vertices that are explicitly checked and found to be in X_{free} and X_{obs} , respectively. Define

¹ Strictly speaking, this bound should read: $K_n \leq \lceil \lambda(X) n^{d/(d+1)} \rceil$, where $\lceil \cdot \rceil$ is the standard ceiling function (i.e., $\lceil a \rceil$ returns the smallest integer greater than a). We will omit these technical details from now on to keep the notation simple.

$C := C_{\text{free}} \cup C_{\text{obs}}$. Our key insight is summarized in the following lemma, which will be used to derive a bound on the number of points in C that fall into interior cells.

Lemma 2. *Given any algorithm in Table 2, suppose that algorithm is run with n samples using our collision checking algorithm. Let $C_n(\mathbf{z}) \in \mathbf{I}_n$ be some interior cell. Then, there exists at most one vertex from C in $C_n(\mathbf{z})$, i.e.,*

$$|C_n(\mathbf{z}) \cap C| \leq 1 \text{ for all } C_n(\mathbf{z}) \in \mathbf{I}_n.$$

Before proving Lemma 2, we introduce some additional notation and establish two intermediate results. Let $\mathbf{N}_n(\mathbf{z}) \subseteq \mathbf{C}_n$ denote the set of all cells that have a cell distance of at most $\lceil \gamma_u \rceil + 1$ to $C_n(\mathbf{z})$, i.e., $\mathbf{N}_n(\mathbf{z}) := \{C_n(\mathbf{z}') : \|\mathbf{z}' - \mathbf{z}\|_\infty \leq \lceil \gamma_u \rceil + 1\}$. $\mathbf{N}_n(\mathbf{z})$ includes the cell $C_n(\mathbf{z})$ together with all its neighbors with cell distance less than $\lceil \gamma_u \rceil + 1$. The cells in $\mathbf{N}_n(\mathbf{z})$ are such that they exclude points that are sufficiently far away from points in $C_n(\mathbf{z})$ and they include points that are sufficiently far away from the boundary of the obstacle set. The last two properties are made precise in the following two lemmas.

Lemma 3. *Any point that lies in a cell outside of $\mathbf{N}_n(\mathbf{z})$ has distance at least $\gamma_u l_n$ to any point that lies in $C_n(\mathbf{z})$, i.e., $\|x - x'\| \geq \gamma_u l_n$ for all $x \in C_n(\mathbf{z})$ and all $x' \in C_n(\mathbf{z}')$ with $C_n(\mathbf{z}') \notin \mathbf{N}_n(\mathbf{z})$.*

Proof. The claim follows immediately by the construction of $\mathbf{N}_n(\mathbf{z})$.

Lemma 4. *Any point in a cell from $\mathbf{N}_n(\mathbf{z})$ has distance at least $(\lceil (1/\alpha) \gamma_u \rceil + 1) l_n$ to any point that lies on the obstacle set boundary, i.e., $\|x - x'\| \geq (\lceil (1/\alpha) \gamma_u \rceil + 1) l_n$ for all $x \in \partial X_{\text{obs}}$ and all $x' \in C_n(\mathbf{z}')$ with $C_n(\mathbf{z}') \in \mathbf{N}_n(\mathbf{z})$.*

Proof. The interior cell $C_n(\mathbf{z})$ has a cell distance of at least $2(\lceil (1/\alpha) \gamma_u \rceil + 1)$ to any cell that intersects with the boundary of the obstacle set. Any cell in $\mathbf{N}_n(\mathbf{z})$ has a cell distance of at most $\lceil \gamma_u \rceil + 1 \leq \lceil (1/\alpha) \gamma_u \rceil + 1$ to $C_n(\mathbf{z})$. Thus, by the triangle inequality (for the cell distance function), any cell in $\mathbf{N}_n(\mathbf{z})$ has a cell distance of at least $\lceil (1/\alpha) \gamma_u \rceil + 1$ to any cell that intersects the obstacle boundary.

Now we are ready to prove Lemma 2.

Proof (Proof of Lemma 2). The proof is by contradiction. Suppose there exists two points $x_1, x_2 \in C$ that fall into $C_n(\mathbf{z})$. Suppose x_1 is added into C before x_2 . Let x_{nearest} denote the nearest point when Algorithm 1 was called with x_2 .

First, we claim that x_{nearest} lies in some cell in $\mathbf{N}_n(\mathbf{z})$. Clearly, x_{nearest} is either x_1 or it is some other point that is no farther than x_1 to x_2 . Note that x_1 and x_2 has distance at most $\gamma_u l_n$. By Lemma 3, any point that lies in a cell outside of $\mathbf{N}_n(\mathbf{z})$ has distance at least $\gamma_u l_n$ to x_2 . Thus, x_{nearest} must lie in some cell in $\mathbf{N}_n(\mathbf{z})$. However, $\text{Dist}(x_{\text{nearest}}) \geq (1/\alpha) \|x_2 - x_{\text{nearest}}\|$ by Lemma 4; In that case, x_2 should have never been added to C , even when the `SetDistance` procedure returns α -factor of the actual distance to the obstacle set boundary. Hence, we reach a contradiction.

The following lemma provides an upper bound on the *expected* number of samples that fall into boundary cells, thus an upper bound on the expected number of points in C that fall into a boundary cell.

Lemma 5. Let X_n denote a set of n samples drawn independently and uniformly from X . Let S_n denote the number of samples that fall into boundary cells, i.e.,

$$S_n := \left| \{x \in X_n : x \in C_n(\mathbf{z}) \text{ with } C_n(\mathbf{z}) \in \mathbf{B}_n\} \right|.$$

Then, there exists some constant c_3 , independent of n , such that $\mathbb{E}[S_n] \leq c_3 n^{\frac{d}{d+1}}$.

Proof. Let E_i denote the event that the i -th sample falls into a boundary cell. Let Y_i denote the indicator random variable corresponding to this event. From Lemma 1 and the discussion following it, the fraction of cells that are of boundary type is $c_2 n^{-\frac{1}{d+1}}$. Thus, $\mathbb{E}[Y_i] = \mathbb{P}(E_i) = c_2 i^{-\frac{1}{d+1}}$ and $\mathbb{E}[S_n] = \mathbb{E}[\sum_{i=1}^n Y_i] = \sum_{i=1}^n \mathbb{E}[Y_i] = \sum_{i=1}^n c_2 i^{-\frac{1}{d+1}} \leq c_2 \int_1^n x^{-\frac{1}{d+1}} dx$, where $c_2 \int_1^n x^{-\frac{1}{d+1}} dx = \frac{c_2(d+1)}{d} \left(n^{\frac{d}{d+1}} - 1 \right)$. Thus, $\mathbb{E}[S_n] \leq c_3 n^{\frac{d}{d+1}}$, where c_3 is a constant that is independent of n .

The following lemma gives an upper bound on the number of points in C .

Lemma 6. There exists a constant c_4 , independent of n , such that

$$\mathbb{E}[\text{card}(C)] \leq c_4 n^{\frac{d}{d+1}}.$$

Proof. On one hand, the number of points in C that fall into an interior cell is at most the total number of interior cells by Lemma 2, and thus less than the total number of cells K_n , which satisfies $K_n \leq \lambda(X) n^{\frac{d}{d+1}}$ (see Equation (1)). On the other hand, the expected number of points in C that fall into a boundary is no more than the expected number of samples that fall into boundary cells, which is bounded by $c_3 n^{\frac{d}{d+1}}$, where c is a constant independent of n (see Lemma 5). Thus, we conclude that $\mathbb{E}[\text{card}(C)] \leq c_4 n^{\frac{d}{d+1}}$ for some constant c_4 .

Finally, we are ready to prove Theorem 1.

Proof (Theorem 1). First, we show that p_{cc} is a non-increasing function of n using a standard coupling argument. We couple the events $\{I(n) = 1\}$ and $\{I(n+1) = 1\}$ with the following process. Consider the run of the algorithm with $n+1$ samples. Let A_n and A_{n+1} denote the events that the n th and the $(n+1)$ st samples are explicitly checked, respectively. Clearly, $\mathbb{P}(A_{n+1}) \leq \mathbb{P}(A_n)$ in this coupled process, since the first $(n-1)$ st samples are identical. Moreover, $\mathbb{P}(A_n) = \mathbb{P}(\{I(n) = 1\}) = p_{cc}(n)$ and $\mathbb{P}(A_{n+1}) = \mathbb{P}(\{I(n+1) = 1\}) = p_{cc}(n+1)$. Thus, $p_{cc}(n+1) \leq p_{cc}(n)$ for all $n \in \mathbb{N}$. Note that this implies that $\lim_{n \rightarrow \infty} p_{cc}(n)$ exists.

Next, we show that $\lim_{n \rightarrow \infty} (1/n) \sum_{k=1}^n p_{cc}(k) = 0$. Clearly, $\sum_{k=1}^n I_{\text{point}}(k) = |C|$. Hence, $\frac{\sum_{k=1}^n \mathbb{E}[I_{\text{point}}(k)]}{n} = \frac{\mathbb{E}[\sum_{k=1}^n I_{\text{point}}(k)]}{n} \leq \frac{c_4 n^{\frac{d}{d+1}}}{n} = c_4 n^{-\frac{1}{d+1}}$, where the inequality follows from Lemma 6. Similarly, $\limsup_{n \rightarrow \infty} (\mathbb{E}[I_{\text{path}}(k)] - \mathbb{E}[I_{\text{point}}(k)]) = 0$, since all paths fit into an Euclidean ball with radius $o(n^{-1/(d+1)})$, which is asymptotically smaller than the side length of each cell $l_n = n^{-1/(d+1)}$. Hence,

$$\begin{aligned} \limsup_{n \rightarrow \infty} \frac{\sum_{k=1}^n p_{cc}(k)}{n} &= \limsup_{n \rightarrow \infty} \left(\frac{\sum_{k=1}^n \mathbb{E}[I_{\text{point}}(k)] + \mathbb{E}[I_{\text{path}}(k)]}{n} \right) \\ &= \limsup_{n \rightarrow \infty} \frac{\mathbb{E}[\sum_{k=1}^n I(k)]}{n} \leq \limsup_{n \rightarrow \infty} \frac{c_4 n^{\frac{d}{d+1}}}{n} = \limsup_{n \rightarrow \infty} c_4 n^{-\frac{1}{d+1}} = 0, \end{aligned}$$

Finally, $p_{cc}(n+1) \leq p_{cc}(n)$ for all $n \in \mathbb{N}$ and $\lim_{n \rightarrow \infty} (1/n) \sum_{k=1}^n p_{cc}(k) = 0$ together imply that $\lim_{n \rightarrow \infty} p_{cc}(n) = 0$.

5 Experiments

To validate and demonstrate the utility of our method, we compared the performance of implementations of the RRT and RRT* algorithms, both with and without our proposed modification for collision checking. Our implementations are single threaded, utilize a kd-tree [3] for point-proximity (i.e., nearest neighbor and k -nearest neighbor) queries, and a segment-Voronoi hierarchy [2] for set distance queries. The experiments were run on a 1.73 GHz Intel Core i7 computer with 4GB of RAM, running Linux. We performed experiments on an environment consisting of a unit-square workspace with 150 randomly placed convex polygonal obstacles (see Figure 1). The goal region is a square with side length 0.1 units at the top right corner; the starting point is at the bottom left corner.

Figure 1 (Left) shows both the tree of trajectories generated by RRT*, and the set of collision-free balls that are stored in the augmented graph. After 2,000 samples, the collision-free balls have filled a significant fraction of the free space, leaving only a small amount of area uncovered near the obstacle boundaries. As proved in Section 4, any future sample is likely to land in a collision-free ball common to both it and its nearest neighbor, making future collision checks much less likely.

To generate timing profiles, both RRT and RRT* were run on the obstacle configuration in Figure 1 with and without the use of the new collision algorithm. In each run, both variants of both algorithms are run with the same initial random seed for the point sampling process so that the points sampled in each algorithm are the same (i.e., the sample sequence ω is the same for both algorithms).

Figure 1 (Right) illustrates the wall-clock measured runtime required to reach various tree sizes for each of these four implementations. Runtimes are averaged over 30 runs and in graph-size buckets of 1000 nodes. As expected from the amortized complexity bounds, longer runs enable the proposed approach to achieve greater savings. For example, the runtime of RRT* is reduced by 40% at 10,000 vertices, and by 70% at 100,000 vertices, vs. the baseline implementation. The runtime of RRT is reduced by 70% at 10,000 vertices, and by 90% at 100,000 vertices.

The increased efficiency stemming from our proposed approach also results in an effective increase in the rate at which RRT* converges to the globally optimal solution. These implementations. Figure 2 (Left) illustrates the cost of the best path found in the baseline RRT* and the modified RRT*. In general and on average, the modified RRT* finds paths of lower cost significantly faster than the baseline RRT*.

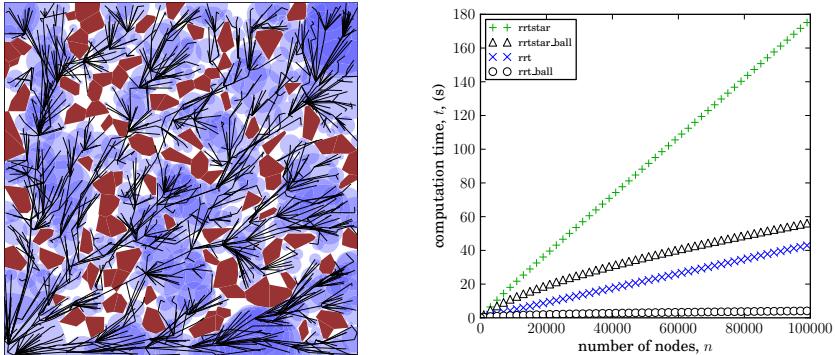


Fig. 1 (Left) Search tree, black, and balls where explicit collision checks are unnecessary, purple, for the modified RRT* algorithm with $n = 2,000$. (Right) Runtimes for the four algorithms tested, averaged over 30 runs of each algorithm, and using graph size buckets of 1000 nodes.

Figure 2 (Right) illustrates the average number of explicit checks required for points which are not in collision vs. different graph sizes. As the number of samples added to the database grows, the probability of performing an explicit check decreases. At a graph size of 100,000 nodes, on average only one out of every 100 samples required an explicit check when the implementations reached that point.

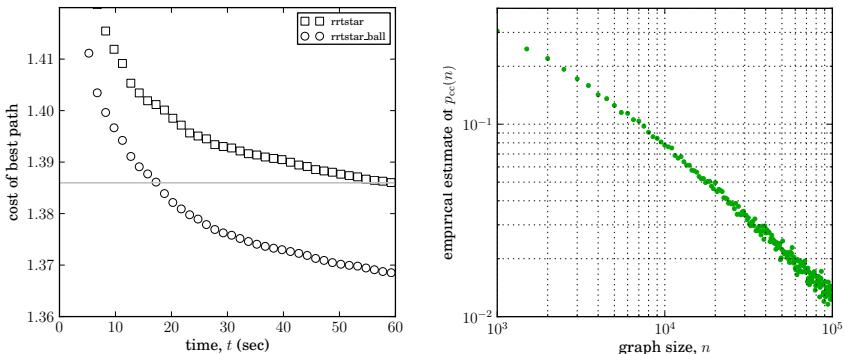


Fig. 2 (Left) RRT* best-path cost vs. time, with and without the proposed method, averaged over 30 runs and in buckets of 100 time samples. The proposed method yields significantly faster convergence. (Right) The experimental probability of performing an explicit collision query vs. graph size. Only 1% of new nodes require an explicit check when graph size is 100,000 nodes.

Figure 3 illustrates the computation of a single iteration of the four algorithms at various different graph sizes and for two different obstacle configurations. Increasing the number of obstacles increases the average iteration time while the graphs remain small; as the graph grows, the iteration times for higher obstacle count case approach those of the lower obstacle count case.

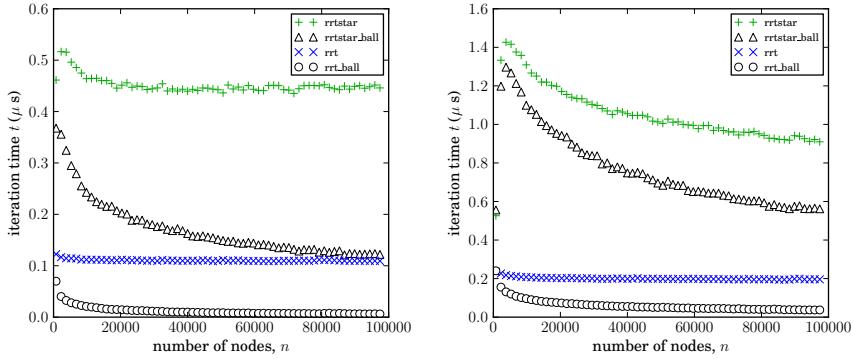


Fig. 3 The average computation time of a single iteration vs. algorithm (plot style), over 30 runs, in a configuration with 500 and 1,000 obstacles (left and right, respectively)

A Voronoi diagram obstacle index makes exact collision distance computation no more expensive than collision checking, however generating a Voronoi diagram in higher dimensions is prohibitively complicated. To address this, we compare the performance of the RRT with and without our proposed modification in a second implementation utilizing a kd-tree for point-proximity searches, and an axis-aligned bounding box tree with Algorithm 3 for set distance queries and collision checking. Obstacles are generated as randomly placed simplices.

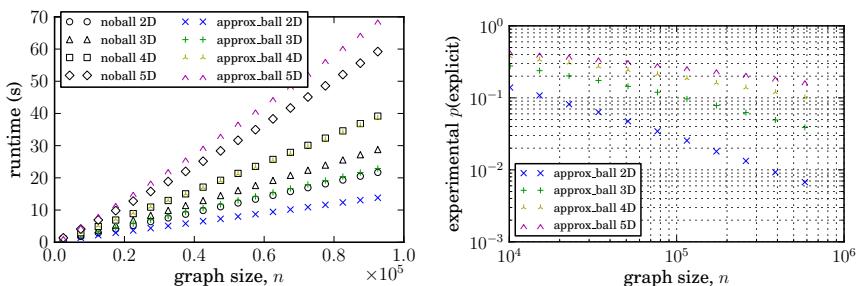


Fig. 4 Mean wall-clock runtime (Left) and experimental p_{explicit} (Right), over 20 runs, for RRT in a unit workspace $X = [0, 1]^d$ for several d

Figure 4 illustrates the measured runtime of this implementation, as well as the observed frequency of performing an explicit collision check for a unit workspace in 2 to 5 dimensions. For this choice of index, the modified algorithm using an approximate collision distance reaches a given graph size faster in 2 and 3 dimensions, at the roughly the same speed in 4 dimensions, and slower in 5 dimensions. The number of explicit collision checks is also shown. There is a significant degradation of the performance in higher dimensions. However, we note that uniform placement of obstacles is a somewhat degenerate scenario in higher dimensions. For many problems of interest, it is likely that obstacles will lie in a lower dimensional subspace, which is a fact that the proposed algorithm is not able to exploit. Extensions to the algorithm that would be able to take advantage of lower dimensionality of the obstacle are left to future work.

6 Conclusions

In this paper, we have introduced a novel approach to collision checking in sampling-based algorithms. This approach allows us to demonstrate, both theoretically and experimentally, that collision-checking is not necessarily a computational bottleneck for sampling-based motion planning algorithms. Rather, the complexity is driven by nearest-neighbor searches within the graph constructed by the algorithms. Experiments indicate significant runtime improvement in practical implementations.

The proposed approach is very general but there are some important implementation details to be aware of when using it. First, we indicate that points which are sampled in collision are kept in order to characterize the obstacle set (i.e., in addition to those that are not in collision). While this allows that the expected number of explicit collision checks goes to zero, the number of points required to truly characterize the obstacle set can be quite large, especially in high dimension. In practice, strategies for biasing samples away from the obstacle set are likely to be more effective than keeping points which are sampled in collision. If in-collision samples are not kept and no biasing is used, the expected number of explicit checks will approach the proportion of the workspace volume which is in collision. However, even in such a case, the strategy described in this paper is effective at marginalizing the extra collision checks in the asymptotically optimal variants of sampling based motion planning algorithms. As an example, the expected runtime ratio between RRT* and RRT will be a constant which does not depend on the obstacle configuration, even if no in-collision samples are kept.

In addition, we show that calculating a sufficient approximation of the collision distance of a particular point and obstacle often does not require more computation than the worst case of performing a collision query. While this is true for a single obstacle, it is important to note that collision checking is often done using a spatial index and the choice of index may affect how the efficiency of a collision distance query compares to a simple collision query.

Also, In practice our method may benefit multi-query algorithms (e.g. PRM) and asymptotically optimal algorithms (e.g. PRM*, RRT*) more than single-query

feasible planning algorithms (e.g. RRT). The latter return after finding a single path, and thus experience less ball coverage of the free space and proportionately more explicit checks, in a sparse environment. The multi-query and asymptotically optimal algorithms require more collision checks per new node, and so offer more opportunities for savings.

Lastly, some of the steps leverage the fact that in path planning problems, the straight line connecting two points is a feasible trajectory for the system. This is in general not the case for robotic systems subject to, e.g., differential constraints. Extensions to such systems is a topic of current investigation.

Acknowledgments. This work was partially supported by the Office of Naval Research, MURI grant #N00014-09-1-1051, the Army Research Office, MURI grant #W911NF-11-1-0046, and the Air Force Office of Scientific Research, grant #FA-8650-07-2-3744.

References

1. Amato, N., Song, G.: Using Motion Planning to Study Protein Folding Pathways. *Journal of Computational Biology* 9(2), 149–168 (2004)
2. Aurenhammer, F.: Voronoi Diagrams — A Survey of a Fundamental Data Structure. *ACM Computing Surveys* 23(3), 345–405 (1991)
3. Bentley, J.L.: Multidimensional binary search trees used for associative searching. *Commun. ACM* 18, 509–517 (1975)
4. Bohlin, R., Kavraki, L.E.: Path Planning using Lazy PRM. In: International Conference on Robotics and Automation (2000)
5. Boor, V., Overmars, M.H., van der Stappen, A.F.: The Gaussian sampling strategy for probabilistic roadmap planners. In: IEEE Int. Conf. on Robotics and Automation, pp. 1018–1023 (1999)
6. Cortes, J., Simeon, T., Ruiz de Angulo, V., Guieysse, D., Remaud-Simeon, M., Tran, V.: A path planning approach for computing large-amplitude motions of flexible molecules. *Bioinformatics* 21(1), 116–125 (2005)
7. Hsu, D., Kavraki, L.E., Latombe, J.C., Motwani, R., Sorkin, S.: On finding narrow passages with probabilistic roadmap planners. In: Agarwal, P., Kavraki, L.E., Mason, M.T. (eds.) *Robotics: The Algorithmic Perspective* (WAFR 1998), pp. 141–154. A.K. Peters/CRC Press, Wellesley (1998)
8. Karaman, S., Frazzoli, E.: Sampling-based algorithms for optimal motion planning. *Int. Journal of Robotics Research* 30(7), 846–894 (2011)
9. Kavraki, L., Svestka, P., Latombe, J., Overmars, M.: Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation* 12(4), 566–580 (1996)
10. Latombe, J.: *Motion Planning: A Journey of Molecules, Digital Actors, and Other Artifacts*. *International Journal of Robotics Research* 18(11), 1119–1128 (2007)
11. LaValle, S.: *Planning Algorithms*. Cambridge University Press (2006)
12. LaValle, S., Kuffner, J.J.: Randomized kinodynamic planning. *International Journal of Robotics Research* 20(5), 378–400 (2001)
13. Samet, H.: *Foundations of multidimensional and metric data structures*. Morgan Kaufmann (2006)

Faster Sample-Based Motion Planning Using Instance-Based Learning^{*}

Jia Pan, Sachin Chitta, and Dinesh Manocha

Abstract. We present a novel approach to improve the performance of sample-based motion planners by learning from prior instances. Our formulation stores the results of prior collision and local planning queries. This information is used to accelerate the performance of planners based on probabilistic collision checking, select new local paths in free space, and compute an efficient order to perform queries along a search path in a graph. We present fast and novel algorithms to perform k -NN (k -nearest neighbor) queries in high dimensional configuration spaces based on locality-sensitive hashing and derive tight bounds on their accuracy. The k -NN queries are used to perform instance-based learning and have a sub-linear time complexity. Our approach is general, makes no assumption about the sampling scheme, and can be used with various sample-based motion planners, including PRM, Lazy-PRM, RRT and RRT*, by making small changes to these planners. We observe up to 100% improvement in the performance of various planners on rigid and articulated robots.

1 Introduction

Motion planning is an important problem in robotics, virtual prototyping and related areas. Most of the practical methods for motion planning of high-DOF (degrees-of-freedom) robots are based on random sampling in configuration spaces, including

Jia Pan · Dinesh Manocha
Department of Computer Science,
University of North Carolina, Chapel Hill
e-mail: {panj, dm}@cs.unc.edu

Sachin Chitta
Willow Garage Inc., Menlo Park, CA 94025, USA
e-mail: sachinc@willowgarage.com

* This research is partially supported by NSF grants 1000579 and 1117127, and willow garage.

PRM [13] and RRT [14]. The resulting algorithms avoid explicit computation of obstacle boundaries in the configuration space (\mathcal{C} -space) and use sampling techniques to compute paths in the free space ($\mathcal{C}_{\text{free}}$). The main computations include probing the configuration space for collision-free samples, joining nearby collision-free samples by local paths, and checking whether the local paths lie in the free space. There is extensive work on different sampling strategies, faster collision checking, or biasing the samples based on local information to handle narrow passages.

The collision detection module is used as an oracle to collect information about the free space and approximate its topology. This module is used to classify a given configuration or a local path as either collision-free (i.e. in $\mathcal{C}_{\text{free}}$) or in-collision (i.e. overlaps with \mathcal{C}_{obs}). Most motion planning algorithms tend to store only the collision-free samples and local paths, and use them to compute a global path from the initial configuration to the goal configuration. However, the in-collision configurations or local paths are typically discarded.

One of our goals is to exploit all prior or historical information related to collision queries and improve the performance of the sample-based planner. Some planners tend to utilize the in-collision configurations or the samples near the boundary of the configuration obstacles (\mathcal{C}_{obs}) to bias the sample generation or improve the performance of planners in narrow passages [4, 9, 18, 23]. However, it can be expensive to perform geometric reasoning based on the outcome of a large number of collision queries in high-dimensional spaces. As a result, most prior planners only use partial or local information about configuration spaces, and can't provide any guarantees in terms of improving the overall performance.

Main Results: We present a novel approach to improve the performance of sample-based planners by learning from prior instances of collision checking, including all in-collision samples. Our formulation uses the historical information generated using collision queries to compute an approximate representation of \mathcal{C} -space as a hash table. Given a new probe or collision query in \mathcal{C} -space, we first perform instance-based learning on the approximate \mathcal{C} -space and compute a collision probability. This probability is used as a similarity result or a prediction of the exact collision query and can improve the efficiency of a planner in the following ways:

- The probability is used to design a collision filter for a local planning query in high-dimensional configuration spaces.
- Explore the \mathcal{C} -space around a given configuration and select a new sample (e.g. for RRT planners [14]).
- Compute an efficient order to perform local planning queries along a path in the graph (e.g. for lazyPRM planners [13]).

The underlying instance-based learning is performed on approximate \mathcal{C} -space using k -NN (k -nearest neighbor) queries. All the prior configurations used by the planning algorithm and their collision outcomes are stored incrementally in a hash table. Given a new configuration or a local path, our algorithm computes the nearest neighbors in the hash table. We use locality-sensitive hashing (LSH) algorithms to

perform approximate k -NN computations in high-dimensional configuration spaces. Specifically, we present a line-point k -NN algorithm that can compute the nearest neighbors of a line. We derive bounds on the accuracy and time complexity of k -NN algorithms and show that the collision probability computed using LSH-based k -NN algorithm converges to exact collision detection as the size of dataset increases.

We present improved versions of PRM, lazyPRM, RRT planning algorithms based on instance-based learning. Our approach is general and can be combined with any sampling scheme. Furthermore, it is quite efficient for high-dimensional configuration spaces. We have applied these planners to rigid and articulated robots, and observe up to 100% speedups based on instance-based learning. The additional overheads are in terms of storing the prior instances in a hash table and performing k -NN queries, which take a small fraction of the overall planning time.

The rest of the paper is organized as follows. We survey related work in Section 2. Section 3 gives an overview of sample-based planners, instance-based learning and our approach. We present the learning algorithms and analyze their accuracy and complexity in Section 4. We show the integration of instance-based learning with different motion planning algorithms in Section 5 and highlight the performance of modified planners on various benchmarks in Section 6.

2 Related Work

In this section, we give a brief overview of prior work on the use of machine learning techniques in motion planning and performing efficient collision checking to accelerate sample-based motion planning.

2.1 Machine Learning in Motion Planning

Many techniques have been proposed to improve the performance of sample-based motion planning based on machine learning. Marco et al. [16] combine a set of basic PRM motion planners into a powerful ‘super’ planner by assigning different basic planners to different regions in \mathcal{C} -space, based on offline supervised learning. Some unsupervised learning approaches construct an explicit or implicit representation of $\mathcal{C}_{\text{free}}$ and perform adaptive sampling based on this model. Burns and Brock [5] use entropy to measure each sample’s utility to improve the coverage of PRM roadmap. Hsu et al. [11] adaptively combine multiple sampling strategies to improve the roadmap’s connectivity. Some variants of RRT, which use workspace or taskspace bias (e.g., [10]), can be extended by changing the bias parameters adaptively. Scholz and Stilman [21] combine RRT with reinforcement learning. Given a sufficient number of collision-free samples in narrow passage, learning techniques have been used to estimate a zero-measure subspace to bias the sampling in narrow passages [7]. Our approach is complimentary to all these techniques.

2.2 Collision Detection Oracle in Motion Planning

Some of the previous approaches tend to exploit the knowledge about \mathcal{C} -space gathered using collision checking. Boor et al. [4] use pairs of collision-free and in-collision samples to collect information about \mathcal{C}_{obs} and perform dense sampling near \mathcal{C}_{obs} . The same idea is used in many variants of RRT, such as retraction-based planners [18]. Sun et al. [23] bias sampling near narrow passages by using two in-collision samples and one collision-free sample to identify narrow passages in \mathcal{C} -space. Kavraki et al. [13] use in-collision samples to estimate the visibility of each sample and perform heavier sampling in regions with small visibility. Denny and Amato [9] present a variation of PRM that memorizes in-collision samples and constructs roadmaps in both $\mathcal{C}_{\text{free}}$ and \mathcal{C}_{obs} , in order to generate more samples in narrow passages. All of these methods utilize in-collision samples to provide better sampling strategies for the planners, in the form of different heuristics. Our approach neither makes assumptions about the underlying sampling scheme nor biases the samples. As a result, our algorithm can be combined with these methods.

2.3 k -Nearest Neighbor (k -NN) Search

The problem of finding the k -nearest neighbor within a database of high-dimensional points is well-studied in various areas, including databases, computer vision, and machine learning. Samet's book [20] provides a good survey of various techniques used to perform k -NN search. In order to handle large and high-dimensional spaces, most practical algorithms are based on approximate k -NN queries [6]. In these formulations, the algorithm is allowed to return a point whose distance from the query point is at most $1 + \varepsilon$ times the distance from the query to its k -nearest points; $\varepsilon > 1$ is called the *approximation factor*.

3 Overview

In this section, we give an overview of the sample-based planner and provide a brief background on instance-based learning.

3.1 Notations and Symbols

We denote the configuration space as \mathcal{C} -space, and each point within the space as a configuration \mathbf{x} . \mathcal{C} -space is composed of two parts: the collision-free points ($\mathcal{C}_{\text{free}}$) and the in-collision points (\mathcal{C}_{obs}). \mathcal{C} -space may be non-Euclidean, but it is possible to approximately embed a non-Euclidean space into a higher-dimensional Euclidean space (e.g., using the Linial-London-Rabinovich embed [15]) to perform k -nearest neighbor queries. A *local path* in \mathcal{C} -space is a continuous curve that connects two configurations. It is difficult to compute \mathcal{C}_{obs} or $\mathcal{C}_{\text{free}}$ explicitly, therefore

sample-based planners use collision checking between the robot and obstacles to probe the \mathcal{C} -space implicitly. These planners perform two kinds of queries: the *point query* and the *local path query*. We use the symbol Q to denote either of these queries.

3.2 Enhance Motion Planner with Instance-Based Learning

The goal of a motion planner is to compute a collision-free continuous path between the initial and goal configurations in \mathcal{C} -space. The resulting path should completely lie in $\mathcal{C}_{\text{free}}$ and should not intersect with \mathcal{C}_{obs} . As shown in Figure 1(a), sample-based planners learn about the connectivity of \mathcal{C} -space implicitly based on collision queries. The query results can also be used to bias the sampling scheme of the planner via different heuristics (e.g., retraction rules).

Instance-based learning is a well known family of algorithms in machine learning that learn properties of new problem instances by comparing them with the instances observed earlier that have been stored in memory [19]. In our case, we store all the results of prior collision queries, including collision-free as well as in-collision queries. Our goal is to sufficiently exploit such prior information and accelerate the planner computation. The problem instance in our context is the collision query being performed on a given configuration or a local path in \mathcal{C} -space. In particular, performing exact collision queries for local planning can be expensive. Collision checking for a discrete configuration is relatively cheap, but still can be time consuming if the environment or robot's geometric representation has a high complexity. We utilize the earlier instances or the stored information by performing k -NN queries and geometric reasoning on query results.

Our new approach to exploit prior information for motion planning is shown in Figure 1(b). When the collision checking routine finishes probing the \mathcal{C} -space for a given query, it stores all the obtained information in a dataset corresponding to historical collision query results. If the query is a point within \mathcal{C} -space, the stored information is its binary collision status. If the query is a local path, the stored information includes the collision status of different configuration points along the path. The resulting dataset of historical collision results constitutes the complete knowledge we have about \mathcal{C} -space, coming from collision checking routines. Therefore, we use it as an approximate description of the underlying \mathcal{C} -space: the in-collision samples are an approximation of \mathcal{C}_{obs} , while the collision-free samples are used to encode $\mathcal{C}_{\text{free}}$. These samples are used by instance-based learning algorithms to estimate the collision status of new queries.

Given a new query Q , either a point or a local path, we first perform k -NN search on the dataset to find its neighbor set S , which provides information about the local \mathcal{C} -space around the query. If S contains sufficient information to infer the collision status of the query, we compute a collision probability for the new query based on S ; otherwise we perform exact collision checking for this query. The calculated collision probability provides prior information about the collision status of the given query and is useful in many ways. First, it can be used as a culling filter to avoid the

Algorithm 1. learning-based-collision-query(Q)

```

begin
  if  $Q$  is point query then
     $S \leftarrow$  point-point- $k$ -NN( $Q$ )
    if  $S$  provides sufficient information for reasoning then
       $\quad$  approximate-collision-query( $S, Q$ )
    else exact-collision-query( $S, Q$ )
  if  $Q$  is line query then
     $S \leftarrow$  line-point- $k$ -NN( $Q$ )
    if  $S$  provides sufficient information for reasoning then
       $\quad$  approximate-continuous-collision-query( $S, Q$ )
    else exact-continuous-collision-query( $S, Q$ )
end

```

exact (and expensive) collision checking for queries that correspond to the configurations or local paths deep inside $\mathcal{C}_{\text{free}}$ or \mathcal{C}_{obs} . Secondly, it can be used to decide an efficient order to perform exact collision checking for a set of queries. For example, many planners like RRT need to select a local path that can best improve the local exploration in $\mathcal{C}_{\text{free}}$, i.e., a local path with a long length but is also collision-free. The collision probability computation can be used to compute an efficient sorting strategy and thereby reduces the number of exact collision tests.

Whether we have sufficient information about S (in Algorithm 1) is related to how much confidence we have in terms of performing inferencing from S . For example, if there exists an in-collision sample very close to the new query, then there is a high probability that the new query is also an in-collision query. The probability decreases when the distance between the sample and the query increases. A description of our prior instance based collision framework is given in Algorithm 1, which is used as an inexpensive routine to perform probabilistic collision detection. More details about this routine and its applications are given in Section 4.

4 Probabilistic Collision Detection Based on Instance-Based Learning

In this section, we discuss how to avoid the expensive exact collision detection query by estimating the collision probability for a given query. The estimation is implemented by performing efficient k -NN queries on the historical collision query results for the given environment.

4.1 LSH-Based Approximate k -NN Query

A key issue in terms of the instance-based learning framework is its computational efficiency. As we generate hypotheses directly from training instances, the complexity of this computation can grow with the size of historical data. If we use exact

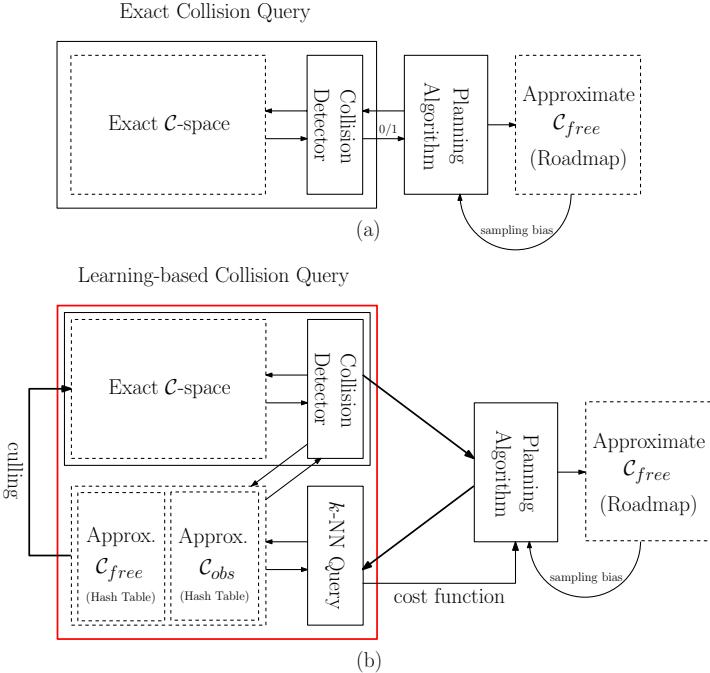


Fig. 1 Comparison between how collision checking is used in prior approaches (a) and our method (b). (a) The collision detection routine is the oracle used by the planner to gather information about $\mathcal{C}_{\text{free}}$ and \mathcal{C}_{obs} . The planner performs binary collision queries, either on point configurations or 1-dimensional local paths, and estimates the connectivity of $\mathcal{C}_{\text{free}}$ (shown as Approximate $\mathcal{C}_{\text{free}}$). Moreover, some planners utilize the in-collision results to bias sample generation by using different heuristics. (b) Our method also performs collision queries. However, we store all in-collision results (as Approximate \mathcal{C}_{obs}) in addition to collision-free results (as Approximate $\mathcal{C}_{\text{free}}$). Before performing an exact collision query, our algorithm performs a k -NN query on the given configuration or local path to compute a collision probability for each query. The collision probability can be used as a cost function to compute an efficient strategy to perform exact collision queries during motion planning. We use novel LSH-based algorithms to perform k -NN queries efficiently and speed up the overall planner.

k -NN computation as the learning method, its complexity can be a linear function of the size of the dataset, especially for high-dimensional spaces. To improve the efficiency of the instance-based learning, we use approximate k -NN algorithms.

Given a dataset $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$ of N points in \mathcal{R}^d , we consider two types of retrieval queries. One is to retrieve points from \mathcal{D} that are closest to a given point query, i.e. the well-known k -NN query, and we call it the *point-point k-NN* query. The second query tries to find the points from \mathcal{D} that are closest to a given line in \mathcal{R}^d , whose direction is \mathbf{v} and passes through a point \mathbf{a} , where $\mathbf{v}, \mathbf{a} \in \mathcal{R}^d$. We call the second query the *line-point k-NN* query. The two types of k -NN queries are illustrated in Figure 2.

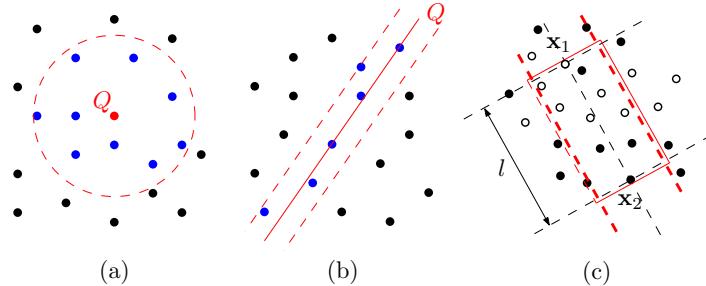


Fig. 2 Two types of k -NN queries used in our method: (a) point-point k -NN; (b) line-point k -NN. Q is the query item and the results of different queries are shown as blue points in each figure. We present novel LSH-based algorithms for fast computation of these queries. (c) Use line-point k -NN query to compute prior instances that can influence the collision status of a local path, which connects x_1 and x_2 in \mathcal{C} -space. The query line is the line segment between x_1 and x_2 . The white points are prior collision-free samples in the dataset and the black points are prior in-collision samples.

In order to develop an efficient instance-based learning framework, we use locality-sensitive hashing (LSH) as an approximate method for k -NN queries, which is mainly designed for point-point queries [1]. However, it can be extended to line queries [2] and hyperplane queries [12]. Basri [3] et al. further extend it to perform point/subspace queries. In this work, we design an efficient LSH-based approximate line-point k -NN query and also derive its error bound similar to the error bound [8] derived for LSH-based point-point k -NN query. The new error bound is important because it enables us to derive the error bound for our approximate collision detection algorithm and eventually prove the completeness of the instance-learning based motion planners. Please refer to [17] for more details of our new line-point k -NN query algorithm, which is omitted in this paper due to limit of space.

4.2 \mathcal{C} -Space Reasoning Based on k -NN Queries

Our approach stores the outcome of prior instances of exact collision queries, including point queries and local path queries, within a database (shown as Approximate $\mathcal{C}_{\text{free}}$ and Approximate \mathcal{C}_{obs} in Figure 1(b)). In this section, we describe our approach to use those stored instances to perform probabilistic collision queries.

The first case is the query point, i.e., the task is to compute the collision status for a sample \mathbf{x} in \mathcal{C} -space. We first perform point-point k -NN query to compute the prior collision instances closest to \mathbf{x} . Next, we use a simple k -NN classifier to estimate \mathbf{x} 's *collision probability* as

$$\mathbb{P}[\mathbf{x} \text{ in collision}] = \frac{\sum_{\mathbf{x}_i \in S \setminus \mathcal{C}_{\text{free}}} w_i}{\sum_{\mathbf{x}_i \in S} w_i}, \quad (1)$$

where $w_i = e^{-\lambda \text{dist}(\mathbf{x}_i, \mathbf{x})}$ is the distance-tuned weight for each k -NN neighbor and S is the neighborhood set computed using point-point k -NN query. In this case, the parameter λ controls the magnitude of the weights. λ corresponds to the *obstacle density* of \mathcal{C}_{obs} , if we model \mathcal{C}_{obs} as a point set generated using a Poisson point process in the \mathcal{C} -space [22].

The second case is the line query, i.e., the goal is to estimate the collision status of a local path in \mathcal{C} -space. We require the local path to lie within the neighborhood of the line segment l connecting its two endpoints, i.e., the local path should not deviate too much from l . The first step is to perform a line-point k -NN query to find the prior collision query results closest to the infinite line that l lies on. Next, we need to filter out the points in S whose projections are outside the truncated segment of l , as shown in Figure 2(c). Finally, we apply our learning method on the filtered results, denoted as S , to estimate the collision probability of the local path.

We compute the line's collision probability via an optimization scheme. The line l is divided into I segments and we assign each segment, say l_i , a label c_i to indicate its collision status: we assign label $c_i = 0$ if l_i is collision-free and assign label $c_i = 1$ if it is in-collision. Given line l 's neighborhood set S computed using the line-point k -NN query, we now compute the label assignments for these segments. First, we compute the conditional collision probability of one point $\mathbf{x}_j \in S$, given the collision status of one line segment l_i :

$$\mathbb{P}[\mathbf{x}_j \text{ in collision} | c_i] = \begin{cases} 1 - e^{-\lambda \text{dist}(l_i, \mathbf{x}_j)}, & c_i = 0, \\ e^{-\lambda \text{dist}(l_i, \mathbf{x}_j)}, & c_i = 1; \end{cases} \quad (2)$$

$$\mathbb{P}[\mathbf{x}_j \text{ collision free} | c_i] = \begin{cases} e^{-\lambda \text{dist}(l_i, \mathbf{x}_j)}, & c_i = 0, \\ 1 - e^{-\lambda \text{dist}(l_i, \mathbf{x}_j)}, & c_i = 1, \end{cases} \quad (3)$$

where $\text{dist}(l_i, \mathbf{x}_j)$ is the distance between \mathbf{x}_j and l_i 's midpoint. Given this formalization, we can compute l_i 's posterior collision probability, given l 's neighborhood set S :

$$\mathbb{P}[c_i | S] \propto \mathbb{P}[S | c_i] \cdot \mathbb{P}[c_i] = \prod_{\mathbf{x}_j \in S} \mathbb{P}[\mathbf{x}_j \text{ collision status} | c_i] \cdot \mathbb{P}[c_i]. \quad (4)$$

The error in this label assignment, i.e., the probability that the computed l_i 's label is not the same as the outcome of the exact collision query, is

$$\mathbb{P}_{\text{error}}[c_i | S] = c_i \cdot \mathbb{P}[c_i = 0 | S] + (1 - c_i) \cdot \mathbb{P}[c_i = 1 | S]. \quad (5)$$

The label assignment algorithm needs to minimize this error probability for each segment l_i . Moreover, the assignment should be coherent, i.e., there is a high probability that adjacent line segments should have the same label. As a result, we can compute a suitable label assignment $\{c_i^*\}_{i=1}^I$ using a binary integer programming:

$$\{c_i^*\} = \underset{\{c_i\} \in \{0,1\}^I}{\text{argmin}} \sum_{i=1}^I \mathbb{P}_{\text{error}}[c_i | S] + \kappa \sum_{i=1}^{I-1} (c_i - c_{i+1})^2, \quad (6)$$

where κ is a weight parameter. The binary integer programming problem can be solved efficiently using dynamic programming. After that, we can estimate the collision probability for the line as

$$\mathbb{P}[l \text{ in collision}] = \max_{i: c_i^* = 1} \mathbb{P}[c_i = 1 | S]. \quad (7)$$

As a byproduct, the approximate first time of contact can be given as $\min_{i: c_i^* = 1} i / I$.

A natural way to use the collision probability formulated as above is to use a specific threshold to justify whether a given query is in-collision or not: if the query's collision probability is larger than the given threshold, we return in-collision; otherwise we return collision-free. We can prove that, for any threshold $0 < t < 1$, the collision status returned by the instance-based learning will converge to the exact collision detection results, when the size of the dataset increases (asymptotically):

Theorem 1. *The collision query performed using LSH-based k-NN will converge to the exact collision detection as the size of the dataset increases, for any threshold between 0 and 1.*

5 Accelerating Sample-Based Planners

In this section, we first discuss techniques to accelerate various sample-based planners based on instance-based learning. Next, we analyze the factors that can influence the performance of the learning-based planners. Finally, we prove the completeness and optimality for the instance-learning enhanced planners.

The basic approach to benefit from the learning framework is highlighted in Algorithm 1, i.e., use the computed collision probability as a filter to reduce the number of exact collision queries. If a given configuration or local path query is close to in-collision instances, then it has a high probability of being in-collision. Similarly, if a query has many collision-free instances around it, it is likely to be collision-free. In our implementation, we only cull away queries with high collision probability. For queries with high collision-free probability, we still perform exact collision tests on them in order to guarantee that the overall collision detection algorithm is conservative. In Figure 3(a), we show how our probabilistic culling strategy can be integrated with PRM algorithm by only performing exact collision checking (`collide`) for queries with collision probability (`icollide`) larger than a given threshold t . Note that the neighborhood search routine (`near`) can use LSH-based point-point k-NN query.

In Figure 3(b), we show how to use the collision probability as a cost function with the lazyPRM algorithm [13]. In the basic version of lazyPRM algorithm, the expensive local path collision checking is delayed till the search phase. The basic idea is that the algorithm repeatedly searches the roadmap to compute the shortest path between the initial and the goal node, performs collision checking along the edges, and removes the in-collision edges from the roadmap. However, the shortest path usually does not correspond to a collision-free path, especially in complex environments. We improve the lazyPRM planning using instance-based learning.

| | |
|---|--|
| <pre> sample($\mathcal{D}^{\text{out}}, n$) $V \leftarrow \mathcal{D} \cap \mathcal{C}_{\text{free}}$, $E \leftarrow \emptyset$ foreach $v \in V$ do $U \leftarrow \text{near}(G^{V,E}, v, \mathcal{D}^{\text{in}})$ foreach $u \in U$ do if $\text{icollide}(v, u, \mathcal{D}^{\text{in}}) < t$ if $\neg \text{collide}(v, u, \mathcal{D}^{\text{out}})$ $E \leftarrow E \cup (v, u)$ near: nearest neighbor search. icollide: probabilistic collision checking based on k-NN. collide: exact local path collision checking. $\mathcal{D}^{\text{in/out}}$: prior instances as input/output. </pre> <p style="text-align: center;">(a) I-PRM</p> | <pre> sample($\mathcal{D}^{\text{out}}, n$) $V \leftarrow \mathcal{D} \cap \mathcal{C}_{\text{free}}$, $E \leftarrow \emptyset$ foreach $v \in V$ do $U \leftarrow \text{near}(G^{V,E}, v, \mathcal{D}^{\text{in}}) \{v\}$ foreach $u \in U$ do $w \leftarrow \text{icollide}(v, u, \mathcal{D}^{\text{in}})$ $l \leftarrow \ (v, u)\$ $E \leftarrow E \cup (v, u)^{w,l}$ do search path p on $G(V, E)$ which minimizes $\sum_e l(e) + \lambda \min_e w(e)$. foreach $e \in p$, $\text{collide}(e, \mathcal{D}^{\text{out}})$ while p not valid </pre> <p style="text-align: center;">(b) I-lazyPRM</p> |
| <pre> $V, \mathcal{D} \leftarrow x_{\text{init}}$, $E \leftarrow \emptyset$ while x_{goal} not reach $x_{\text{rnd}} \leftarrow \text{sample-free}(\mathcal{D}^{\text{out}}, 1)$ $x_{\text{nst}} \leftarrow \text{inearst}(G^{V,E}, x_{\text{rnd}}, \mathcal{D}^{\text{in}})$ $x_{\text{new}} \leftarrow \text{isteer}(x_{\text{nst}}, x_{\text{rnd}}, \mathcal{D}^{\text{in,out}})$ if $\text{icollide}(x_{\text{nst}}, x_{\text{new}}) < t$ if $\neg \text{collide}(x_{\text{nst}}, x_{\text{new}})$ $V \leftarrow V \cup x_{\text{new}}$, $E \leftarrow E \cup (x_{\text{new}}, x_{\text{nst}})$ inearst: find the nearest tree node that is long and has high collision-free probability. isteer: steer from a tree node to a new node, using icollide for validity checking. rewire: RRT* routine used to update the tree topology for optimality guarantee. </pre> <p style="text-align: center;">(c) I-RRT</p> | <pre> $V, \mathcal{D} \leftarrow x_{\text{init}}$, $E \leftarrow \emptyset$ while x_{goal} not reach $x_{\text{rnd}} \leftarrow \text{sample-free}(\mathcal{D}^{\text{out}}, 1)$ $x_{\text{nst}} \leftarrow \text{inearst}(G^{V,E}, x_{\text{rnd}}, \mathcal{D}^{\text{in}})$ $x_{\text{new}} \leftarrow \text{isteer}(x_{\text{nst}}, x_{\text{rnd}}, \mathcal{D}^{\text{in,out}})$ if $\text{icollide}(x_{\text{nst}}, x_{\text{new}}) < t$ if $\neg \text{collide}(x_{\text{nst}}, x_{\text{new}})$ $V \leftarrow V \cup x_{\text{new}}$ $U \leftarrow \text{near}(G^{V,E}, x_{\text{new}})$ foreach $x \in U$, compute weight $c(x) =$ $\lambda \ (x, x_{\text{new}})\ + \text{icollide}(x, x_{\text{new}}, \mathcal{D}^{\text{in}})$ sort U according to weight c. Let x_{min} be the first $x \in U$ with $\neg \text{collide}(x, x_{\text{new}})$ $E \leftarrow E \cup (x_{\text{min}}, x_{\text{new}})$ foreach $x \in U$, rewire(x) </pre> <p style="text-align: center;">(d) I-RRT*</p> |

Fig. 3 Instance-based learning framework can be used to improve different motion planners. Here we present four modified planners.

We compute the collision probability for each roadmap edge during roadmap construction, based on Equation 7. The probability (w) as well as the length of the edge (l) are stored as the costs of the edge. During the search step, we try to compute a shortest path with minimum collision probability, i.e., a path that minimizes the cost $\sum_e l(e) + \lambda \min_e w(e)$, where λ is a parameter that controls the relative weightage of path length and collision probability. As the prior knowledge about the obstacles is considered based on collision probability, the resulting path is more likely to be collision-free.

Finally, the collision probability can be used by motion planner to explore $\mathcal{C}_{\text{free}}$ in an efficient manner. We use RRT to illustrate this benefit (Figure 3(c)). Given a random sample x_{rnd} , RRT computes a node x_{nst} among prior collision-free configurations that are closest to x_{rnd} and expands from x_{nst} towards x_{rnd} . If there is no obstacle in the \mathcal{C} -space, this exploration technique is based on Voronoi heuristic that biases planner in the unexplored regions. However, the existence of obstacles affects its performance: the planner may run into \mathcal{C}_{obs} shortly after expansion and the resulting exploration is limited. Based on instance-based learning, we can first estimate the collision probability for local paths connecting x_{rnd} with each of its neighbors and choose x_{nst} to be the one with a long edge length, but a small collision probability,

i.e., $x_{\text{nst}} = \text{argmax}(l(e) - \lambda \cdot w(e))$, where λ is a parameter used to control the relative weight of these two terms. A similar strategy can also be used for RRT*, as shown in Figure 3(d).

The learning-based planners are faster, mainly because we replace part of the expensive, exact collision queries with relatively cheap k -NN queries. Let the timing cost for a single exact collision query be T_C and for a single k -NN query be T_K , where $T_K < T_C$. Suppose the original planner performs C_1 collision queries and the instance-based learning enhanced planners performs C_2 collision queries and $C_1 - C_2$ k -NN queries, where $C_2 < C_1$. We also assume that the two planners spend the same time A on other computations within a planner, such as sample generation, maintaining the roadmap, etc. Then the speedup ratio obtained by instance-based learning is:

$$R = \frac{T_C \cdot C_1 + A}{T_C \cdot C_2 + T_K \cdot (C_2 - C_1) + A}. \quad (8)$$

Therefore, if $T_C \gg T_K$ and $T_C \cdot C_1 \gg A$, we have $R \approx C_1/C_2$, i.e., if a higher number of exact collision queries are culled, we can obtain a higher speedup. The extreme speedup ratio C_1/C_2 may not be reached, because 1) $T_C \cdot C_1 \gg A$ may not hold, such as when the planner is in narrow passages (A is large) or in open spaces ($T_C \cdot C_1$ is small); 2) $T_C \gg T_K$ may not hold, such as when the environment and robot have low geometric complexity (i.e., T_C is small) or the instance dataset is large and the cost of the resulting k -NN query is high (i.e., T_K is large).

Note that R is only an approximation of the actual acceleration ratio. It may overestimate the speedup because a collision-free local path may have collision probability higher than a given threshold and our method will filter it out. If such a collision-free local path is critical for the connectivity of the roadmap, such false positives due to instance-based learning will cause the resulting planner to perform more exploration and thereby increases the planning time. As a result, we need to choose an appropriate threshold that can provide a balance: we need a large threshold to filter out more collision queries and increase R ; at the same time, we need to use a small threshold to reduce the number of false positives. However, the threshold choice is not important in asymptotic sense, because according to Theorem 1, the false positive error converges to 0 when the dataset size increases.

R may also underestimate the actual speedup. The reason is that the timing cost for different collision queries can be different. For configurations near the boundary of \mathcal{C}_{obs} , the collision queries are more expensive. Therefore, the timing cost of checking the collision status for an in-collision local path is usually larger than that of a collision-free local path, because the former always has one configuration on the boundary of \mathcal{C}_{obs} . As a result, it is possible to obtain a speedup larger than C_1/C_2 .

Finally, as a natural consequence of Theorem 1, we can prove the completeness and optimality of the new planners:

Theorem 2. *I-PRM, I-lazyPRM, I-RRT are probabilistic complete. I-RRT* is probabilistic complete and asymptotic optimal.*

6 Results

In this section, we highlight the performance of our new planners. Figure 4 and Figure 5 show the articulated PR2 and rigid body benchmarks we used to evaluate the performance. We evaluated each planner on different benchmarks, and for each combination of planner and benchmark we ran 50 instances of the planner and computed the average planning time as an estimate of the planner’s performance on this benchmark.

The comparison results are shown in Table 1 and Table 2, corresponding to PR2 benchmarks and rigid body benchmarks, respectively. Based on these benchmarks, we observe:

- The learning-based planners provide more speedup on articulated models. The reason is that exact collision checking on articulated models is more expensive than exact collision checking on rigid models. This makes T_C larger and results in larger speedups.
- The speedup of I-PRM over PRM is relatively large, as exact collision checking takes a significant fraction of overall time within PRM algorithm. I-lazyPRM also provides good speedup as the candidate path nearly collision-free and can greatly reduce the number of exact collision queries in lazy planners. The speedups of I-RRT and I-RRT* are limited or can even be slower than the original planners, especially on simple rigid body benchmarks. The reason is that the original planners are already quite efficient on the simple benchmarks and instance-based learning can result in some overhead.
- On benchmarks with narrow passages, our approach does not increase the probability of finding a solution. However, probabilistic collision checking is useful in culling some of the colliding local paths.

We need to point out that the acceleration results shown in Table 1 and Table 2 show only part of the speedups that can be obtained using learning-based planners. As more collision queries are performed and results stored in the dataset, the resulting planner has more information about \mathcal{C}_{obs} and $\mathcal{C}_{\text{free}}$, and becomes effective

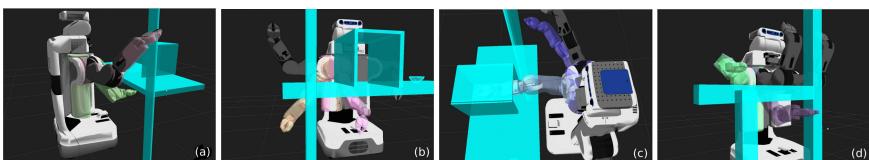


Fig. 4 PR2 planning benchmarks: robot arms with different colors show the initial and goal configurations. The first three benchmarks are of the same environment, but the robot’s arm is in different places: (a) moves arm from under desk to above desk; (b) moves arm from under desk to another position under desk and (c) moves arm from inside the box to outside the box. In the final benchmark, the robot tries to move arm from under a shelf to above it. The difficulty order of the four benchmarks is (c) > (d) > (b) > (a).

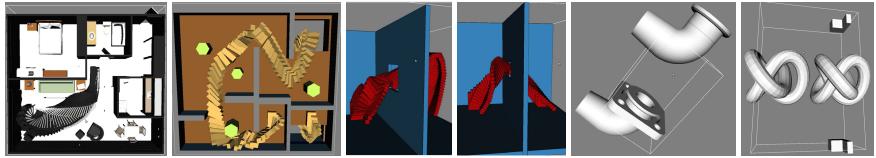


Fig. 5 Rigid body planning benchmarks: from left to right, apartment, cubicles, easy, twistycool, flange and torus. Apartment tries to move the piano to the hallway near the door entrance; cubicles moves the robot in a simple office-like environment and the robot needs to fly through the basement; easy and twistycool are of a similar environment, but twistycool contains a narrow passage. Both flange and torus contain a narrow passage.

Table 1 Performance comparison on different combinations of planners and PR2 benchmarks (in milliseconds). ‘fail’ means all the queries cannot find a collision-free path within 1,000 seconds. The percentage in the brackets shows the speedup obtained using instance-based learning.

| | PRM | I-PRM | lazyPRM | I-lazyPRM | RRT | I-RRT | RRT* | I-RRT* |
|-----|-------|------------|---------|------------|------|------------|------|------------|
| (a) | 12.78 | 9.61 (32%) | 1.2 | 0.87 (37%) | 0.96 | 0.75 (28%) | 1.12 | 1.01 (11%) |
| (b) | 23.7 | 12.1 (96%) | 1.7 | 0.90 (88%) | 1.36 | 0.89 (52%) | 2.08 | 1.55 (34%) |
| (c) | fail | fail | fail | fail | 4.15 | 2.77 (40%) | 3.45 | 2.87 (20%) |
| (d) | 18.5 | 13.6 (36%) | 2.52 | 1.06 (37%) | 7.72 | 5.33 (44%) | 7.39 | 5.42 (36%) |

Table 2 Performance comparison on different combinations of planners and rigid body benchmarks (in milliseconds). The percentage in the brackets shows the speedup based on instance-based learning. For twistycool, which has a narrow passage, we record the number of successful queries among 50 queries (for a 1000 second budget).

| | PRM | I-PRM | lazyPRM | I-lazyPRM | RRT | I-RRT | RRT* | I-RRT* |
|------------|-------|-------------|---------|------------|-------|-------------|-------|------------|
| apartment | 5.25 | 2.54 (106%) | 2.8 | 1.9 (47%) | 0.09 | 0.10 (-10%) | 0.22 | 0.23 (5%) |
| cubicles | 3.92 | 2.44 (60%) | 1.62 | 1.37 (19%) | 0.89 | 0.87(2%) | 1.95 | 1.83 (7%) |
| easy | 7.90 | 5.19 (52%) | 3.03 | 2.01 (50%) | 0.13 | 0.15(-13%) | 0.26 | 0.27 (-4%) |
| flange | fail | fail | fail | fail | 48.47 | 25.6 (88%) | 46.07 | 26.9 (73%) |
| torus | 31.52 | 23.3 (39%) | 4.16 | 2.75 (51%) | 3.95 | 2.7 (46%) | 6.01 | 4.23 (42%) |
| twistycool | 1/50 | 3/50 | 2/50 | 3/50 | 4/50 | 3/50 | 2/50 | 3/50 |

in terms of culling. Ideally, we can filter out all in-collision queries and obtain a high speedup. In practice, we don’t achieve ideal speedups due to two reasons: 1) we only have a limited number of samples in the dataset; 2) the overhead of k -NN query increases as the dataset size increases. As a result, when we perform the global planning query repeatedly, the planning time will decrease to a minimum,

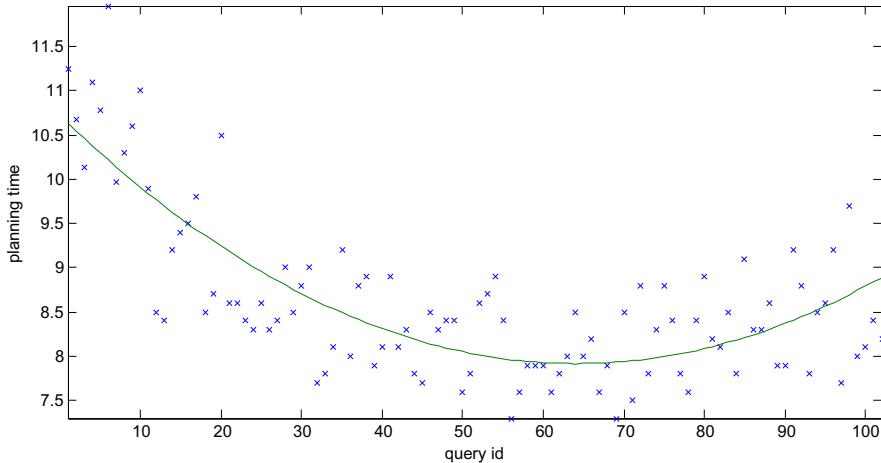


Fig. 6 The time taken by I-PRM when it runs more than 100 times on the benchmark shown in Figure 4 (b). The planning time of a single query first decreases and then increases. The best acceleration acquired is $12.78/7.5 = 70\%$, larger than the 32% in Table 1.

and then increases. This is shown in Figure 6. To further improve the performance, we need to adaptively change the LSH parameters to perform k -NN queries efficiently for datasets of varying sizes.

7 Conclusion and Future Work

In this paper, we use instance-based learning to improve the performance of sample-based motion planners. The basic idea is to store the prior collision results as an approximate representation of \mathcal{C}_{obs} and $\mathcal{C}_{\text{free}}$ and replace the expensive exact collision detection query by a relatively cheap probabilistic collision query. We integrate approximate collision routine with various sample-based motion planners and observe 30–100% speedup on rigid and articulated robots.

References

1. Andoni, A., Indyk, P.: Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Communications of the ACM* 51(1), 117–122 (2008)
2. Andoni, A., Indyk, P., Krauthgamer, R., Nguyen, H.L.: Approximate line nearest neighbor in high dimensions. In: Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 293–301 (2009)
3. Basri, R., Hassner, T., Zelnik-Manor, L.: Approximate nearest subspace search. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33(2), 266–278 (2011)
4. Boor, V., Overmars, M., van der Stappen, A.: The gaussian sampling strategy for probabilistic roadmap planners. In: Proceedings of IEEE International Conference on Robotics and Automation, pp. 1018–1023 (1999)

5. Burns, B., Brock, O.: Toward optimal configuration space sampling. In: Proceedings of Robotics: Science and Systems (2005)
6. Chakrabarti, A., Regev, O.: An optimal randomised cell probe lower bound for approximate nearest neighbour searching. In: Proceedings of IEEE Symposium on Foundations of Computer Science, pp. 473–482 (2004)
7. Dalibard, S., Laumond, J.P.: Linear dimensionality reduction in random motion planning. International Journal of Robotics Research 30(12), 1461–1476 (2011)
8. Datar, M., Immorlica, N., Indyk, P., Mirrokni, V.S.: Locality-sensitive hashing scheme based on p-stable distributions. In: Proceedings of Symposium on Computational Geometry, pp. 253–262 (2004)
9. Denny, J., Amato, N.M.: The toggle local planner for probabilistic motion planning. In: Proceedings of IEEE International Conference on Robotics and Automation (2012)
10. Diankov, R., Ratliff, N., Ferguson, D., Srinivasa, S., Kuffner, J.: Bispace planning: Concurrent multi-space exploration. In: Proceedings of Robotics: Science and Systems (2008)
11. Hsu, D., Sanchez-Ante, G., Sun, Z.: Hybrid PRM sampling with a cost-sensitive adaptive strategy. In: Proceedings of IEEE International Conference on Robotics and Automation, pp. 3874–3880 (2005)
12. Jain, P., Vijayararasimhan, S., Grauman, K.: Hashing hyperplane queries to near points with applications to large-scale active learning. In: Neural Information Processing Systems (2010)
13. 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)
14. Kuffner, J., LaValle, S.: RRT-connect: An efficient approach to single-query path planning. In: Proceedings of IEEE International Conference on Robotics and Automation, pp. 995–1001 (2000)
15. Linial, N., London, E., Rabinovich, Y.: The geometry of graphs and some of its algorithmic applications. Combinatorica 15(2), 215–245 (1995)
16. Morales, M., Tapia, L., Pearce, R., Rodriguez, S., Amato, N.M.: A Machine Learning Approach for Feature-Sensitive Motion Planning. In: Erdmann, M., Hsu, D., Overmars, M., van der Stappen, F. (eds.) Algorithmic Foundation of Robotics VI. STAR, vol. 17, pp. 361–376. Springer, Heidelberg (2004)
17. Pan, J., Chitta, S., Manocha, D.: Faster sample-based motion planning using instance-based learning. Tech. rep., Department of Computer Science, University of North Carolina, Chapel Hill (2012), <http://cs.unc.edu/~panj/techreport.pdf>
18. Rodriguez, S., Tang, X., Lien, J.M., Amato, N.: An obstacle-based rapidly-exploring random tree. In: Proceedings of IEEE International Conference on Robotics and Automation, pp. 895–900 (2006)
19. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach. Prentice Hall (2003)
20. Samet, H.: Foundations of Multidimensional and Metric Data Structures. Morgan Kaufmann Publishers Inc. (2005)
21. Scholz, J., Stilman, M.: Combining motion planning and optimization for flexible robot manipulation. In: Proceedings of IEEE-RAS International Conference on Humanoid Robots, pp. 80–85 (2010)
22. Stoyan, D., Kendall, W.S., Mecke, J.: Stochastic Geometry and Its Applications. John Wiley & Sons (1996)
23. Sun, Z., Hsu, D., Jiang, T., Kurniawati, H., Reif, J.H.: Narrow passage sampling for probabilistic roadmap planners. IEEE Transactions on Robotics 21(6), 1105–1115 (2005)

Scale-Free Coordinates for Multi-robot Systems with Bearing-Only Sensors

Alejandro Cornejo, Andrew J. Lynch, Elizabeth Fudge,
Siegfried Bilstein, Majid Khabbazian, and James McLurkin

Abstract. We propose *scale-free coordinates* as an alternative coordinate system for multi-robot systems with large robot populations. Scale-free coordinates allow each robot to know, up to scaling, the relative position and orientation of other robots in the network. We consider a weak sensing model where each robot is only capable of measuring the angle, relative to its own heading, to each of its neighbors. Our contributions are three-fold. First, we derive a precise mathematical characterization of the computability of scale-free coordinates using only bearing measurements, and we describe an efficient algorithm to obtain them. Second, through simulations we show that even in graphs with low average vertex degree, most robots are able to compute the scale-free coordinates of their neighbors using only 2-hop bearing measurements. Finally, we present an algorithm to compute scale-free coordinates that is tailored to low-cost systems with limited communication bandwidth and sensor resolution. Our algorithm mitigates the impact of sensing errors through a simple yet effective noise sensitivity model. We validate our implementation with real-world robot experiments using static accuracy measurements and a simple scale-free motion controller.

1 Introduction

Large populations of robots can solve many challenging problems such as mapping, exploration, search-and-rescue, and surveillance. All these applications require robots to have at least some information about the *network geometry*: knowledge about other robots positions and orientations relative

Alejandro Cornejo
Massachusetts Institute of Technology, Cambridge, MA
e-mail: acornejo@csail.mit.edu

James McLurkin
Rice University, Houston, TX
e-mail: jmclurkin@rice.edu

to their own [1]. Different approaches to computing network geometry have trade-offs between the amount of information recovered, the complexity of the sensors, the amount of communications required and the cost. A GPS system provides each robot with a global position, which can be used to derive complete network geometry, but GPS is not available in many environments: indoors, underwater, or on other planets. The cost and complexity of most vision- and SLAM-based approaches makes them unsuitable for large populations of simple robots.

This work proposes *scale-free coordinates* as a slightly weaker alternative to the complete network geometry. We argue that scale-free coordinates provide sufficient information to perform many canonical multi-robot applications, while still being implementable using a weak sensing platform. Informally, scale-free coordinates provide the complete network geometry information up to an unknown scaling factor *i.e.* the robots can recover the shape of the network, but not its scale.

Formally, the scale-free coordinates of a set of robots S is described by a set of tuples $\{(x_i, y_i, \theta_i) \mid i \in S\}$. The relative position of robot $i \in S$ is represented by the coordinates (x_i, y_i) which match are correct up to the same (but unknown) multiplicative constant α . The relative orientation of robot $i \in S$ is represented by θ_i . Of particular interest to us are the *local scale-free coordinates* of a robot, which are simply the scale-free coordinates of itself and its neighbors, measured from its reference frame.

We consider a simple sensing model in which each robot can only measure the angle, relative to its own heading, to neighboring robots. These sensors are appropriate for low-cost robots that can be deployed in large populations [2]. Our approach allows each robot to use the bearing measurements available in the network to determine the relative positions and orientations of any subset of robots up to scaling. We remark that in this work we make no assumptions on the relationship between the Euclidean distance between two robots and presence of an edge in the communication graph between them. In particular, *we do not* assume the communication graph is a unit disk graph, or any other type of geometric graph.

With only local bearing measurements, a robot has the capability to execute a large number of algorithms [3, 4, 5], but this information is insufficient to directly compute all the parameters of its network geometry. For instance, consider the canonical problem of controlling a multi-robot system to a centroidal Voronoi configuration [6]. This is straightforward to solve with the complete network geometry, but it is not possible to use only the bearing measurements to your neighbors. Figure 1 shows two configurations with the same bearing measurements that produce very different Voronoi cells (in this diagram we assume robots at the center of adjacent Voronoi cells are neighbors in the communication graph). Local scale-free coordinates are sufficient for each robot to compute the shape of its Voronoi cell. However, since scale-free distances have no units, the robot cannot distinguish between 3 m or 3 cm

distance to the centroid. This presents challenges to algorithms, in particular to motion control, which we consider in our experiments in Section 5.3.

There are three main contributions in this work. Section 3 presents the theoretical foundation for scale-free coordinates, and proves the necessary and sufficient conditions required to compute scale-free coordinates for the entire configuration of robots. We then generalize this approach to compute the scale-free coordinates of any subset of the robots. Section 4 shows through simulations, that in random configurations most robots are able to compute their local scale-free coordinates in only 3 or 4 communication rounds, even in networks with low average degree. Section 5 presents a simplified algorithm, tailored for our low-cost multi-robot platform [7], to compute local scale-free coordinates using information from the 2-hop neighborhood around each robot. The 2-hop algorithm computes scale-free coordinates efficiently with a running time that is linear in the number of angle measurements. Our platform is equipped with sensors that only measure coarse bearing to neighboring robots, we mitigate the effect of this errors through a noise sensitivity model. We show accuracy data from static configurations, and implement a simple controller to demonstrate the feasibility of using the technique for motion control.

1.1 Related Work

Much of the previous work on computation of coordinates for multi-robot systems focuses on computing coordinates for each robot using beacon or anchor robots (or landmarks) with known coordinates [8, 9, 10]. There are also distributed approaches, which do not require globally accessible beacon robots, but instead use multi-hop communication to spread the beacon positions throughout the network [11]. Generally, these approaches do not scale for large swarms of simple mobile robots. Moreover, these approaches are generally based on some form of triangulation. In contrast, the approach proposed in this paper can be used to compute the scale-free coordinates even in graphs where there does not exist a single triangle.

The literature presents multiple approaches to network geometry such as pose in a shared external reference frame [12], pose in a local reference frame [1], distance-only [13, 14, 15] bearing-only [16, 17, 18], sorted order of bearing [19], or combinatorial visibility [20].

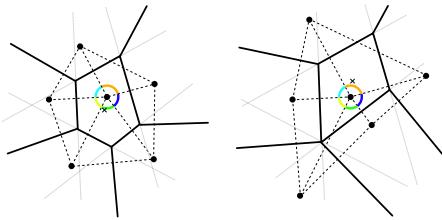


Fig. 1 Two distinct Voronoi cells with the same angle measurements. A robot cannot distinguish these cells using only the angle measurements to its neighbors.

The closest in spirit to our work is the “robust quads” work of Moore et. al. [13]. Using inter-robot distance information, they find *robust quadrilaterals* in the network around each robot and combine them to recover the positions of the robot’s neighbors. Our work is in the same vein, except that we use inter-robot bearing information instead, which allows us to also recover relative orientation. In the error-free case, we present localization success rates which are comparable to the Moore results. However, our approach has less requirements on the graph; scale-free coordinates can be extracted in a graph formed by robust quadrilaterals, but there are graphs without even a single robust quadrilateral where scale-free coordinates are computable.

From the computational geometry literature, the closest work to ours it that of Whiteley [21], who studied directional graph rigidity using the tools of matroid theory. This paper follows a simpler alternative algebraic characterization that allows us to directly compute the scale-free coordinates of any subset of robots. In addition, Bruck [22] addresses the problem of finding a planar spanner of a unit disk graph by only using local angles. The Bruck work is similar to our approach of forming a virtual coordinate system, but their focus delves into routing schemes for sensor networks.

Bearing-only models are more limited than range-bearing models and the type of problems to solve is reduced. In addition, the amount of inter-robot communication often increases greatly. The inter-robot communication requirement is often overlooked in the literature. However, algorithms that require large amounts of information from neighboring robots or many rounds of message passing are impractical on systems with limited bandwidth. This work uses the bearing-only sensor model with scale-free coordinates to balance the trade-off between cost, complexity, communications, and capability.

2 System Model and Definitions

We assume each robot is deployed at an arbitrary position in the Euclidean plane and with an arbitrary orientation unit vector. The communication network is modeled as an undirected graph, $G = (V, E)$, where every vertex in the graph represents a robot, and $N(u) = \{v \mid \{u, v\} \in E\}$ denotes the neighbors of robot u . We consider a synchronous network model, where the execution progresses in synchronous lock-step rounds. During each round every robot can send a message to its neighbors, and receive any messages sent to it by its neighbors. Moreover we assume that when node u receives a message from node v , it also measures the angle $\theta(u, v)$, relative to its own orientation, from u to v . These assumption greatly simplifies the analysis, and can be implemented easily in a physical system via synchronizers [1].

We define the *realization* of graph G as a function $p : V(G) \rightarrow \mathbb{R}^2$ that maps each vertex of G to a point in the Euclidean plane. We use p_0 to denote the ground truth realization of G , specifically $p_0(v)$ is the position of robot v in a fixed global coordinate system. The function $\phi : V(G) \rightarrow [0, 2\pi)$ maps

each robot v to its *orientation* $\phi(v)$, which is defined as the counter-clockwise angle between the \hat{x} -axis of the global coordinate system and v 's orientation unit vector. For neighboring robots $\{u, v\} \in E(G)$ the angle measurement $\theta(u, v)$ from u to v is the counter-clockwise angle between the orientation unit vector of u and the vector from u to v (see Fig 2). We emphasize that at the beginning of the executions each robot knows only its own unique identifier. We do not assume a global coordinate system; the only way for robot u to sense other robots is by measuring the angles $\theta(u, v)$ for each of its neighbors $v \in N(u)$.

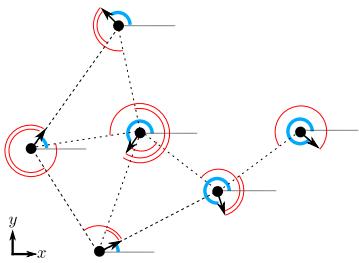


Fig. 2 The position of each robot is depicted by a black disk, and the orientation by a thick (blue) arc. Thin dotted lines connect neighboring robots. Thin (red) arcs represent the angle measurements.

associates to each directed counterpart of every edge $\{u, v\} \in E(G)$ a “length” such that $\ell(u, v) = \ell(v, u)$ (i.e. ℓ is symmetric). Similarly, the function $\omega : \overleftrightarrow{E}(G) \rightarrow [0, 2\pi)$ represents a set of *angle-constraints* on the graph, and associates to each directed counterpart of every edge $\{u, v\} \in E(G)$ an “angle” such that $\omega(u, v) = (\omega(v, u) + \pi) \bmod 2\pi$ (i.e., ω is antisymmetric, and this implies $\psi(\omega(u, v)) = -\psi(\omega(v, u))$). Observe that if all the robots had the same orientation then the set of all angle measurements would describe a set of angle-constraints on G .

We say p is a *satisfying realization* of (G, ℓ) iff every edge $(u, v) \in \overleftrightarrow{E}(G)$ satisfies $\|p(v) - p(u)\| = \ell(u, v)$. Realizations are *length-equivalent* if one can be obtained from the other by a translation, rotation or reflection (distances are invariant to these operations). A length-constrained graph (G, ℓ) has a *unique realization* if all its satisfying realizations are length-equivalent. Similarly, we say p is a *satisfying realization* of (G, ω) iff every edge $(u, v) \in \overleftrightarrow{E}(G)$ satisfies $p(v) - p(u) = \psi(\omega(u, v)) \|p(v) - p(u)\|$. Realizations are *angle-equivalent* if one can be obtained from the other by a translation or uniform-scaling (angles are invariant to these operations). An angle-constrained graph (G, ω) has a *unique realization* if all its satisfying realizations are angle-equivalent.

We define the function $\psi(\theta) := [\cos \theta \sin \theta]^T$ that maps an angle θ to the \hat{x} -axis when anti-clockwise rotated θ radians. Analogously ψ^{-1} receives a unit vector and returns an angle via *atan2* (i.e., $\psi(\psi^{-1}(\hat{n})) = \hat{n}$).

For each undirected edge we consider its two directed counterparts, specifically we use $\overleftrightarrow{E}(G) = \{(u, v), (v, u) \mid \{u, v\} \in E(G)\}$ to denote the directed edges present in G . The function $\ell : \overleftrightarrow{E}(G) \rightarrow \mathbb{R}^+$ represents a set of *length-constraints* on the graph, and as-

3 Theoretical Foundation for Scale-Free Coordinates

This section develops a mathematical framework that characterizes the computability of scale-free coordinates and outlines an efficient procedure to compute them. The full procedure derivation with proofs appears in a tech report [23]. Here we omit some intermediate results and present a self-contained summary. First all the robots in the network agree on a common reference orientation. In a connected graph this is accomplished by having each robot propagating orientation offsets to the entire network with a broadcast tree. As a side-effect of this procedure every robot can compute the relative orientation of every other robot. The details of this distributed algorithm, along with proofs of correctness, appear in [23]. In the rest of the paper we assume that all angle measurements are taken with respect to a global \hat{x} -axis and therefore constitute a valid set of angle-constraints on the graph.

Given an angle-constrained graph, the task of computing scale-free coordinates for every robot is equivalent to finding a unique satisfying realization of the graph. If such a realization does not exist, then either there is no set of scale-free coordinates consistent with the angle-measurements, (perhaps due to measurement errors), or there are multiple distinct sets of scale-free coordinates which produce the same angle measurements, and it is impossible to know which one of them corresponds to the ground truth. We note that every realization of a graph induces a unique set of length- and angle-constraints which are simultaneously satisfied by that realization:

Proposition 1. *A realization p of a graph G induces a unique set of length- and angle-constraints ℓ_p and ω_p which are simultaneously satisfied by p .*

However, the converse does not hold, since there are length- and angle-constraints that do not have a realization which satisfies them simultaneously.

The necessary and sufficient conditions that determine if a set of angle-constraints have a satisfying realization are captured by the cycles of the graph. In particular, given any realization p of G , traversing a directed cycle C of G and returning to the starting vertex there will be no net change in position or orientation. Formally:

$$\sum_{(u,v) \in E(C)} (p(v) - p(u)) = \sum_{(u,v) \in E(C)} \ell_p(u,v) \psi(\omega_p(u,v)) = \mathbf{0}. \quad (1)$$

Since by definition $\ell_p(u,v) = \ell_p(v,u)$ and $\psi(\omega_p(u,v)) = -\psi(\omega_p(v,u))$ we can verify that the direction in which we traverse an undirected cycle is not relevant, since both directions produce the same equation. Since the terms of the equations are two-dimensional vectors, each cycle generates two scalar equations for the x - and y -component. If the realization p of G is unknown, but we know both G and a set of angle-constraints ω of G , then equation (1) represents two linear restrictions on the length of the edges of any realization p which satisfies (G, ω) .

The number of cycles in a graph can be exponential, however we show it suffices to consider only the cycles in a *cycle basis* of G . For a detailed definition of a cycle basis we refer the interested reader to [24]. Briefly, a cycle basis of a graph is a subset of the simple undirected cycles present in a graph, and a connected graph on n vertices and m edges has a cycle basis with exactly $m - n + 1$ cycles. A cycle basis of G can be constructed in $\mathcal{O}(m \cdot n)$ time by first constructing a spanning tree T of G . This leaves $m - n + 1$ non-tree edges, each of which forms a unique simple cycle when added to T . Let $\mathcal{C} = \{C_1, \dots, C_q\}$ be any cycle basis of G .

It will be useful to represent the length of the edges of a realization as a real vector. Let $\mathcal{E} = \{e_1, \dots, e_m\}$ be any ordered set of directed edges that cover all the undirected edges in $E(G)$. Specifically for every undirected edge in $E(G)$ one of its directed counterparts (but not both) is present in $\mathcal{E}(G)$, conversely if a directed edge is present in $\mathcal{E}(G)$ then its undirected version is in $E(G)$. Let \mathbf{x} be an $m \times 1$ column vector whose i^{th} entry represents the length of the directed edge $e_i \in \mathcal{E}$ of *any* satisfying realization of (G, ω) .

Applying equation (1) to a cycle basis of G results in the following:

$$\begin{array}{c} e_1 \ \dots \ e_m \\ \hline C_1 \left[\begin{array}{ccc} a_{11} & \dots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{q1} & \dots & a_{qm} \end{array} \right] \underbrace{\begin{bmatrix} \ell_p(e_1) \\ \vdots \\ \ell_p(e_m) \end{bmatrix}}_{\mathbf{x}} = \mathbf{0}. \\ \vdots \\ C_q \end{array} \quad (2)$$

Here $A_{(G, \omega)}$ is a $2q \times m$ matrix constructed using G , \mathcal{C} and ω . Row i corresponds to a cycle $C_i \in \mathcal{C}$, and column j corresponds to an edge $(u, v) \in \mathcal{E}$. If $(u, v) \in E(C_i)$ then $a_{ij} = \psi(\omega(u, v))$, if $(v, u) \in E(C_i)$ then $a_{ij} = -\psi(\omega(u, v)) = \psi(\omega(v, u))$, otherwise $a_{ij} = 0$. Since these are vector equations, there are two scalar rows in $A_{(G, \omega)}$ for every cycle in \mathcal{C} – one equation for the x -components and one for the y -components of each cycle.

Equation (2) is a homogeneous system, therefore the solution space is precisely the null space of $A_{(G, \omega)}$, denoted by $\text{null}(A_{(G, \omega)})$. Our main result relates the null space of $A_{(G, \omega)}$ to the space of satisfying realizations of (G, ω) .

Let $P_{(G, \omega)}$ be a set of realizations that satisfy (G, ω) , where all equivalent realizations are mapped to a single realization that “represents” its equivalence class. We define the function $f_\omega : P_{(G, \omega)} \rightarrow \mathbb{R}^m$ that maps a realization in $P_{(G, \omega)}$ to a (positive) m -dimensional real vector which contains in its i^{th} entry the length of the directed edge $e_i \in \mathcal{E}$. Therefore $f_\omega(p)$ is simply a vector representation of the set of length-constraints ℓ_p satisfied by p . Observe that proposition 1 implies that when the domain of f_ω is restricted to $P_{(G, \omega)}$ then f_ω has an inverse f_ω^{-1} . We now state the main theorem of this section:

Theorem 2. $p \in P_{(G, \omega)}$ if and only if $f_\omega(p) \in \text{null}(A_{(G, \omega)})$.

This theorem implies each column in the null space basis of $A_{(G, \omega)}$ corresponds to a distinct satisfying realization of (G, ω) , and therefore a distinct

set of scale-free coordinates. If the nullity of $A_{(G,\omega)}$, the number of columns of its null space basis, is zero no set of scale-free coordinates is consistent with the angle-measurements. If the nullity is one, then there is a single set of scale-free coordinates which are consistent with the angle-measurements. If on the other hand the nullity of $A_{(G,\omega)}$ is greater than one, then there are multiple distinct sets of scale-free coordinates consistent with the angle measurements and it is impossible to know which one corresponds to the ground truth. We summarize this in the following corollary.

Corollary 1. *(G, ω) has a unique satisfying realization \iff the nullity of $A_{(G,\omega)}$ is one \iff the scale-free coordinates of every robot in G are computable.*

3.1 Local Scale-Free Coordinates

This subsection describes a procedure that uses the null space basis of $A_{(G,\omega)}$ to compute the scale-free coordinates of any subset of robots. Of particular interest to us is computing the scale-free coordinate of a specific robot and its neighbors (i.e., its local scale-free coordinates). From corollary 1 it follows that if the null space basis of $A_{(G,\omega)}$ has a single column, then we can compute the scale-free coordinates of any subset of robots, since we can compute scale-free coordinates for all robots simultaneously. However, it might be the case that the null space basis of $A_{(G,\omega)}$ has more than one column, but it is still possible to compute the scale-free coordinates of some subset of the robots.

For a set $S \subseteq V(G)$ of vertices, let $G[S]$ be the subgraph of G induced by S , and let $\ell[S]$ and $\omega[S]$ be the length- and angle-constraints that correspond to the edges in $G[S]$. We say an angle-constrained (G, ω) has a unique S -subset realization iff when restricted to the vertices of S all realizations of (G, ω) projected to the vertices in S are equivalent. From this definition we can see that the scale-free coordinates of the subset of robots in S are computable iff (G, ω) has a unique S -subset realization. Using these definitions we can prove the following.

Lemma 3. *The angle-constrained graph (G, ω) has a unique S -subset realization iff there is a superset $S' \supseteq S$ such that $(G[S'], \omega[S'])$ has a unique realization.*

The FIXEDTREE algorithm leverages this lemma to compute scale-free coordinates for any subset of robots. The FIXEDTREE algorithm receives as input a graph G , a subset of vertices $S \subseteq V(G)$, and a null space basis \mathcal{N} of $A_{(G,\omega)}$. If there exists a superset $S' \supseteq S$ such that $(G[S'], \omega[S'])$ has a unique realization it will return this set. From corollary 1 it follows that we can use this set S' and the null space basis \mathcal{N} to compute the unique satisfying realization, and therefore the scale-free coordinates, of $S' \supseteq S$.

Recall that each row in the null space basis \mathcal{N} corresponds to an edge of G , we define a labeling of the edges in G using \mathcal{N} . Fix an edge $e \in G$ and

Algorithm 1. FIXEDTREE(G, S, \mathcal{N})

```

Pick  $w \in S$  arbitrarily.
for each  $\{w, v\} \in E(G)$  where  $\{w, v\}$  is not degenerate in  $\mathcal{N}$  do
     $\mathcal{N}' \leftarrow \text{FIX edge } \{w, v\} \text{ in } \mathcal{N}$ 
     $T \leftarrow \text{BFS traversal of } G \text{ rooted at } w \text{ using only edges fixed in } \mathcal{N}'$ .
    if  $T$  spans all vertices in  $S$  then
        return  $(\mathcal{N}, T)$ 
    end for
return NoSOLUTION

```

let j be the row in \mathcal{N} that corresponds to e , (1) if there are both zero and non-zero entries in row j then e is *degenerate*, (2) if all entries in row j are the same then e is *fixed*, (3) otherwise e is *flexible*.

The FIX transformation –which relies on elementary column operations– receives an edge e and a null space basis \mathcal{N} where e is labeled as flexible, and returns a null space basis \mathcal{N}' where edge e is labeled as fixed. Specifically to FIX an edge e , which corresponds to a row j in a null space basis \mathcal{N} , it suffices to multiply each column i of \mathcal{N} by the reciprocal of element n_{ij} in that column.

The algorithm uses the FIX transformation to finds a tree in G (if it exists) that spans the vertices in S and whose edges can be simultaneously fixed in the null space basis \mathcal{N} . In other words, the FIX algorithm finds a projection of the null space basis \mathcal{N} which is of rank 1 and spans all the vertices in S . The proof of correctness algorithm follows from lemma 3 and theorem 2.

4 Simulation

Here we show that in the robots are deployed in random positions, it is feasible for each robot to compute the local scale-free coordinates of its neighbors using only the angle-measurements taken by other near-by robots. The simulation uses the FIXEDTREE algorithm presented in the previous section. We use G_u^k to denote the k -neighborhood of robot u , which is the set of nodes at k or less hops away from u and the edges between these nodes. In practice to obtain its k -neighborhood G_u^k and the corresponding angle measurements, robot u will need $k + 1$ communication rounds, using messages of size at most $O(\Delta^k)$ where Δ is the maximum degree of the graph. To compute the local scale-free coordinates for robot u using only its k -neighborhood we let $G = G_u^k$, $S = \{u\} \cup N(u)$ and \mathcal{N} be the null space basis of $A_{(G_u^k, \omega)}$. In other words, we use only the null space of the matrix associated with the k -neighborhood of each node and not the entire graph. The computational complexity of the whole procedure is dominated by computing the null space basis. This was implemented using singular value decomposition requiring $O(m^3)$ time where m is the number of edges in G_u^k .

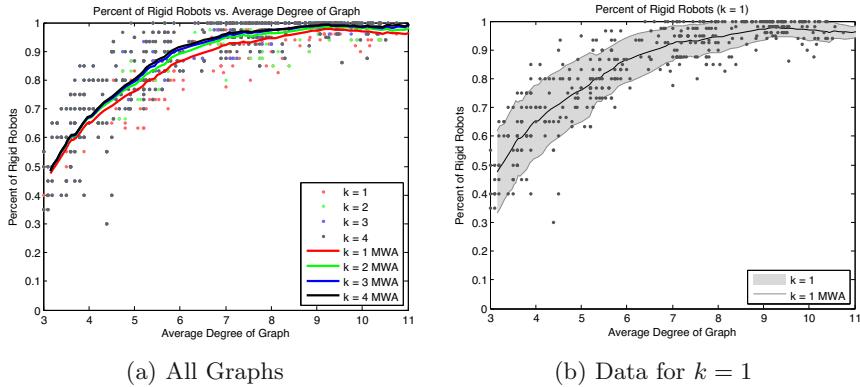


Fig. 3 Simulation data from 500 random graphs. **a:** Percentage of robots with scale-free coordinates vs. average degree of graph. A moving window average for each communication depth is overlaid on the plot. **b:** A closer look at $k=1$ data in all generated graphs. The shaded area represents one standard deviation away from the moving window average.

We ran simulations to determine how useful the algorithm would be in random graphs of various average degrees. Each robot is modeled as a disk with a 10 cm diameter and a communication range of 1 m. For each trial, we consider a circular environment with a 4 m diameter. We assume lossless bidirectional communication and noiseless bearing-only sensors. To be consistent with our hardware platform, we used the same sensing range as the communication range. We considered stationary configurations, but the results presented are applicable while the system is in motion as long as the physical speed of the robots is negligible compared to the speed of communication and computation [1]. Random connected geometric graphs were generated by placing robots uniformly at random in the environment and discarding disconnected graphs. The parameters in the experiments are the population size and the communication depth k (or hop count). The population size controls the density of the graph and the communication depth controls the amount of information available to each robot. In real systems the communication depth will be limited by bandwidth constraints.

We carried out simulations using populations of 20, 30, 40, and 50 robots, running 100 experiments for each population size. Our results show that in 43% of these graphs all robots can successfully compute scale-free coordinates for every other robot, if they use a communication depth $k = \text{diam}(G)$, i.e. $G_u^k = G$. Since in practice bandwidth will be limited we are more interested in the percentage of robots which were able to compute local scale-free coordinates using small constant communication depths.

Our results shown in Figure 3a are encouraging; even for graphs with an average degree as low as 4, we can expect at least half of robots to successfully

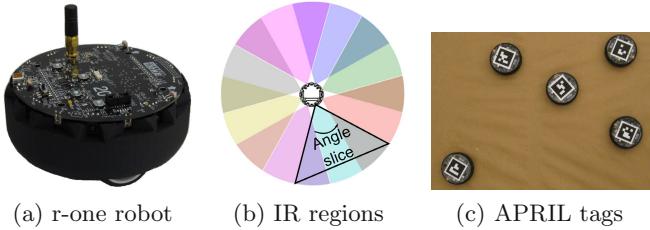


Fig. 4 a: The r-one robot for multi-robot research was designed by MRS group at Rice University. **b:** Top view of the r-one's IR receiver detection regions. Each receiver detects an overlapping 68° , allowing a robot to determine the neighbor angle within 22.5° . **c:** An image from the overhead camera from our data collection system. The robots are outfitted with APRIL tags for detection of ground-truth 2D position and orientation.

compute their local scale-free coordinates. For more typical graphs with an average degree of 6-7, on average 90% of robots can compute their local scale-free coordinates with a communication depth of only $k = 1$. The $k = 1$ depth is the most practical in our experimental platform, because only two communication rounds are required with messages of size $O(\Delta)$. Figure 3b shows a closer look at the data of Figure 3a for $k = 1$. For graphs of degree 6, approximately 80% of the robots can compute their local scale-free coordinates using $k = 1$, indicating that this is a feasible technique for bandwidth-limited platforms.

Figure 3b also allows for a direct comparison to the robust quad results of Moore [13]. Our localization success rates are somewhat better than robust quad results, with the same communication depth $k = 1$, around 10% more nodes with low degree can localize using our algorithm with bearing-only measurements than the robust-quads algorithm with distance-only measurements. In addition, increasing communication depth from $k = 1$ to $k = 2$ increases likelihood of a given robot being able to compute its local scale-free coordinates. Subsequent increases in k have diminishing returns.

5 Hardware Experiments

For our experiments, we use the r-one robot shown in Figure 4a [2]. It is a 11 cm robot with a 32-bit ARM-based microcontroller running at 50 mhz with no floating point unit. The local IR communication system is used for inter-robot communication and localization. Each robot has eight IR transmitters and eight receivers. The transmitters broadcast in unison and emit a radially uniform energy pattern. The robot's eight IR receivers are radially spaced to produce 16 distinct detection regions (shown in Figure 4b). By monitoring the overlapping regions, the bearing of neighbors can be estimated to within $\approx \pi/8$. The IR receivers have a maximum bit rate of 1250bps. Each robot transmits $(\Delta+1)$ 4-byte messages during each round, one a system announce message, and the others contain the bearing measurements to that robot's

neighbors. The system supports a maximum $\Delta = 8$, and we used a $\Delta = 4$ for the motion experiments. A round period of 2500 ms was used to minimize the number of message collisions.

The APRIL tags software system [25] (shown in Figure 4c) is used to provide ground-truth pose information. The system provides a mean position error of 6.56mm and mean angular error of 9.6mrad, which we accept as ground truth.

5.1 TwoHop Scale-Free Algorithm

Given the computational and bandwidth constraints of our platform, it is unfeasible to compute in real-time the null space of the system of cycle equations described in Section 4. However, our simulation results show that a system that uses only 2-hop angle measurements should work reasonably well in practice. In this section we describe a simple distributed algorithm that computes local scale-free coordinates using only 2-hop angle measurements, and which can be implemented easily and efficiently in hardware without a floating point unit (this corresponds to $k = 1$ in our simulation experiments, but as we mentioned earlier, this requires two communication rounds and angle measurements from 2-hops, hence the name). Later we describe how to modify the algorithm to deal with sensing errors.

The main insight behind our algorithm is that instead of considering an arbitrary cycle-basis, when restricted to a 2-hop neighborhood of u we can always restrict ourselves to a cycle-basis composed solely of triangles of which node u is a part of. This is a consequence of the following lemma.

Theorem 4. *Robot u can compute its local scale-free coordinates using 2-hop angle measurements if and only if the graph induced by the vertices in $N(u)$ is connected.*

The basic idea behind the TwoHop SCALE-FREE algorithm is to traverse a tree of triangles, computing the lengths of the edges of the triangles using the SINELAW. Specifically, SINELAW receives a triangle (u, z, w) in the 2-hop neighborhood of u . It assumes the length ℓ_z of the edge (u, z) is known (up to scale), and uses the inner angles $\psi_z = \theta(z, u) - \theta(z, w)$ and $\psi_w = \theta(w, z) - \theta(w, u)$ to return the length (up to scale) of edge (u, v) .

$$\text{SINELAW}(u, z, w) = \ell_z \left| \frac{\sin(\theta(z, u) - \theta(z, w))}{\sin(\theta(w, z) - \theta(w, u))} \right|$$

The following algorithm has a running time which is linear in the number of angle measurements in the 2-hop neighborhood of u .

Noise Sensitivity. To deal with coarse sensor measurements while preserving the computational efficiency (and simplicity) of the algorithm we introduce the concept of *noise sensitivity*. Informally, the noise sensitivity of a triangle captures the expected error of the lengths of a triangle when its

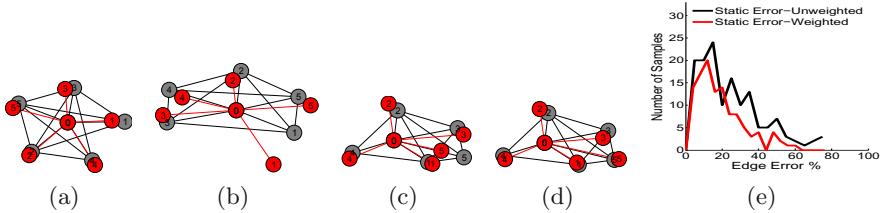


Fig. 5 Scale-free coordinates plot as red nodes and ground-truth data as grey nodes. The IR communication links plot as black edges between grey nodes. The red lines depict the measured bearing between each robot, the block lines are the edges from the ground-truth positions. Four cases are presented: **a:** Accurate scale-free coordinates. **b:** Configuration with a bearing error to robot 1. **c:** Scale-free edge length error to robot 5. **d:** Scale-free edge length corrected with noise sensitivity. **e:** Edge error histograms with and without noise sensitivity for 28 robot configurations and 140 edges.

angles are subject to small changes. For example, observe that given a triangle (u, v, z) , as ψ_z gets closer to zero, the output of the SINELAW becomes more “sensitive to noise”, since a small change in the angle measurements used to compute ψ_z translates to a potentially very large change in the computed length. Formally, the noise sensitivity of each triangle can be defined as a function of the magnitude of the vector gradient of SINELAW(u, v, z). This provides us with an approximation of the expected error in the computed length when using a particular triangle.

Hence, to reduce the effect of noisy measurements in the computed scale-free coordinates it suffices to find a spanning tree of triangles that has the smallest total noise sensitivity. This can be achieved by any standard minimum spanning tree algorithm at minimal additional computational cost. Specifically in our setting a minimum spanning tree of triangles can be found in $O(m \log m)$ time, where m is the number of angle measurements in the 2-hop neighborhood of u .

Algorithm 2. TWOHOP SCALE-FREE algorithm running at node u

```

1: Fix  $v \in N(u)$ 
2: mark  $v$  and set  $\ell_v \leftarrow 1$ 
3:  $Q \leftarrow \text{queue}(v)$ 
4: while  $Q \neq \emptyset$  do
5:    $z \leftarrow Q.\text{pop}()$ 
6:   for each unmarked  $w \in N(z) \cap N(u)$  do
7:     mark  $w$  and set  $\ell_w \leftarrow \text{SINELAW}(u, z, w)$ 
8:      $Q.\text{push}(w)$ 
9:   end for
10: end while

```

5.2 Static Evaluation

We generated 32 random configurations of six r-one robots. Four trials failed due to lost messages between robots, we discarded them and analyze the 28 successful trials. The configurations shown in Figure 5 illustrate some typical errors and the overall accuracy of our experiments. Ideally, the red nodes and edges will directly cover the black edges and grey nodes. The low resolution of the r-one localization system is the largest source of error. Lost messages between robots would occasionally remove edges from the local network, resulting in missing triangles.

To analyze each static configuration, we needed a way to compare scale-free edge lengths to ground-truth edge lengths. For each configuration, we computed an α_{opt} scaling factor that minimized the total edge length error when compared to ground truth. An example of a bearing inaccuracy is shown in Figure 5b for robot 1 to robot 0. Despite this error, our algorithm still effectively computes the edge coefficient. Bearing measurement errors cause the most significant problems in our scale-free coordinates. However, the majority of the bearing errors are still within the 22.5°designed tolerance of the robot. Figure 5c illustrates a scale-free edge length error to robot 5. In this case, the error was caused by a poor selection of triangles. We handled this scenario with noise sensitivity to select a better set of triangles. The corrected position of robot 5 is shown in Figure 5d. The summary error statistics are shown in Figure 5e. Running the algorithm without error sensitivity produced a mean error of 23.4%, and with sensitivity produced a mean error of 19.4%. Given our coarse bearing measurements, these results are reasonable, and are adequate for motion control.

5.3 Dynamic Evaluation: Real-Time Centroid Behavior

This experiment measures the ability of the robot to move to a position specified by local scale-free coordinates, in this case, the centroid of a group of robots. Our controller is basic, it computes the centroid, rotates, and moves a fixed distance. This is intentional — the aim of these experiments is to illustrate the performance of scale-free coordinates algorithm, so we use unfiltered data. We also avoided using any odometry information to improve performance. Since our neighbor round is a (very long) 2500 ms, measuring neighbor bearings while moving can introduce errors, therefore robots remain stationary when measuring the neighbor bearings.

For the first experiment, four stationary robots were arranged in an arbitrary polygon and one moving robot is placed at random initial positions outside the polygon. At each iteration of the algorithm, the moving robot moves a distance of (d_{step}) towards the centroid. For this experiment, we used the robot diameter of 11 cm for (d_{step}). The trajectories of the moving

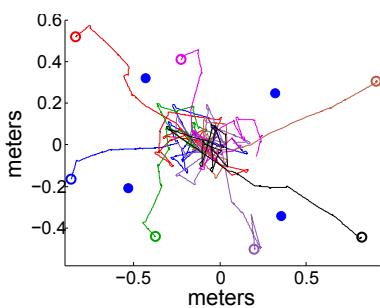


Fig. 6 Four static robots (blue dots) were placed in an arbitrary polygon. A mobile robot was placed in random locations outside the polygon (colored circles). Trajectories of the robot moving toward the centroid are represented by the different colored lines. The robot quickly reaches the centroid, but then oscillates because it does not know how far the goal is from its current position.

robot converging to the centroid are shown in Figure 6. The robot continues to move around the centroid without settling because without knowing the distance to the centroid, the robot cannot know when to stop. We expect the diameter of the convergence region around the centroid to have a mean diameter of approximately $d_{step} = 0.11$ m, which is consistent with our measurement of 0.14 m.

The second experiment looks at the controller's response to a change in the goal position. The stationary robots start in the blue positions, then were moved to the red positions halfway through the experiment. The trajectory shown in Figure 7a show the moving robot successfully converging to the new position, and the size of the convergence region in Figure 7b is again

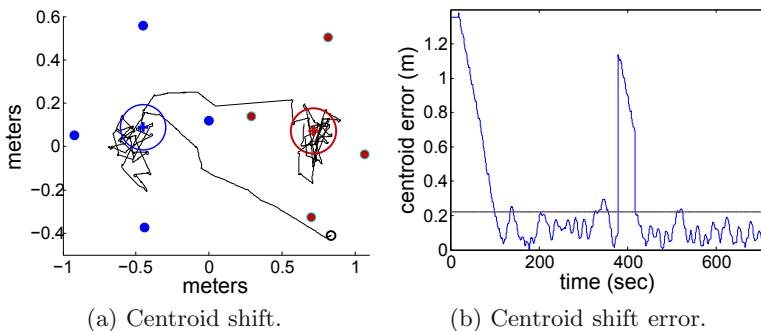


Fig. 7 a: This experiment shifts a group of robots to demonstrate a large shift in the centroid denoted by the blue plus sign. The four blue dots represent the initial polygon of static robots. The red dots represent the shifted group of robots. The black line trajectory shows the trajectory of the motion robot searching for the centroid. **b:** Corresponding error vs. time of trajectory shown in (a) between the motion robot position and the centroid. The robot begins at the black circle with significant error and then oscillates less than $2d_{step}$ around centroid. When the group is shifted the error spikes again and settles to another oscillation around the shifted centroid.

around d_{step} , and mostly bounded by $2d_{step}$, which is shown as the circles in Figure 7a and the horizontal line in Figure 7b.

While the size of the convergence region is set by the step size, the time of convergence is limited by the communications bandwidth — more bandwidth can allow shorter rounds. This blurs the distinction between sensing and communication, but is consistent with the robot speed ratio [26].

6 Conclusion and Future Work

This paper presents local scale-free coordinates as an alternative coordinate system of intermediate power. Our noise sensitivity provided a computationally simple way to deal with sensor errors. However, in future work we will incorporate a full error model to provide superior performance.

In a separate project, we are studying the accuracy of a particle filter to estimate range using odometry and the bearing sensors [27]. This approach uses less communications and provides metrical estimates of range, but requires the robots to be moving, and remain neighbors long enough for the estimate to converge. On the other hand, the approach presented in this paper can be applied even if the robots are static (or to sensor networks). It is unclear which of these two approaches is the most powerful, in the sense proposed by O’Kane [28], which is an interesting question. We believe that for many applications, scale-free coordinates are a viable alternative for relative localization in multi-robot platforms with large populations of simple, low-cost robots.

References

1. McLurkin, J.: Analysis and Implementation of Distributed Algorithms for Multi-Robot Systems. Ph.D. thesis, Massachusetts Institute of Technology (2008)
2. McLurkin, J., Lynch, A.J., Rixner, S., Barr, T.W., Chou, A., Foster, K., Bilstein, S.: A low-cost multi-robot system for research, teaching, and outreach. In: Proc. of the Tenth Int. Symp. on Distributed Autonomous Robotic Systems, DARS 2010 (October 2010)
3. Wei, R., Mahony, R., Austin, D.: A bearing-only control law for stable docking of unicycles. In: Proceedings of the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003), vol. 3, 4, pp. 3793–3798 (2003)
4. Lemaire, T., Lacroix, S., Sola, J.: A practical 3D bearing-only SLAM algorithm. In: 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2005), pp. 2449–2454 (2005)
5. Scheding, S., Dissanayake, G., Nebot, E.M., Durrant-Whyte, H.: An experiment in autonomous navigation of an underground mining vehicle. IEEE Transactions on Robotics and Automation 15(1), 85–95 (1999)
6. Cortes, J., Martinez, S., Karatas, T., Bullo, F.: Coverage control for mobile sensing networks. In: Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 2002, vol. 2, pp. 1327–1332 (2002)

7. McLurkin, J., Lynch, A., Rixner, S., Barr, T., Chou, A., Foster, K., Bilstein, S.: A low-cost multi-robot system for research, teaching, and outreach. In: Proc. of the Tenth Int. Symp. on Distributed Autonomous Robotic Systems, DARS 2010, p. 200 (2010)
8. Hendrickson, B.: The molecule problem: Exploiting structure in global optimization. *SIAM Journal on Optimization* 5(4), 835–857 (1995)
9. Eren, T., Goldenberg, D.K., Whiteley, W., Yang, Y.R., Morse, A.S., Anderson, B.D.O., Belhumeur, P.N.: Rigidity, computation, and randomization in network localization. In: Proc. 23rd IEEE Conference on Computer Communications, vol. 4, pp. 2673–2684 (2004)
10. Bekris, K.E., Argyros, A.A., Kavraki, L.E.: Angle-based methods for mobile robot navigation: Reaching the entire plane. In: Proc. IEEE International Conference on Robotics and Automation (ICRA), pp. 2373–2378 (2004)
11. Nagpal, R., Shrobe, H.E., Bachrach, J.: Organizing a Global Coordinate System from Local Information on an Ad Hoc Sensor Network. In: Zhao, F., Guibas, L.J. (eds.) IPSN 2003. LNCS, vol. 2634, pp. 333–348. Springer, Heidelberg (2003)
12. Ranganathan, P., Morton, R., Richardson, A., Strom, J., Goeddel, R., Bulic, M., Olson, E.: Coordinating a team of robots for urban reconnaissance. In: Proceedings of the Land Warfare Conference (LWC) (November 2010)
13. Moore, D., Leonard, J., Rus, D., Teller, S.: Robust distributed network localization with noisy range measurements. In: Proc. 2nd International Conference on Embedded Networked Sensor Systems, pp. 50–61 (2004)
14. Priyantha, N.B., Chakraborty, A., Balakrishnan, H.: The cricket location-support system. In: Proceedings of the 6th Annual International Conference on Mobile Computing and Networking, Boston, Massachusetts, United States, pp. 32–43. ACM (2000)
15. Lee, S., Amato, N.M., Fellers, J.: Localization based on visibility sectors using range sensors. In: Proc. IEEE Int. Conf. Robot. Autom. (ICRA), pp. 3505–3511 (2000)
16. Sundaram, B., Palaniswami, M., Reddy, S., Sinickas, M.: Radar localization with multiple unmanned aerial vehicles using support vector regression. In: Third International Conference on Intelligent Sensing and Information Processing, ICISIP 2005, pp. 232–237 (2005)
17. Montesano, L., Gaspar, J., Santos-Victor, J., Montano, L.: Cooperative localization by fusing vision-based bearing measurements and motion. In: 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2005), pp. 2333–2338 (August 2005)
18. Loizou, S.G., Kumar, V.: Biologically inspired bearing-only navigation and tracking. In: 2007 46th IEEE Conference on Decision and Control, pp. 1386–1391 (2007)
19. Ghrist, R., Lipsky, D., Poduri, S., Sukhatme, G.S.: Surrounding Nodes in Coordinate-Free Networks. In: Skella, S., Amato, N.M., Huang, W.H., Mishra, B. (eds.) Algorithmic Foundations of Robotics VII. STAR, vol. 47, pp. 409–424. Springer, Heidelberg (2006)
20. Bilò, D., Disser, Y., Mihalák, M., Suri, S., Vicari, E., Widmayer, P.: Reconstructing Visibility Graphs with Simple Robots. In: Kutten, S., Žerovnik, J. (eds.) SIROCCO 2009. LNCS, vol. 5869, pp. 87–99. Springer, Heidelberg (2010)
21. Whiteley, W.: Matroids from Discrete Geometry. *AMS Contemporary Mathematics* 197, 171–312 (1996)

22. Bruck, J., Gao, J., Jiang, A.(A.): Localization and routing in sensor networks by local angle information. *ACM Transactions on Sensor Networks* 5(1), 1–7 (2009)
23. Cornejo, A., Khabbazian, M., McLurkin, J.: Theory of scale-free coordinates for multi-robot system with bearing-only sensors. Technical Report (2011), <http://mrs1.rice.edu/publications>
24. Horton, J.D.: A Polynomial-Time algorithm to find the shortest cycle basis of a graph. *SIAM Journal on Computing* 16(2), 358 (1987)
25. Olson, E.: Apriltag: A robust and flexible multi-purpose fiducial system. Technical report, University of Michigan APRIL Laboratory (May 2010)
26. McLurkin, J.: Measuring the accuracy of distributed algorithms on Multi-Robot systems with dynamic network topologies. In: 9th International Symposium on Distributed Autonomous Robotic Systems (DARS) (2008)
27. Rykowski, J.B.: Pose Estimation With Low-Resolution Bearing-Only Sensors. M.S. thesis, Rice University (2011)
28. O’Kane, J.M., LaValle, S.M.: Comparing the power of robots. *The International Journal of Robotics Research* 27(1), 5 (2008)

Mapping Polygons with Agents That Measure Angles

Yann Disser, Matúš Mihalák, and Peter Widmayer

Abstract. We study the problem of mapping an initially unknown environment with autonomous mobile robots. More precisely, we consider simplistic agents that move from vertex to vertex along the boundary of a polygon and measure angles at each vertex. We show that such agents are already capable of drawing a map of any polygon in the sense that they can infer the exact geometry up to similarity. Often, such tasks require the agent to have some prior bound on the size of the environment. In this paper, we provide an efficient reconstruction algorithm that *does not* need any a priori knowledge about the total number of vertices.

1 Introduction

From the perspective of a technology enthusiast it is fascinating to see more and more problems in our daily lives being taken over by autonomous robots. We have developed a good understanding of how to make sophisticated robots perform impressive tasks like navigating cars through traffic. On the other hand, we still do not understand how much sophistication is actually needed for even the most fundamental problems. In other words: How powerful do the sensors and movement capabilities of a robot need to be at the very least for a given problem? From a theoretical point of view, trying to answer this question leads to a better insight of both the essential difficulty of the problem at hand and of the inherent power of different sensors. From a practical point of view, investigating the question might enable us to replace sophisticated robot designs with cheap and robust counterparts that can more easily be produced in masses.

Yann Disser · Matúš Mihalák · Peter Widmayer
ETH Zurich, Institute of Theoretical Computer Science
e-mail: {ydisser,mmihalak,widmayer}@inf.ethz.ch

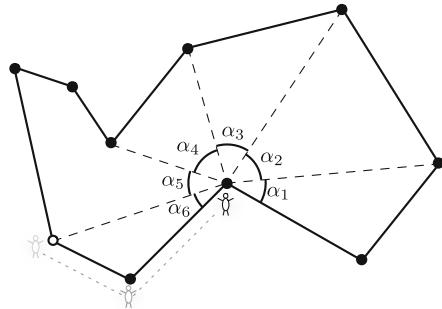


Fig. 1 The agent moves from vertex to vertex along the boundary and measures angles at each vertex

This paper aims to be a step towards the ultimate goal of developing a complete classification of robotic tasks and required capabilities. Our focus is on the problem of drawing a map, which is a core problem in almost any task that requires a robot to operate in an initially unknown environment. We approach the problem from a theoretical perspective and base our analysis on simplistic models for both environments and robots. More precisely, we take the initially unknown environment to be a simple polygon and consider the following minimalistic *agent* model instead of dealing with realistic *robots*.

Ideally, we want the agent model to be as simplistic as possible, while still allowing the agent to draw a complete map of its surrounding polygon. We make the following modeling assumptions, where \mathcal{P} refers to the environment and a *line of sight* refers to a line segment lying inside \mathcal{P} connecting two vertices (cf. Fig. 1):

- The agent is initially located at a vertex of \mathcal{P} .
- The agent can move along the boundary of \mathcal{P} .
- While situated at a vertex v , the agent can order the lines of sight as they appear in a counter-clockwise sweep of \mathcal{P} at v .
- While situated at a vertex v , the agent can perceive the angle between any two lines of sight.

Note that we do not assume the agent to be aware of the number n of vertices of \mathcal{P} . Since we are interested whether the information that the agent can collect suffices to draw a map of \mathcal{P} at all, we do not impose limitations on the memory or computational power available to the agent. Of course, given the choice, we prefer efficient mapping algorithms.

So, does our model empower the agent to draw a map? The answer to this question depends on what we consider to be a “map” of a polygon. The only geometrical data that is available to our agent is encoded in angles.

This means that we cannot hope the agent to do better than to draw a map that describes the geometry of the polygon up to similarity. The question is whether the data available through the sensors of the agent alone already *uniquely* determines the shape of \mathcal{P} , or whether there can be two polygons of different shape that yield the same observations.

In this paper, we show that the agent as modeled above can infer the shape of any polygon uniquely, and hence draw a map.

1.1 Related Work

This paper is a follow-up to our earlier results in [9], where we give an algorithm for reconstructing a polygon from a list of angle measurements. The difference towards this earlier paper is that we consider an agent that does not know the number of vertices n a priori. When n is known, it is easy for the agent to collect all data initially. In that case we may as well ignore the agent and view the problem as a geometrical reconstruction problem, where a list of angle-lists is given and we look for the shape of the polygon. In this paper, we make use of a central result of [9] to design an algorithm that allows the agent to reconstruct the visibility graph incrementally while collecting angle measurements along the boundary. We also use the observations made in [7] to improve the efficiency of the resulting algorithm.

Many studies related with our setting are concerned with geometrical reconstruction problems, where geometrical objects have to be reconstructed from given measurement data, i.e., without considering agents that have to gather the data first. The main focus of work in this area, typically, is to decide whether a certain kind of data encodes enough information for a reconstruction or not. Work in this area includes [8, 20, 23]. Another variant of geometrical reconstruction problems ask for any object compatible to given measurement data without requiring the solution to be unique. The main focus for studies for this variant usually lies in finding efficient algorithms. Work in this area includes [2, 16, 21, 22].

There have been different approaches to modeling minimalistic agents for various environments and objectives [1, 11, 17, 24]. Some works have even established hierarchies of agent models and frameworks that allow to compare otherwise unrelated models [4, 10, 19]. We based our model on the one proposed in [24], which has also been studied previously in [3, 4, 6, 5, 12, 18].

The agents (called *Bitbots*) that were studied in [17] are similar to ours in that they can only move along the walls of the environment. While Bitbots have somewhat more powerful movement capabilities than our agents, they are much more limited in their sensing. As it turns out, this makes them incapable of inferring even a topological map [17].

Another similar setting was considered in [3]. This study uses the same movement capabilities as our model, but again only provides agents with

very basic, combinatorial sensors. And again, it turns out that such agents cannot even infer a topological map of their environment.

There is also an example of agents with weaker sensors than ours which *can* infer a topological map: The agents in [6] can only distinguish convex from reflex angles, but they are allowed to move through the interior of the environment and they need to be provided with knowledge of at least a bound on the total number of vertices. In addition, the reconstruction algorithm that is developed in [6] is extremely inefficient compared to the one we introduce here.

2 The Mapping Problem

Throughout this paper, we consider the exploration of a simple polygon \mathcal{P} by an autonomous agent. In particular, we assume \mathcal{P} not to have holes or self-intersections. We let V denote the set of vertices of \mathcal{P} and let $n = |V|$ be its size. We assume the agent to initially be located at a vertex v_0 of \mathcal{P} , and denote all other vertices of \mathcal{P} by v_1, v_2, \dots, v_{n-1} in the order in which appear along a counterclockwise tour of the boundary starting at v_0 . We express the cyclic order of the vertices in notation by writing $v_{i\pm k}$ for $v_{i\pm k \bmod n}$.

In the model which we adopt the agent can (1) move from vertex to vertex along the boundary of \mathcal{P} and (2) make local observations in \mathcal{P} while situated at a vertex. The agent is aware that its environment is a simple polygon, but other than that it has no initial knowledge about \mathcal{P} . In particular, it has no knowledge about n . This means that, initially, the agent does not know how many vertices it needs to visit in order to complete an exact tour of the boundary.

We now define precisely what local observations the agent can make at a vertex v_i . All vertices v_j which can be connected to v_i via straight line segments in \mathcal{P} (including the boundary) are considered to be *visible* to v_i and hence to be visible to the agent. We refer to the corresponding line segments as *lines of sight*, and we order the vertices visible to v_i *locally* according to the angles formed inside \mathcal{P} by the $\overline{v_iv_{i+1}}$ and the corresponding line of sight. The local observation of the agent at v_i is encoded in the vector of angles $\boldsymbol{\alpha}(v_i) = (\alpha_1, \alpha_2, \dots)$, where α_l is the angle inside \mathcal{P} between the l -th and the $(l+1)$ -th line segment. We refer to $\boldsymbol{\alpha}(v_i)$ as the *angle measurement at v_i* . From $\boldsymbol{\alpha}(v_i)$ it is easy to infer the angle spanned by any two non-consecutive vertices visible to v_i , simply by summing up all enclosed angles. On the other hand, the angle measurement does not provide the identity of the visible vertices, i.e., the agent cannot tell which vertices of V are visible to v_i just by inspecting $\boldsymbol{\alpha}(v_i)$. Of course, the agent always knows that the first visible vertex is v_{i+1} and the last one is v_{i-1} .

In order to argue that an agent can always draw a map, we need to provide an algorithm that governs how the agent moves and reasons about the

knowledge it has gained so far. We use the term *exploration strategy* to refer to such an algorithm. An exploration strategy consists of a succession of three types of operations: (1) moving to a neighboring vertex and collecting the angle measurement at the new location, (2) making computations that may involve any of the data collected so far, and (3) terminating. The efficiency of an exploration strategy is measured upon termination by the number of physical moves and sensing operations, the number of atomic computations, and the required memory. We say that an exploration strategy computes a certain quantity if it always terminates after a finite number of moves and atomic computations, in a state where the agent knows the desired quantity.

The *shape* of \mathcal{P} is given by its geometry disregarding scale, rotation and translation. We say that an exploration strategy *reconstructs* a polygon \mathcal{P} if an agent executing it in \mathcal{P} knows the shape of \mathcal{P} upon termination. We say the agent can solve the *mapping problem* if there is an exploration strategy that reconstructs every polygon \mathcal{P} . As long as the agent only measures angles, computing the shape is the best it can hope for. Any scaled, rotated or translated version of a polygon \mathcal{P} yields the exact same observations and can therefore not be distinguished from \mathcal{P} by any exploration strategy for the agent.

Our main result relies on the fact that the agent can efficiently compute the visibility graph G_{vis} of \mathcal{P} , which is defined as follows. Every vertex of \mathcal{P} is a node of G_{vis} , and there is an edge in G_{vis} for every two vertices of \mathcal{P} that see each other. The number of vertices and edges of G_{vis} is denoted by n, m , respectively, throughout the paper. We will argue that from G_{vis} together with the angle measurement for every vertex, the agent can efficiently compute the shape of \mathcal{P} . Note that the characterization of visibility graph is a long-standing open problem [13, 15, 14].

3 Reconstructing the Shape of \mathcal{P}

We first show that once the agent knows the visibility graph G_{vis} of \mathcal{P} and the angle measurements at each vertex, it can compute the shape of \mathcal{P} . We show this by considering a subgraph of G_{vis} that corresponds to a triangulation of \mathcal{P} , and by using this subgraph to construct the shape of \mathcal{P} . After we established that knowing G_{vis} allows the agent to solve the mapping problem we move on to our main result, namely the design of an exploration strategy that computes the visibility graph.

3.1 Inferring the Shape from G_{vis}

In this section we argue that the visibility graph $G_{\text{vis}} = (V, E)$ of \mathcal{P} together with the angle measurements $\alpha(v_i)$ for every vertex $v_i \in V$ uniquely

determine the shape of \mathcal{P} . We forget the agent for the moment and establish how to efficiently reconstruct the shape of \mathcal{P} from this data. Once the agent has acquired knowledge of G_{vis} and the angle measurements, it can obtain the shape by using the computation described below.

Consider a vertex $v_i \in V$ of degree d in G_{vis} . Let $\alpha(v_i) = (\alpha_1, \alpha_2, \dots, \alpha_{d-1})$ be the angle measurement at v_i , and let $N(v_i) = \{v_{i+\delta_1}, v_{i+\delta_2}, \dots, v_{i+\delta_d}\}$ be its neighborhood in G_{vis} , with $1 = \delta_1 < \delta_2 < \dots < \delta_d = n - 1$. For geometrical reasons, it turns out that the *global* vertex $v_{i+\delta_l}$ is exactly the l -th vertex *locally* visible to v_i . This means in particular that α_l is the angle inside \mathcal{P} between the line segments $\overline{v_i v_{i+\delta_l}}$ and $\overline{v_i v_{i+\delta_{l+1}}}$. Knowing the angle measurements hence implies knowing the angle between $\overline{v_i v_{i+\delta_l}}$ and $\overline{v_i v_{i+\delta_k}}$ for any $1 \leq l < k \leq d$. This, in turn, means that we can enhance G_{vis} by assigning an angle to every pair of edges at each vertex of G_{vis} .

We give the following fact without its simple but technical proof. Intuitively, if two neighbors of a vertex v_i do not see each other, there needs to be something obstructing their line of sight. This in turns implies that v_i sees another vertex in-between. A proof can be found in [13, 14] (Lemma 3, resp. Lemma 6.2.3).

Proposition 1. *Let $v_i \in V$ be a vertex of degree d and let $N(v_i) = \{v_{i+\delta_1}, v_{i+\delta_2}, \dots, v_{i+\delta_d}\}$ be its neighborhood in G_{vis} , with $1 = \delta_1 < \delta_2 < \dots < \delta_d = n - 1$. Then $v_{i+\delta_l}$ and $v_{i+\delta_{l+1}}$ see each other for every $1 \leq l < d$.*

We proceed with the proof of the main theorem of this section.

Theorem 1. *The shape of a polygon \mathcal{P} can be inferred from its visibility graph together with its angle measurements in time and space $\mathcal{O}(m)$.*

Proof. We show how to inductively obtain the shape of a polygon \mathcal{P} from its enhanced visibility graph G_{vis} . If $|V| = 3$, the polygon is a triangle and, since v_0, v_1, v_2 need to appear in counter-clockwise order, the shape is uniquely determined.

Consider a polygon \mathcal{P} with $|V| > 3$ vertices with its enhanced visibility graph G_{vis} . We distinguish two cases concerning the neighborhood of v_1 (cf. Fig. 2). If v_1 has degree two, then v_0, v_2 see each other by Proposition 1. By induction, we can compute both the shape of the subpolygon induced by v_2, v_3, \dots, v_0 as well as the shape of the triangle v_0, v_1, v_2 . We obtain the shape of \mathcal{P} by combining both these shapes. If v_1 has degree at least 3, we determine the second visible vertex v_i of v_1 . By induction, we can compute the shape of the subpolygons induced by v_1, v_2, \dots, v_i and $v_i, v_{i+1}, \dots, v_0, v_1$. We can again combine both shapes to obtain the shape of \mathcal{P} .

A careful recursive implementation of this algorithm runs in time $\mathcal{O}(m)$ and essentially needs the space required for storing G_{vis} . \square

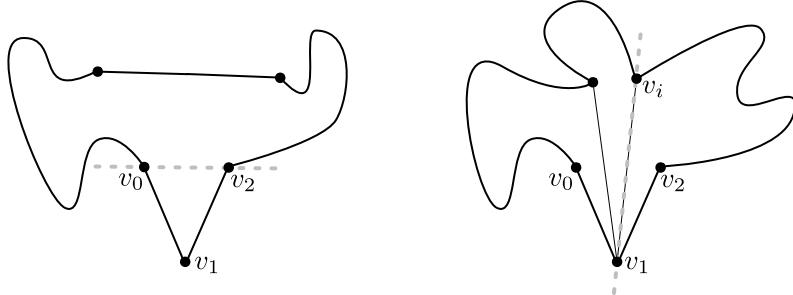


Fig. 2 The two different cases in the induction for deducing the shape of \mathcal{P} from G_{vis}

3.2 An Exploration Strategy to Compute G_{vis}

In this section, we introduce an exploration strategy that builds G_{vis} incrementally. The following definitions will allow us to describe increasing portions of G_{vis} that have already been inferred by our strategy. The graph $G_i^j = (V_i^j, E_i^j)$ is defined to be the subgraph of G_{vis} induced by the vertices v_i, v_{i+1}, \dots, v_j . Observe that, in this notation, $G_{\text{vis}} = G_0^{n-1}$. The degree of a vertex v_k in G_i^j is denoted by $d_i^j(v_k)$, and we write $d_k := d_0^{n-1}(v_k)$ for the degree of v_k in the visibility graph G_{vis} of \mathcal{P} .

The key of our exploration strategy is to maintain a growing subgraph of G_{vis} that has already been determined. After each new measurement, the agent incorporates the new data to obtain a bigger part of G_{vis} . More precisely, in step t , the agent moves from vertex v_{t-1} to vertex v_t and computes G_0^t from G_0^{t-1} and the new angle measurement at v_t . The main ingredient to our exploration strategy originates from [7, 9] and can be formalized as follows.

Lemma 1 ([7, 9]). *The graph G_i^j , $0 \leq i < j$, can be computed from G_i^{j-1} , G_{i+1}^j , and $\alpha(v_i), \alpha(v_{i+1}), \dots, \alpha(v_j)$ in constant time.*

The main problem that has to be solved for the computation in Lemma 1 can be reformulated as follows: Given the edges $E_i^j \setminus \{v_i, v_j\}$ and the angles given by $\alpha(v_i), \alpha(v_{i+1}), \dots, \alpha(v_j)$, decide whether $\{v_i, v_j\} \in E_i^j$, i.e., decide whether v_i sees v_j or not. In [9], we provided a necessary and sufficient condition for making this decision. Later, Chen and Wang [7] showed how to test the condition in constant time. The notation that we use here is motivated by the incremental nature of the strategy that the agent has to employ, and therefore different from the one used in the original papers. For convenience, we describe the inner workings of Lemma 1 in Sect. 3.3.

Theorem 2. *There is an exploration strategy that computes the shape of any polygon \mathcal{P} using $\mathcal{O}(n)$ moves and sensing operations, $\mathcal{O}(n^2)$ atomic computations, and $\mathcal{O}(m)$ bits of memory.*

Proof. Lemma 1 provides the means for our incremental exploration strategy. We give a listing of the strategy in Algorithm 1.1. It uses the operations SENSE and MOVE to govern the physical actions of the agent. When located at a vertex v_k , the operation SENSE returns $\alpha(v_k)$, while the operation MOVE moves the agent one step along the boundary to v_{k+1} .

Algorithm 1.1. Algorithm for computing the visibility graph G_{vis} .

```

function compute $G_{\text{vis}}$ 
 $G_0^0 \leftarrow (v_0, \emptyset);$ 
 $\alpha(v_0) \leftarrow \text{SENSE};$ 
 $j \leftarrow 0$ 
while  $d_0^j(v_0) < |\alpha(v_0)| + 1$ :
     $j \leftarrow j + 1;$ 
    MOVE;
     $\alpha(v_j) \leftarrow \text{SENSE};$ 
     $G_j^j = (v_j, \emptyset);$ 
    for  $l \leftarrow j - 1, j - 2, \dots, 1, 0$ 
        compute  $G_l^j$  from  $G_{l+1}^j, G_l^{j-1}, \alpha(v_l), \alpha(v_{l+1}), \dots, \alpha(v_j)$ ; (see Lemma 1)
    return  $G_0^j;$ 

```

The algorithm maintains the invariant that, after each iteration of the outer loop, G_l^r has been computed for all $0 \leq l \leq r < j$. In particular, $G_{\text{vis}} = G_0^{n-1}$ has been computed after $n - 1$ iterations. Because $d_0^{n-1}(v_0) = |\alpha(v_0)| + 1$, this means that the algorithm terminates after $n - 1$ iterations and returns $G_{\text{vis}} = G_0^{n-1}$. The agent performs exactly $n - 1$ moves and n sensing operations. The total computational running time is $\mathcal{O}(n^2)$, due to Lemma 1. Together with Theorem 1, we have proven the claim. \square

3.3 Deciding Whether v_i Sees v_j

In this section, we describe in more detail how to construct G_i^j from G_i^{j-1} and G_{i+1}^j when the angle measurements $\alpha(v_i), \alpha(v_{i+1}), \dots, \alpha(v_j)$ are known (Lemma 1). Since all the edges in $E_i^j \setminus \{v_i, v_j\}$ are already present in either G_i^{j-1} or G_{i+1}^j , the problem boils down to deciding whether $\{v_i, v_j\} \in E_i^j$, i.e., whether v_i sees v_j . Of course, this problem is non-trivial only when $v_j \neq v_{i+1}$. Consider Fig. 3 alongside the discussion below.

Recall that, in our notation, v_i sees d_i vertices in total. Of those vertices, we can identify the first $d_i^{j-1}(v_i)$, simply by looking at G_i^{j-1} . Now, if v_i actually sees v_j , then v_j is the first unidentified vertex of v_i , i.e., the $(d_i^{j-1}(v_i) + 1)$ -th vertex among the vertices visible to v_i . Similarly, for vertex v_j , the last

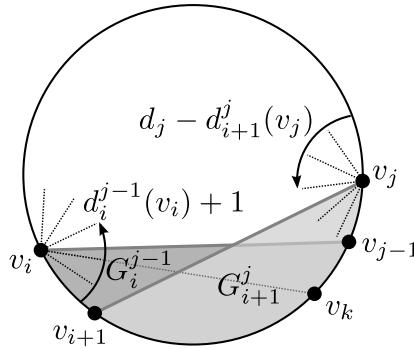


Fig. 3 Illustration of the construction of G_i^j from G_i^{j-1} and G_{i+1}^j . The difficulty is to decide whether v_i sees v_j .

$d_{i+1}^j(v_j)$ vertices visible to v_j are identified. If v_j sees v_i , then v_i is the last unidentified vertex of v_j , i.e., the $(d_j - d_{i+1}^j(v_j))$ -th visible vertex of v_j .

By Proposition 1, the $d_i^{j-1}(v_i)$ -th and the $(d_i^{j-1}(v_i) + 1)$ -th visible vertex of v_i see each other. If v_i would see v_j , then v_j would be this $(d_i^{j-1}(v_i) + 1)$ -th visible vertex of v_i , and hence v_i , the $d_i^{j-1}(v_i)$ -th visible vertex of v_i , and v_j would form a triangle in \mathcal{P} . The three enclosed angles of this triangle would sum up to 180 degrees. Whether this is the case can be determined by inspecting G_i^{j-1} and G_{i+1}^j : From G_i^{j-1} we know the global identity v_k of the $d_i^{j-1}(v_i)$ -th visible vertex of v_i , and from G_{i+1}^j we can check whether v_k sees v_j . If v_k does not see v_j , we can conclude that v_i does not see v_j . Otherwise, let α denote the angle at v_i between v_k and the $(d_i^{j-1}(v_i) + 1)$ -th visible vertex, let β denote the angle at v_k between v_j and v_i , and let γ denote the angle at v_j between the $(d_j - d_{i+1}^j(v_j))$ -th visible vertex and vertex v_k . Again, if $\alpha + \beta + \gamma \neq 180^\circ$, we can conclude that v_i does not see v_j . What is more surprising is that the opposite holds as well, i.e., if $\alpha + \beta + \gamma = 180^\circ$, then v_i and v_j must see each other. This has been proved formally in [7, 9] – we illustrate the idea behind the proof in Fig. 4. Intuitively, if $\alpha + \beta + \gamma = 180^\circ$, then the triangle v_i, v_k, v_j is empty, and hence nothing can obstruct the line of sight between v_i and v_j .

Overall, this necessary and sufficient criterion gives us a simple and computationally efficient means to check whether v_i and v_j see each other, and thus whether or not $\{v_i, v_j\} \in E_i^j$.

Lemma 2 ([9]). *Let $v_i, v_j \in V$ with $v_j \neq v_{i+1}$ and let $v_k, \alpha, \beta, \gamma$ be defined as before, then*

$$v_i \text{ sees } v_j \text{ if and only if } \alpha + \beta + \gamma = 180^\circ.$$

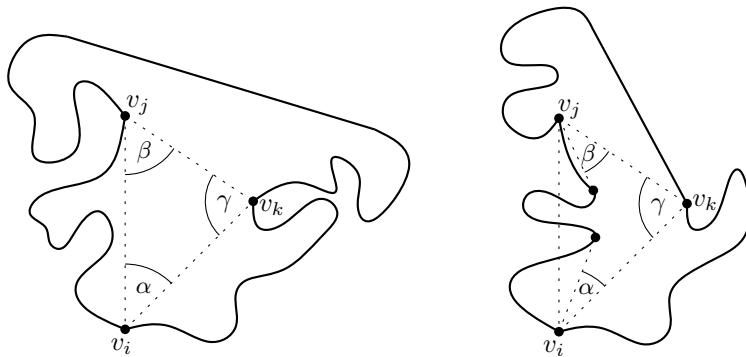


Fig. 4 An illustration that v_i sees v_j if and only if $\alpha + \beta + \gamma = 180^\circ$

4 Conclusion

We have presented an exploration strategy for an agent that moves along the boundary of a polygon and observes the angles formed by the lines of sight at each vertex. We have shown that the agent can infer the shape of the polygon in $O(n)$ moves using $O(n^2)$ atomic computations and $O(m)$ memory. The exploration strategy we presented is based on an incremental construction of the visibility graph, and does not need any knowledge about n , which is an improvement over previous results. This, in fact, is a rare property when considering problems for autonomous agents. We usually need some means to “break the symmetry”, such as pebbles that can be used to mark vertices.

A natural open problem is to achieve a better running time than $\mathcal{O}(n^2)$ if the underlying visibility graph is sparse. We can of course not do better than $\mathcal{O}(m)$, but the gap remains. Maybe $\mathcal{O}(m)$ atomic computations are enough?

References

1. Ando, H., Oasa, Y., Suzuki, I., Yamashita, M.: Distributed memoryless point convergence algorithm for mobile robots with limited visibility. *IEEE Transactions on Robotics and Automation* 15(5), 818–828 (1999)
2. Biedl, T., Durocher, S., Snoeyink, J.: Reconstructing Polygons from Scanner Data. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) *ISAAC 2009*. LNCS, vol. 5878, pp. 862–871. Springer, Heidelberg (2009)
3. Bilò, D., Disser, Y., Mihalák, M., Suri, S., Vicari, E., Widmayer, P.: Reconstructing visibility graphs with simple robots. *Theoretical Computer Science* 444, 52–59 (2012)
4. Brunner, J., Mihalák, M., Suri, S., Vicari, E., Widmayer, P.: Simple Robots in Polygonal Environments: A Hierarchy. In: Fekete, S.P. (ed.) *ALGOSENSORS 2008*. LNCS, vol. 5389, pp. 111–124. Springer, Heidelberg (2008)
5. Chalopin, J., Das, S., Disser, Y., Mihalák, M., Widmayer, P.: How Simple Robots Benefit from Looking Back. In: Calamoneri, T., Diaz, J. (eds.) *CIAC 2010*. LNCS, vol. 6078, pp. 229–239. Springer, Heidelberg (2010)

6. Chalopin, J., Das, S., Disser, Y., Mihalák, M., Widmayer, P.: Telling convex from reflex allows to map a polygon. In: Proceedings of the 28th International Symposium on Theoretical Aspects of Computer Science, pp. 153–164 (2011)
7. Chen, D., Wang, H.: An improved algorithm for reconstructing a simple polygon from the visibility angles. Computational Geometry: Theory and Applications 45, 254–257 (2012)
8. Coullard, C., Lubiwi, A.: Distance visibility graphs. In: Proceedings of the 7th Annual Symposium on Computational Geometry, pp. 289–296 (1991)
9. Disser, Y., Mihalák, M., Widmayer, P.: A polygon is determined by its angles. Computational Geometry: Theory and Applications 44, 418–426 (2011)
10. Donald, B.R.: On information invariants in robotics. Artificial Intelligence 72(1–2), 217–304 (1995)
11. Ganguli, A., Cortés, J., Bullo, F.: Distributed deployment of asynchronous guards in art galleries. In: Proceedings of the 2006 American Control Conference, pp. 1416–1421 (2006)
12. Gfeller, B., Mihalák, M., Suri, S., Vicari, E., Widmayer, P.: Counting Targets with Mobile Sensors in an Unknown Environment. In: Kutyłowski, M., Cicchon, J., Kubiak, P. (eds.) ALGOSENSORS 2007. LNCS, vol. 4837, pp. 32–45. Springer, Heidelberg (2008)
13. Ghosh, S.K.: On recognizing and characterizing visibility graphs of simple polygons. Discrete and Computational Geometry 17(2), 143–162 (1997)
14. Ghosh, S.K.: Visibility Algorithms in the Plane. Cambridge University Press (2007)
15. Ghosh, S.K., Goswami, P.P.: Unsolved problems in visibility graph theory. In: Proceedings of the India-Taiwan Conference on Discrete Mathematics, pp. 44–54 (2009)
16. Jackson, L., Wismath, S.K.: Orthogonal polygon reconstruction from stabbing information. Computational Geometry 23(1), 69–83 (2002)
17. Katsev, M., Yershova, A., Tovar, B., Ghrist, R., LaValle, S.M.: Mapping and pursuit-evasion strategies for a simple wall-following robot. IEEE Transactions on Robotics 27(1), 113–128 (2011)
18. Komuravelli, A., Mihalák, M.: Exploring Polygonal Environments by Simple Robots with Faulty Combinatorial Vision. In: Guerraoui, R., Petit, F. (eds.) SSS 2009. LNCS, vol. 5873, pp. 458–471. Springer, Heidelberg (2009)
19. O’Kane, J.M., LaValle, S.M.: On comparing the power of robots. International Journal of Robotics Research 27(1), 5–23 (2008)
20. O'Rourke, J.: Uniqueness of orthogonal connect-the-dots. In: Toussaint, G.T. (ed.) Computational Morphology, pp. 97–104. North-Holland (1988)
21. Rappaport, D.: On the complexity of computing orthogonal polygons from a set of points. Technical Report SOCS-86.9, McGill University, Montreal, Canada (1986)
22. Sidlesky, A., Barequet, G., Gotsman, C.: Polygon reconstruction from line cross-sections. In: Proceedings of the 18th Annual Canadian Conference on Computational Geometry, pp. 81–84 (2006)
23. Snoeyink, J.: Cross-ratios and angles determine a polygon. Discrete and Computational Geometry 22(4), 619–631 (1999)
24. Suri, S., Vicari, E., Widmayer, P.: Simple robots with minimal sensing: From local visibility to global geometry. International Journal of Robotics Research 27(9), 1055–1067 (2008)

Counting Moving Bodies Using Sparse Sensor Beams

Lawrence H. Erickson, Jingjin Yu, Yaonan Huang, and Steven M. LaValle

Abstract. This paper examines the problem of determining the distribution of a number of indistinguishable moving bodies located in regions separated by sensor beams that can detect whether a body moves across them. We characterize the conditions under which an exact distribution of bodies can be determined, and compute bounds on the expected number of sensor observations required to determine this exact distribution for a certain movement model of the bodies.

1 Introduction

Consider determining, in a large office building with many rooms, how its anonymous occupants are scattered in the rooms. Such information can be of vital importance in scenarios such as coordinated building evacuation in an emergency or characterizing building usage for energy optimization. Let the *distribution* of the occupants be the precise number of occupants per room. The task is relatively easy if an initial distribution of the occupants is known: We may simply place sensor beams at doorways of rooms to figure out the change in population for each room as the occupants move around. Adding or subtracting the change from the initial count then yields the answer. But what if the a priori distribution of the occupants is unavailable? Could the task still be solved without additional sensors?

In this paper, we show that, somewhat surprisingly, the aforementioned simplistic sensor setup is still powerful enough for determining the occupant distribution as required, given only an initial total population and enough time. More precisely, we

Lawrence H. Erickson · Yaonan Huang · Steven M. LaValle
Department of Computer Science, University of Illinois at Urbana-Champaign
e-mail: {lericks4, lavalle}@uiuc.edu, huang134@illinois.edu

Jingjin Yu
Department of Electrical and Computer Engineering,
University of Illinois at Urbana-Champaign
e-mail: jyu18@uiuc.edu

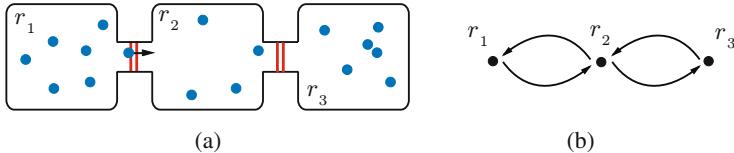


Fig. 1 a) A simple environment with three regions and two sensing units separating them, in which a blue body is moving from region r_1 to region r_2 . b) The corresponding graph representation of the environment, induced by the sensor placement.

characterize the capabilities and limitations of using beams to keep track of *bodies* moving around in a closed and bounded environment partitioned into *regions* by these beams. A beam detector is perhaps one of the simplest sensors: As a fixed sensor, it detects that a moving body passes through it but it cannot determine any other property of the body. With two of these, which we call a sensing unit, it is also possible to tell the movement direction of the passing body. Fixing such sensing units between regions of interest, we have at any moment the net number of bodies that have moved in/out of any region. Figure 1(a) shows one of the simplest environments under this model. Note that the regions and sensing units can be effectively represented as vertices and edges of a (directed) graph (Figure 1(b)). The bodies occupy the vertices of the graph; each sensor observation corresponds to a crossing of a body over an (directed) edge of the graph.

The contributions of this paper are twofold. First, we determine a necessary and sufficient condition on the initial distribution of bodies and the sensor history that allows the determination of an exact count of the bodies in each region. Second, we determine bounds on the expected number of sensor observations required to acquire a count of the bodies in each region for a specific movement model of the bodies. We show that for some starting distributions, the expected number of sensor observations required to determine the distribution is exponential in the number of bodies, while for other starting distributions (even in the same graph), the expected number of observations required to determine a distribution is polynomial in the number of bodies.

Various simple sensor models have been investigated in the task of target tracking and counting. Binary proximity sensors have been employed to estimate positions and velocities of a moving body using particle filters [1] and moving averages [11]. The performance limit of a binary proximity sensor network in tracking a single target was discussed and approached in [13], followed by an extension to the tracking of multiple targets [14]. The task of counting multiple targets using simple sensors was also studied in [2], in which the problem of accurately counting the number of targets with overlapping footprints using pressure sensor arrays was solved with a novel approach of integrating over Euler characteristics. In the works mentioned so far, the sensor network's aggregate sensing range must cover the targets of interest at all times. When only a subset of an environment is guarded, *word problems in groups* [4, 5] naturally arise. For the setup in which targets moving inside a 2D region are monitored with a set of sensor beams, [15] studied the issues of possible target locations, target path reconstruction up to homotopy, and path winding

numbers. Here, the surfacing of more interesting behaviors also induces an increase in computational complexity as well as convergence time, which is similar to what we observe in this paper. Controlling of sensorless wild bodies using various forms of gates was explored in [3], and some of the physically implemented gates used laser sensor beams to detect crossings between regions. Related to the control of sensorless wild bodies is the sensorless manipulation of polygonal parts studied in [6, 10].

If the limitation on using simple sensor models is lifted, research literature on target counting and tracking expands. Given the amount of papers and their diversified nature on this topic, we only mention a few of them. In [9] a simple virtual sensor, capable only of reporting visible features of the polygonal environment and the presence of indistinguishable targets, is able to count static targets using a minimal amount of data storage, though it is unclear how the simple virtual sensor could be implemented by simple physical sensors. In the domain of wireless sensor networks, the study of target counting and tracking is frequently coupled with communication efficiency and other concerns [7]. In [16], Simultaneous Localization and Mapping (SLAM) and Detection and Tracking of Moving Objects (DTMO) are combined to attack both problems at the same time. Real-time people counting with a network of image sensors is studied in [17].

The rest of the paper is organized as follows. Section 2 contains definitions that will be used throughout the paper. Section 3 describes the conditions under which the distribution of bodies can be determined. Section 4 describes a movement model for the bodies called the *exponential random movement model*. This section also describes a physical system that produces behavior closely approximated by the exponential random movement model. Section 5 contains an upper bound on the expected number of sensor observations required to count the bodies in each region when the behavior of the bodies matches the exponential random movement model. Section 6 contains a method of using the upper bound to estimate the number of bodies in the environment if that number is not known in advance. Section 7 contains a tight lower bound on the expected number of sensor observations required to count the number of bodies in each region when the behavior of the bodies matches the exponential random movement model. Section 8 contains results for a very simple 2-cycle environment. Section 9 discusses directions of future research.

2 Definitions

The vertices of an r -vertex directed graph G containing no sinks¹ are populated by a set M of *moving bodies*, in which $|M| = m$. This graph represents the regions and sensor beams of a physical environment (see Figure 1). The bodies are capable of travelling through the edges into other vertices. When a body moves, a sensor observation y is generated that identifies the edge that the body traversed. We assume that only one body moves through a sensor gate at a time, and we assume that the

¹ Graphs with sinks are excluded because bodies located in sinks are unable to move to different vertices, which renders them undetectable.

sensor beams operate without errors. The system is in the k th stage after the k th sensor observation has been generated. Let $V(G)$ be the vertex set of graph G . A *distribution* is an assignment of the moving bodies to vertices of G .

We assume that no information about the initial distribution of moving bodies is known. The *history information state* $\mathcal{I}_{\text{hist}}(k) = [y_1, \dots, y_k]$ is the list containing the first k sensor observations. The *bounds information state* $\mathcal{I}_{\text{bounds}}(k)$ consists of a total number of bodies m and two r -length vectors $[u_{1,k}, \dots, u_{r,k}]$ (the *upper bounds*) and $[\ell_{1,k}, \dots, \ell_{r,k}]$ (the *lower bounds*), where $u_{i,k}$ is the most bodies that could be in vertex v_i in stage k while remaining consistent with the previous sensor observations, and $\ell_{i,k}$ is the fewest bodies that could be in vertex v_i at stage k while remaining consistent with the previous sensor observations. The *interval length* $\text{length}(k, v_i)$ is the value of $u_{i,k} - \ell_{i,k}$ at stage k . A bounds information state with an interval length of 0 is called a *counting information state*. Once a counting information state has been reached, it is trivially easy to keep track of the number of bodies in each region. Since the bodies are indistinguishable to the sensors, a counting information state is an exact description of the system state. For some information state I , let $H(I)$ be the *hypothesis set* of I , defined as the set of distributions of bodies over the vertices that are consistent with the information state I .

When a distribution d and lower bound set L are presented in the form (d, L) , then the lower bound set L is implied to be consistent with the distribution d (in other words, there is no vertex v_i in which ℓ_i is greater than the actual number of bodies in v_i). A distribution and lower bound set (d, L) is *near-complete for vertex* v_i if the lower bound ℓ_j is equal to the number of bodies in v_j for all $j \neq i$, and the number of bodies in v_i is equal to $\ell_i + 1$. Note that (d, L) is near-complete if and only if $\sum \ell_i = m - 1$.

3 Counting Moving Bodies

In this section, the goal is to characterize what types of initial distribution and information history combinations allow the determination of a counting information state. As a first step, we show that, as long as a count of the total number of bodies in the graph is known in advance, the lower bounds of the bounds information state are sufficient to represent all that is known about the distribution of the bodies.

Theorem 1. *For all $k \in \mathbb{N}$ and all $v \in V(G)$, the interval length $\text{length}(k, v) = m - \sum_{i=1}^r \ell_{i,k}$.*

Proof. This will be shown inductively. As a base case, note that when $k = 0$, all interval lengths are m and $\ell_{i,0} = 0$ for all $1 \leq i \leq r$.

Assume that in stage $k - 1$, all intervals had length $m - \sum_{i=1}^r \ell_{i,k-1}$. Let the k th sensor observation be the transition of a body from v_s to v_t . If $\ell_{s,k-1} \geq 1$ and $u_{t,k-1} \leq m - 1$, then each hypothesis $h \in H(\mathcal{I}_{\text{bounds}}(k - 1))$ can be transformed into a distribution h' consistent with $\mathcal{I}_{\text{bounds}}(k - 1) + y_k$ by removing one body from v_s (there must be at least one as $\ell_{s,k-1} \geq 1$) and adding a body to v_t (there must be room for at least one more as $u_{t,k-1} \leq m - 1$). Therefore, $\mathcal{I}_{\text{bounds}}(k)$ is constructed by setting $\ell_{i,k} = \ell_{i,k-1}$ and $u_{i,k} = u_{i,k-1}$ for all $i \neq s, t$. For vertices v_s and

v_t , $\ell_{s,k} = \ell_{s,k-1} - 1$, $u_{s,k} = u_{s,k} - 1$, $\ell_{t,k} = \ell_{t,k-1} + 1$, and $u_{t,k} = u_{t,k-1} + 1$. Only the bounds of v_s and v_t have been changed, and their interval lengths remain unchanged because their upper and lower bounds have changed by the same amount.

Since all intervals have length $m - \sum_{i=1}^r \ell_{i,k-1}$ in stage $k-1$, for any hypothesis in $H(\mathcal{I}_{\text{bounds}}(k-1))$ in which there exists a vertex v_i that contains $u_{i,k-1}$ bodies, all other vertices must contain $\ell_{i,k-1}$ bodies. Therefore, if $\ell_{s,k-1} = 0$, then any hypothesis in which v_s contained 0 bodies in stage $k-1$ must have been false, which means that any hypothesis in which a vertex v_i contained $u_{i,k-1}$ bodies was also false. All other hypotheses in $H(\mathcal{I}_{\text{bounds}}(k-1))$ can be modified into hypotheses consistent with $\mathcal{I}_{\text{bounds}}(k-1) + y_k$ by moving one body from v_s to v_t . For each v_i , where $v_i \neq v_s$, there exists a hypothesis in $H(\mathcal{I}_{\text{bounds}}(k-1))$ where v_i contains $u_{i,k-1} - 1$ bodies and v_s contained one body. There also exists a hypothesis in $H(\mathcal{I}_{\text{bounds}}(k-1))$ where v_i contained no bodies. Therefore, $\mathcal{I}_{\text{bounds}}(k)$ is constructed by setting $\ell_{t,k} = \ell_{t,k-1} + 1$ and $u_{t,k} = u_{t,k-1}$. For each vertex v_i where $v_i \neq v_t$, the bounds are constructed by setting $\ell_{i,k} = \ell_{i,k-1}$ and $u_{i,k} = u_{i,k-1} - 1$. For each vertex, the difference between the upper and lower bound at stage k is one lower than the difference in stage $k-1$.

Note that if $u_{t,k-1} = m$, then for all other vertices v_i , the value of $\ell_{i,k-1}$ must be 0, as any hypothesis that places m bodies into v_t must place zero bodies in all other vertices. Therefore, the situation where a hypothesis is disqualified because a body moves into a vertex with an upper bound of m bodies is a special case of the situation in the previous paragraph.

Therefore, for all $k \in \mathbb{N}$ and all $v \in V(G)$, the interval length $\text{length}(k, v) = m - \sum_{i=1}^r \ell_{i,k}$. \square

Theorem 1 implies that keeping track of an upper bound for each individual vertex is redundant, as all upper bounds can be reconstructed using the lower bounds and the total number of bodies. This means that once the lower bound of a vertex v_i and the number of bodies in v_i are the same, we have in some sense learned all that we can from v_i . A corollary to Theorem 1 formalizes this notion. An *informative observation* is a sensor observation that decreases the interval length.

Corollary 2. *The information state $\mathcal{I}_{\text{bounds}}(k)$ is a counting information state if and only if for each vertex $v \in V(G)$ there exists a stage j , where $0 \leq j \leq k$, and v contains no bodies at stage j .*

Proof. Let d be some initial distribution of bodies. Suppose that there is a vertex p that did not empty out during the first k stages. Let q be a vertex such that $q \neq p$. Now, consider an initial distribution d' that is exactly the same as d except there is one more body in q and one fewer body in p . Since, starting from initial distribution d , the vertex p did not fully empty out in the first k stages, both d and d' are capable of producing $\mathcal{I}_{\text{hist}}(k)$. Note also that starting from initial distribution d' , the vertex q did not empty out in the first k stages. Therefore, it is impossible to determine if the starting distribution was d or d' . Since one would get a different distribution at stage k when starting from d than one would get by starting from d' , and $\mathcal{I}_{\text{hist}}(k)$ did not rule either out, that means that there is more than one hypothesis in $\mathcal{I}_{\text{bounds}}(k)$, so $\mathcal{I}_{\text{bounds}}(k)$ is not a counting information state.

In the other direction, suppose that for each vertex $v \in V(G)$, there exists a stage prior to k in which v was empty. If a body moves in to v , the lower bound on v increases. If a body leaves v when v has a lower bound of 1 or greater, the lower bound on v decreases. If a body leaves v when v has a lower bound of 0, an informative observation occurs. Suppose v started with b bodies and emptied out at stage j (where $0 \leq j \leq k$). Since v is empty at stage j , if there were a entries into v , then there were at least $a + b$ exits from v . Since there were at most a entries into v and v 's lower bound was initially 0, at most a exits could have decremented the lower bound of v . Therefore, at least b exits from v were informative observations. If each vertex empties out, then each vertex produces a number of informative observations equal to the number of bodies that it initially contained. Therefore, if each vertex empties out by stage k , there is one informative observation for each body, which means that the interval length is 0 at stage k . \square

4 The Exponential Random Movement Model

In the following sections, we will determine bounds on the expected number of steps required to converge to a counting information state. In order for these bounds to be well-defined, we require a model for the movement of the bodies. We have chosen to focus on a model in which each individual body has an equal probability of being the next body to move. For the purposes of our results, it is unimportant which vertex the body moves to if it has more than one option. We name this movement model the *exponential random movement model* because one situation in which this behavior occurs is when the amount of time that each body spends in between movements is described by an exponential random variable.

This model may be appropriate when the underlying event causing the movements is a Poisson process, such as the receiving of a phone call, or the detection of radiation with a geiger counter. A physical system that approximately produces this movement model is balls bouncing via specular bounces in polygonal regions separated by small doorways, similar to mathematical billiards. See Figure 2 for a representative environment and the distribution of the length of time a body remains in a single room between transitions.

Let C_2 be the two-vertex directed cycle. Given a graph G and a distribution (d, L) that is near-complete for a vertex $v_i \in V(G)$, let $C_2(d, L)$ be a distribution and lower bound set on C_2 that is near-complete for $v_1 \in C_2$ where v_1 contains the same number of bodies that d places in v_i .

Let $\text{Con}(G, d)$ be a random variable denoting the number of sensor observations required to converge to a counting information state in a directed graph G starting from an initial body distribution d with the bodies using the exponential random movement model. We will refer to $E[\text{Con}(G, d)]$ as the *expected exponential convergence time*, or the *EE-convergence time*. The word “exponential” in the term refers to the behavior of the random variables governing the motion of the bodies. The remainder of this paper will focus primarily on placing bounds on the EE-convergence time.

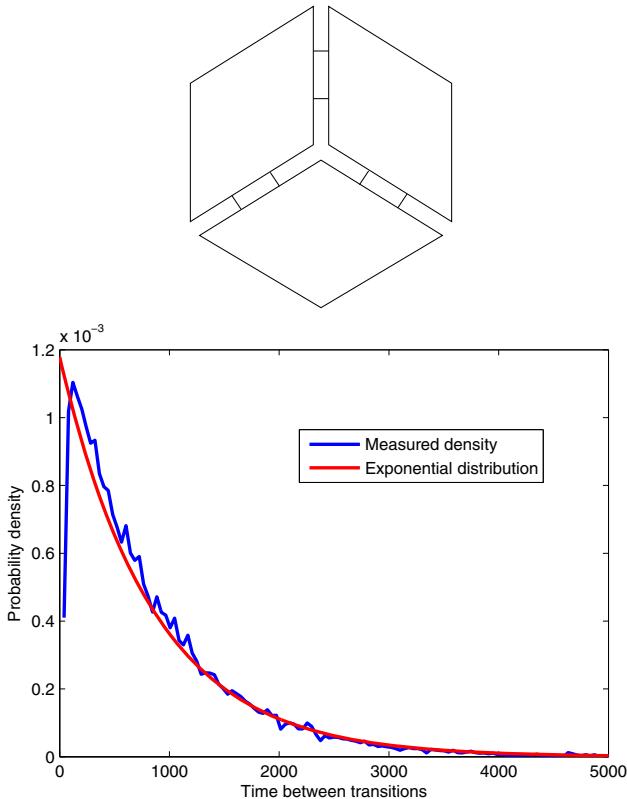


Fig. 2 [top] An environment with three symmetric regions. [bottom] The distribution of the lengths of time spent in a single region between transitions compared to an exponential random variable with the same mean. The values along the x -axis count simulation steps.

5 An Upper Bound on the EE-Convergence Time

Let graph G contain m bodies. For a distribution of bodies and set of lower bounds (d, L) , let $\text{Info}(G, d, L)$ be a random variable denoting the number of stages until an informative observation occurs. Let $\beta(G)$ be the maximum of $E[\text{Info}(G, d, L)]$ over all choices of (d, L) .

Lemma 3. *If $E[\text{Info}(G, d, L)] = \beta(G)$, then (d, L) is near-complete.*

Proof. Suppose that the lemma is false and let L be any set of lower bounds consistent with d such that $\sum_{\ell_i \in L} \ell_i < m - 1$ and $E[\text{Info}(G, d, L)]$ is maximal. Let L' be a set of lower bounds consistent with d such that $\sum_{\ell'_i \in L'} \ell'_i = m - 1$ and for all $1 \leq i \leq r$, $\ell'_i \geq \ell_i$. Note that, because $\sum_{\ell'_i \in L'} \ell'_i = m - 1$, there is exactly one vertex p where p 's lower bound in L' is not equal to its number of bodies (the two values differ by exactly one body). Similarly, either there exists a vertex q such that $q \neq p$

and q 's lower bound in L is also not equal to its number of bodies, or p 's lower bound in L differs from its number of bodies by two or more.

Let S' be the set of finite sequences of sensor observations that, starting from L' and d , contain exactly one informative observation, located at the end of the sequence. For $s \in S$, let $p(s)$ be the probability that sequence s occurs. Note that $p(s)$ is only dependent on s and d , not on L' . Note that $E[\text{Info}(G, d, L')] = \sum_{s' \in S'} p(s')|s'|$. Since the interval length for L' is one, each sequence in S' must at some point move all bodies out of p .

Since p also has a lower bound in L that differs from its number of bodies, any sequence $s' \in S'$ contains a minimum prefix s that is also an informative observation. Note that $E[\text{Info}(G, d, L)] = \sum_{s' \in S} p(s')|s|$. If p has different lower bounds in L and L' , then always $s \neq s'$, as an informative observation for L will be made when p empties all but one body. Otherwise, the aforementioned vertex q exists, and $s \neq s'$ if q is emptied out prior to p emptying out. Since $|s| \leq |s'|$, and there exists at least one sequence where $|s| < |s'|$, we get that $E[\text{Info}(G, d, L)] = \sum_{s' \in S} p(s')|s| < \sum_{s' \in S} p(s')|s'| = E[\text{Info}(G, d, L')]$. However, this contradicts the maximality of $E[\text{Info}(G, d, L)]$.

Therefore, if $E[\text{Info}(G, d, L)] = \beta(G)$, then $\sum_{\ell_i \in L} \ell_i = m - 1$. \square

We will now obtain a slightly stronger result for C_2 that will be useful later.

Lemma 4. *The expected value of $\text{Info}(C_2, d, L)$ is maximized when (d, L) is near-complete for some vertex v_1 , and v_1 contains all m bodies.*

Proof. For a two vertex graph with one edge, Lemma 3 implies that $E[\text{Info}(C_2, d, L)]$ is maximized when there exists a vertex (called v_2), in which the lower bound of v_2 is equal to the number of bodies in v_2 , and another vertex v_1 in which the lower bound of v_1 is one less than the number of bodies in v_1 . An informative observation will only occur if v_1 completely empties.

Let d and L be the distribution where d places all m bodies into v_1 and the lower bound ℓ_1 is equal to $m - 1$. Let d^* be any other distribution, and let L^* be a set of lower bounds such that (d^*, L^*) is near-complete for v_1 . Note that, for both (d, L) and (d^*, L^*) , an informative observation is only possible if v_1 empties completely. However, since d^* places fewer bodies in v_1 than d places in v_1 , any sequence of sensor observations that results in an informative observation starting from (d, L) must result in (d^*, L^*) as an intermediate distribution and set of bounds. Therefore, $E[\text{Info}(C_2, d, L)] = E[\text{Info}(C_2, d^*, L^*)] + E[\text{Catchup}]$, where Catchup is a random variable denoting the number of sensor observations required to reach (d^*, L^*) from (d, L) . Since Catchup is at least one (because $d \neq d^*$), we have that $E[\text{Info}(C_2, d, L)] > E[\text{Info}(C_2, d^*, L^*)]$. \square

We next use $E[\text{Info}(C_2, d_2, L_2)]$ to bound $E[\text{Info}(G, d, L)]$.

Lemma 5. *If (d, L) is near-complete for $v_i \in V(G)$ and $(d_2, L_2) = C_2(d, L)$, then $E[\text{Info}(G, d, L)] \leq E[\text{Info}(C_2, d_2, L_2)]$.*

Proof. An informative observation only occurs if v_i empties completely. We will call a body movement *progressive* if the body moves out of v_i . We will call a body

movement *regressive* if the body moves into v_i . We will call a body movement *neutral* if it is not progressive or regressive.

In C_2 , neutral movements are impossible. If there are z bodies in v_1 , then there is a z/m probability of a progressive move, and an $(m-z)/m$ probability of a regressive movement.

Each body has an identical probability of being selected as the next body to move. If there are z bodies in v_i , then there is a z/m probability of a progressive move. There is an $(m-z)/m$ probability of a regressive or neutral movement.

Therefore, for all (d, L) on G that are near-complete for v_i , the probability of a progressive move is the same as in $C_2(d, L)$, and the probability of a regressive move in (d, L) is at most the probability of a regressive move in $C_2(d, L)$. Therefore, $E[\text{Info}(G, d, L)] \leq E[\text{Info}(C_2, d_2, L_2)]$. \square

Combining all of these results leads to a bound on the EE-convergence time.

Theorem 6. *For any graph G containing m bodies, and any distribution of bodies d , the expectation $E[\text{Con}(G, d)] \leq (2\pi)^{-1/2} m^{3/2} e^{m - \frac{1}{12m+1}}$.*

Proof. Our first task is to bound $\beta(G)$. Lemma 5 implies that $\beta(G) \leq \beta(C_2)$. Lemma 4 implies that $\beta(C_2) = E[\text{Info}(C_2, d, L)]$ when (d, L) is near-complete for a vertex v_1 .

We can bound $\beta(C_2)$ by considering only one specific way of clearing out v_1 . Suppose that, if v_1 is not cleared out in exactly m turns (we will refer to these m turns as a *round*), then the distribution is reset to a near-complete distribution for v_1 and a new round is started. Let $\gamma(C_2)$ be the expected number of observations required to clear v_1 under this “resetting” condition. Since the near-complete distribution has the highest expected number of observations until an informative observation, $\gamma(C_2) \geq \beta(C_2)$.

The probability of a near-complete distribution in C_2 clearing the vertex v_1 in a single round is

$$\prod_{i=0}^{m-1} \frac{m-i}{m} = \frac{m!}{m^m}. \quad (1)$$

Stirling’s approximation yields $m! \geq \sqrt{2\pi m} (m/e)^m e^{\frac{1}{12m+1}}$, which puts an upper bound on the expected number of rounds until v_1 empties out. This upper bound is

$$\frac{e^m}{\sqrt{2\pi m} e^{\frac{1}{12m+1}}}. \quad (2)$$

Since each round consists of m sensor observations, Equation 2 results in the following upper bound on $\gamma(C_2)$, which is also an upper bound on $\beta(G)$.

$$\beta(G) \leq \beta(C_2) \leq \gamma(C_2) \leq \frac{\sqrt{m} e^m}{\sqrt{2\pi} e^{\frac{1}{12m+1}}}. \quad (3)$$

Since $\beta(G)$ is the maximum expected number of stages for an informative observation, and Theorem 1 implies that there can be at most m informative observations, we get that

$$E[\text{Con}(G, d)] \leq m\beta(G) \leq \frac{m^{\frac{3}{2}} e^m}{\sqrt{2\pi} e^{\frac{1}{12m+1}}}. \quad (4)$$

□

6 Estimation of the Number of Moving Bodies

We have been assuming that the total number of moving bodies is known. Suppose that the number of bodies is not known in advance. By combining the results of Sections 5 with the Markov bound

$$\Pr[X \geq \alpha] \leq \frac{E[X]}{\alpha} \quad (5)$$

and Bayes' Theorem,

$$\Pr[A | B] = \frac{\Pr[B | A]\Pr[A]}{\Pr[B]} \quad (6)$$

we can estimate the total number of bodies in a graph, assuming that we have knowledge about the prior probability distribution over the number of bodies.

Let Bdy be a random variable denoting the number of bodies in the graph G .

Theorem 7. *Let z be the number of acquired sensor observations. Let k be the sum of the lower bounds after z observations have been acquired. Let j be an integer such that $j > k$. The following relationship holds:*

$$\Pr[\text{Bdy} = j | \text{Bdy} = k \cup (\text{Bdy} > k \cap \text{Con}(G, d) \geq z)] \leq \frac{j^{\frac{3}{2}} e^j}{z \sqrt{2\pi} e^{\frac{1}{12j+1}}} \frac{\Pr[\text{Bdy} = j]}{\Pr[\text{Bdy} = k]}. \quad (7)$$

Proof. Using Bayes' Theorem, we get that $\Pr[\text{Bdy} = j | \text{Bdy} = k \cup (\text{Bdy} > k \cap \text{Con}(G, d) \geq z)]$ is equal to

$$\frac{\Pr[\text{Bdy} = k \cup (\text{Bdy} > k \cap \text{Con}(G, d) \geq z) | \text{Bdy} = j] \Pr[\text{Bdy} = j]}{\Pr[\text{Bdy} = k \cup (\text{Bdy} > k \cap \text{Con}(G, d) \geq z)]}. \quad (8)$$

Because the condition fixes the value of Bdy , the term $\Pr[\text{Bdy} = k \cup (\text{Bdy} > k \cap \text{Con}(G, d) \geq z) | \text{Bdy} = j]$ can be simplified to $\Pr[\text{Con}(G, d) \geq z | \text{Bdy} = j]$. The Markov bound implies

$$\Pr[\text{Con}(G, d) \geq z | \text{Bdy} = j] \leq \frac{E[\text{Con}(G, d) | \text{Bdy} = j]}{z}. \quad (9)$$

Theorem 6 implies $E[\text{Con}(G, d) \mid \text{Bdy} = j] \leq \frac{j^{\frac{3}{2}} e^j}{\sqrt{2\pi} e^{\frac{1}{12j+1}}}$. Additionally, $\Pr[\text{Bdy} = k \cup (\text{Bdy} > k \cap \text{Con}(G, d) \geq z)] \geq \Pr[\text{Bdy} = k]$, as the term on the right is a strictly stronger condition. Substituting these bounds into Equation 8 yields

$$\Pr[\text{Bdy} = j \mid \text{Bdy} = k \cup (\text{Bdy} > k \cap \text{Con}(G, d) \geq z)] \leq \frac{j^{\frac{3}{2}} e^j}{z \sqrt{2\pi} e^{\frac{1}{12j+1}}} \frac{\Pr[\text{Bdy} = j]}{\Pr[\text{Bdy} = k]}. \quad (10)$$

□

Theorem 7 can also be used to give a lower bound on $\Pr[\text{Bdy} = k \mid \text{Bdy} = k \cup (\text{Bdy} > k \cap \text{Con}(G, d) \geq z)]$.

Corollary 8. *Let z be the number of acquired sensor observations. Let k be the sum of the lower bounds after z observations have been acquired. Let j be an integer such that $j > k$. The following relationship holds:*

$$\Pr[\text{Bdy} = k \mid \text{Bdy} = k \cup (\text{Bdy} > k \cap \text{Con}(G, d) \geq z)] \geq 1 - \lambda, \quad (11)$$

in which

$$\lambda = \sum_{j=k+1}^{\infty} \frac{j^{\frac{3}{2}} e^j}{z \sqrt{2\pi} e^{\frac{1}{12j+1}}} \frac{\Pr[\text{Bdy} = j]}{\Pr[\text{Bdy} = k]}. \quad (12)$$

Proof. The term

$$\sum_{j=k+1}^{\infty} \frac{j^{\frac{3}{2}} e^j}{z \sqrt{2\pi} e^{\frac{1}{12j+1}}} \frac{\Pr[\text{Bdy} = j]}{\Pr[\text{Bdy} = k]} \quad (13)$$

is an upper bound on the probability that $\text{Bdy} > k$. Since $\text{Bdy} \geq k$ is known, any remaining probability mass must belong to $\Pr[\text{Bdy} = k \mid \text{Bdy} = k \cup (\text{Bdy} > k \cap \text{Con}(G, d) \geq z)]$. □

The Markov bound converges slowly. If the variance of $\text{Con}(G, d)$ is relatively low, then we could obtain a faster estimation of the number of bodies by using the Chebyshev bound

$$\Pr[|X - E[X]|] \geq \alpha \sqrt{\text{Var}[X]} \leq \frac{1}{\alpha^2}. \quad (14)$$

Unfortunately, we have been unable to prove any bounds on $\text{Var}[\text{Con}(G, d)]$. However, simulations (in C_2) seem to indicate that $\text{Var}[\text{Con}(G, d)] \approx E[\text{Con}(G, d)]^2$. Therefore, it is unlikely that the Chebyshev bound would produce an estimate that would converge much more quickly than the Markov bound.

7 A Lower Bound for the EE-Convergence Time

Corollary 2 establishes that each vertex must empty out at least once in order to reduce to a counting information state. This implies that each body must move at least once. Under the exponential random movement model, each body has an equal probability of being the one that moved during each sensor observation. Therefore, we

can acquire a lower bound on the expected number of required sensor observations by obtaining the expected number of sensor observations required until each body has moved once. This is a restatement of the classical *coupon collector's problem*.

Theorem 9. *For a graph G containing m bodies, and any distribution of bodies d , the expectation $E[\text{Con}(G, d)] \geq mH_m$, where $H_m = \sum_{i=1}^m \frac{1}{i} = \Theta(\ln m)$.*

Proof. If j different bodies have moved so far, then there is an $(m-j)/m$ probability that the next body that moves is a body that moves for the first time. Therefore, if j different bodies have moved so far, the expected number of stages until a new body moves is $m/(m-j)$. Summing over $0 \leq j \leq m-1$ yields mH_m . \square

Theorem 10. *The bound in Theorem 9 is tight.*

Proof. To demonstrate tightness, we must construct a graph G and initial body distribution d such that $E[\text{Con}(G, d)] = mH_m$. Let G be the graph consisting of m disjoint, disconnected copies of C_2 . For the initial distribution d , place one body in each of the C_2 subgraphs. The information state converges to the counting information state after each vertex that initially contains a body empties out at least once. Each vertex that initially contains a body starts with only one body a piece, and each component contains only one body. Therefore, if each body moves at least once, then each vertex that contained a body in the initial distribution will have emptied out at least once. By Corollary 2, this is sufficient to demonstrate convergence to a counting information state. \square

8 Results for C_2

Due to C_2 's very simple structure, it is easier to analyze than general graphs. It is possible to get an exact answer for the expected number of stages until convergence for any initial distribution of C_2 . Let $\text{Out}(a, m)$ be a random variable denoting the number of stages it would take for vertex v_1 to empty out if v_1 started with a bodies, with m bodies total present in the graph (due to the symmetry in the graph, v_2 could be substituted for v_1 in the definition).

It is easy to determine the value of $\text{Out}(a, m)$ with a recurrence. For base cases, we have $E[\text{Out}(0, m)] = 0$ and $E[\text{Out}(m, m)] = 1 + E[\text{Out}(m-1, m)]$. If there are a bodies in v_1 , then there is an a/m probability that a body leaves v_1 , and an $(m-a)/m$ probability that a body enters v_1 . Therefore, when $a \neq 0$, we have

$$E[\text{Out}(a, m)] = 1 + \frac{a}{m}E[\text{Out}(a-1, m)] + \frac{m-a}{m}E[\text{Out}(a+1, m)]. \quad (15)$$

To obtain a recurrence that determines $E[\text{Con}(C_2, d)]$ for arbitrary d , we must add an additional term. Let $\text{Out}'(a, m)$ be a random variable denoting the number of stages required for *either* of the vertices to empty out, given that the vertex with fewer bodies contains a bodies, and the whole graph contains m bodies.

As a base case, note that when m is odd,

$$[\text{Out}'(\lfloor m/2 \rfloor, m)] = 1 + \lceil m/2 \rceil E[\text{Out}'(\lfloor m/2 \rfloor, m)] + \lfloor m/2 \rfloor E[\text{Out}'(\lfloor m/2 \rfloor - 1, m)]. \quad (16)$$

For even m , the equation is $E[\text{Out}'(m/2, m)] = 1 + E[\text{Out}'(m/2 - 1, m)]$. As in the previous function, $E[\text{Out}'(0, m)] = 0$. For other m , the relationship is similar to Equation 15:

$$E[\text{Out}'(a, m)] = 1 + \frac{a}{m} E[\text{Out}'(a - 1, m)] + \frac{m - a}{m} E[\text{Out}'(a + 1, m)]. \quad (17)$$

Combining these two functions leads to a solution for arbitrary distributions over C_2 .

Theorem 11. *If d is a distribution of m bodies over C_2 , with $m > 1$, that places a bodies in a single vertex, and $m - a$ in the other vertex, with $a \leq m - a$, then $E[\text{Con}(C_2, d)] = E[\text{Out}'(a, m)] + E[\text{Out}(m, m)]$.*

Proof. The term $E[\text{Out}'(a, m)]$ is the expected number of stages for one of the two vertices to empty out. When one vertex empties out, the other must be full, meaning that the expected number of remaining steps for the second vertex to empty out is $E[\text{Out}(m, m)]$. \square

The authors were unable to determine a closed form solution for the expectation of either Out or Out' . However, it is not difficult to derive an exponential lower bound on $E[\text{Out}(m, m)]$. Theorem 11 implies that this is also a lower bound on $E[\text{Con}(C_2, d)]$, regardless of the initial distribution d .

Theorem 12. *If d is a distribution of m bodies over C_2 , with $m > 1$, then*

$$E[\text{Con}(C_2, d)] \geq E[\text{Out}(m, m)] \geq \frac{2^{m+\frac{3}{2}}}{\sqrt{\pi(m-1)(m+1)} e^{\frac{9m-8}{(12m-11)(3m-3)}}} = \Omega\left(\frac{2^m}{m^{\frac{3}{2}}}\right).$$

Proof. For $0 \leq a < m$, with $m > 1$, let $k_{a,m}$ be the expected number of steps required to move from a state in which v_1 contains $a + 1$ bodies to a state in which v_1 contains a elements. Note that $E[\text{Out}(a, m)] + k_{a,m} = E[\text{Out}(a + 1, m)]$. Combining this identity with Equation 15 yields

$$E[\text{Out}(a, m)] = 1 + \frac{a}{m} E[\text{Out}(a - 1, m)] + \frac{m - a}{m} (E[\text{Out}(a, m)] + k_{a,m}). \quad (18)$$

Combining the $E[\text{Out}(a, m)]$ terms and multiplying by m/a gives

$$E[\text{Out}(a, m)] = \frac{m}{a} + E[\text{Out}(a - 1, m)] + \frac{m - a}{a} k_{a,m}. \quad (19)$$

Separating $E[\text{Out}(a, m)]$ into $E[\text{Out}(a - 1, m)] + k_{a-1,m}$ and subtracting $E[\text{Out}(a - 1, m)]$ from both sides yields

$$k_{a-1,m} = \frac{m}{a} + \frac{m-a}{a} k_{a,m}. \quad (20)$$

Since $(m-a)/a \geq 1$ when $a \leq m/2$, the value of $k_{a,m}$ grows as a becomes smaller when $a \leq m/2$. Due to the left term on the right side of Equation 20, we know that $k_{\lfloor m/2 \rfloor, m} \geq 2$. Therefore,

$$k_{0,m} \geq 2 \prod_{a=1}^{\lfloor m/2 \rfloor} \frac{m-a}{a} = 2 \frac{(m-1)!}{\lfloor \frac{m}{2} \rfloor! \lceil \frac{m}{2} \rceil!}. \quad (21)$$

We can use the version of Stirling's approximation found in [12] to bound the factorials. For even m , this becomes

$$k_{0,m} \geq \frac{2}{m} \frac{\sqrt{2\pi m} \left(\frac{m}{e}\right)^m e^{\frac{1}{12m+1}}}{\pi m \left(\frac{m}{2e}\right)^m e^{\frac{1}{3m}}} = \frac{2^{m+\frac{3}{2}}}{\sqrt{\pi m^3} e^{\frac{9m+1}{3m(12m+1)}}}. \quad (22)$$

For odd m , the equation becomes

$$k_{0,m} \geq \frac{4}{m+1} \frac{\sqrt{2\pi(m-1)} \left(\frac{m-1}{e}\right)^{m-1} e^{\frac{1}{12m-11}}}{\pi(m-1) \left(\frac{m-1}{2e}\right)^{m-1} e^{\frac{1}{3m-3}}} = \frac{2^{m+\frac{3}{2}}}{\sqrt{\pi(m-1)} (m+1) e^{\frac{9m-8}{(12m-11)(3m-3)}}}. \quad (23)$$

The two inequalities are almost the same, but the Equation 23 is smaller for $m > 1$. \square

9 Conclusion and Future Work

This paper has presented a necessary and sufficient condition for determining the distribution of a number of moving bodies in an environment made of regions separated by sensor beams. Additionally, this paper has determined bounds on the expected number of sensor observations required to determine this distribution under the exponential random movement model and shown how these bounds can be used to estimate the total number of moving bodies when this number is not known in advance.

One goal is to determine bounds on the expected number of sensor readings until convergence to a counting information state under alternate movement models. For example, if the moving bodies are people, then it is perhaps useful to consider a model where the movement of one body affects the probability of another body moving, as people often tend to enter and exit locations in groups. A model in which the paths of the moving bodies are deterministic but the transition times are random may be useful for modelling "obstacle course"-like situations and seems relatively easy to analyze.

Under the exponential random movement model, theorems 10 and 12 imply that for a single graph, one starting distribution causes convergence to the counting information state in an expected polynomial number of stages, and a different starting

distribution causes convergence in an expected exponential number of stages (note that the graph in Theorem 10 can be treated as C_2 if all the bodies start in one component). It would be useful to know under which conditions this polynomial-to-exponential “phase transition” occurs. Initial concentration of the moving bodies and the length of the shortest directed cycle are likely to be important parameters.

Exploiting the *cover time* of random walks is one technique that could be used to produce improved bounds. The cover time of a vertex $v \in V(G)$ for some graph G is the expected number of steps required for a body initially placed in v that is taking a random walk to reach every vertex in G . The cover time of the graph is the maximum cover time over all possible starting vertices. For a strongly connected graph G containing m bodies, one could make a secondary graph H with a vertex set consisting of the length m strings with characters drawn from $V(G)$. Each vertex of H is a string that contains the location of each of the m bodies in G . For $v_1, v_2 \in V(H)$, an edge exists between them if their corresponding strings of the vertices differ in exactly one character, and the differing character in v_1 has an edge in G to the differing character from v_2 . A random walk by a single body in H can represent the movements of all the bodies in G . Since G is strongly connected, so is H . Since for each vertex $v \in V(G)$, there exists a vertex in H that represents a state in which v is empty, the expected time to convergence to a counting information state in G is less than the expected cover time of H .

For a simple example, consider the case where $G = C_2$. In this case, H is the m -dimension hypercube. In order for each vertex of G to empty out, two vertices of H have to be reached. One is the vertex of H representing the state where all bodies are in $v_1 \in V(G)$, and the other is the vertex of H representing the state where all bodies are in $v_2 \in V(G)$. Since H is a regular graph with 2^m vertices, [8] implies that the expected cover time is at most $4^{m+1/2}$ steps. Therefore, $E[\text{Con}(C_2, d)] \leq 4^{m+1/2}$. This is not as good of a bound as the ones presented earlier in the paper, but further refinement of the technique may yield useful results.

Acknowledgements. This work is supported in part by NSF grant 0904501 (IIS Robotics), NSF grant 1035345 (CNS Cyberphysical Systems), DARPA STOMP grant HR0011-05-1-0008, and MURI/ONR grant N00014-09-1-1052. The authors thank Leonardo Bobadilla and Katrina Gossman for their assistance with this project.

References

1. Aslam, J., Butler, Z., Constantin, F., Crespi, V., Cybenko, G., Rus, D.: Tracking a moving object with a binary sensor network. In: 1st International Conference on Embedded Networked Sensor Systems, pp. 150–161. ACM Press (2002)
2. Baryshnikov, Y., Ghrist, R.: Target enumeration via integration over planar sensor networks. In: Proceedings of Robotics: Science and Systems IV, Zurich, Switzerland (June 2008)
3. Bobadilla, L., Sanchez, O., Czarnowski, J., Gossman, K., LaValle, S.: Controlling wild bodies using linear temporal logic. In: Proceedings of Robotics: Science and Systems (June 2011)
4. Dehn, M.: Papers on Group Theory and Topology. Springer, Berlin (1987)

5. 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)
6. Erdmann, M.A., Mason, M.T.: An exploration of sensorless manipulation. *IEEE Transactions on Robotics & Automation* 4(4), 369–379 (1988)
7. 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)
8. Feige, U.: Collecting coupons on trees, and the cover time of random walks. *Computational Complexity* 6(4), 341–356 (1996)
9. Gfeller, B., Mihalák, M., Suri, S., Vicari, E., Widmayer, P.: Counting Targets with Mobile Sensors in an Unknown Environment. In: Kutylowski, M., Cichoń, J., Kubiak, P. (eds.) *ALGOSENSORS 2007. LNCS*, vol. 4837, pp. 32–45. Springer, Heidelberg (2008)
10. Goldberg, K.Y.: Orienting polygonal parts without sensors. *Algorithmica* 10, 201–225 (1993)
11. 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 (2005)
12. Robbins, H.: A remark on stirling's formula. *The American Mathematical Monthly* 62(1), 26–29 (1955)
13. 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 (2006)
14. 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)
15. Tovar, B., Cohen, F., LaValle, S.M.: Sensor Beams, Obstacles, and Possible Paths. In: Chirikjian, G.S., Choset, H., Morales, M., Murphrey, T. (eds.) *Algorithmic Foundation of Robotics VIII. STAR*, vol. 57, pp. 317–332. Springer, Heidelberg (2009)
16. Wang, C.-C., Thorpe, C., Thrun, S., Hebert, M., Durrant-Whyte, H.: Simultaneous localization, mapping and moving object tracking. *International Journal of Robotics Research* 26(9), 889–916 (2007)
17. Yang, D.B., Gonzalez-Banos, H.H., Guibas, L.J.: Counting people in crowds with a real-time network of simple image sensors. In: Proc. IEEE International Conference on Computer Vision, vol. 1, pp. 122–129 (2003)

Convex Receding Horizon Control in Non-Gaussian Belief Space

Robert Platt

Abstract. One of the main challenges in solving partially observable control problems is planning in high-dimensional belief spaces. Essentially, it is necessary to plan in the parameter space of all relevant *probability distributions* over the state space. The literature has explored different planning technologies including trajectory optimization [8, 6] and roadmap methods [12, 4]. Unfortunately, these methods are hard to use in a receding horizon control context where it is potentially necessary to replan on every time step. Trajectory optimization is not guaranteed to find globally optimal solutions and roadmap methods can have long planning times. This paper identifies a non-trivial instance of the belief space planning problem that is convex and can therefore be solved quickly and optimally even for high dimensional problems. We prove that the resulting control strategy will ultimately reach a goal region in belief space under mild assumptions. Since the space of convex belief space planning problem is somewhat limited, we extend the approach using mixed integer programming. We propose to solve the integer part of the problem in advance so that only convex problems need be solved during receding horizon control.

1 Introduction

The problem of controlling partially observable systems is extremely important in general and is particularly important in robotics. In partially observable systems, noisy and incomplete measurements make it hard to estimate state accurately. For example, it can be very challenging for a manipulation robot to estimate the position of objects to grasp accurately using laser, visual, or tactile data. Similarly, it can be difficult for an autonomous air or land vehicle to localize itself based on noisy range bearings to features in the environment. Since state is never known exactly in these

Robert Platt

SUNY at Buffalo, Buffalo, NY 14206

e-mail: robplatt@buffalo.edu

problems, the objective of control should be to maximize the *probability* that the robot achieves its goals. These problems are frequently solved using planning and control in *belief space*, the space of probability distributions over an underlying state space. The solutions frequently involve strategies for actively gathering information in order to better localize state. This is in contrast to fully observable problems where, since state is fully observed, the only objective is to reach a goal state.

One of the challenges in controlling a partially observable system in belief space is that since belief space can be very high-dimensional, it can be difficult to compute a policy. In fact, finding optimal policies for Markov partially observable problems has been shown to be PSPACE-complete [7]. An alternative is receding horizon control where a complete plan is re-computed on each time step. However, in order for this to be effective, it is necessary to have the ability to re-plan quickly. This is because belief space dynamics can be very stochastic; a single observation can “convince” the system that it needs to adjust its belief state significantly (*i.e.* sequential Bayesian filtering can adjust its belief state estimate significantly based on only one or two observations). When this happens, the old plan becomes irrelevant and a new plan must be found. Unfortunately, most current belief space planning approaches are not fast enough to use in the inner loop of a receding horizon controller [8, 6, 12]. Moreover, most current approaches to belief space planning are based on assuming that belief space is Gaussian – that is, that belief state is always well-represented by a Gaussian [12, 15, 4, 11]. Unfortunately, this is not the case for many common robotics problems occurring in robot navigation or manipulation where multi-modal distributions are more common. In these situations, the Gaussian assumption can result in plans that are arbitrarily bad.

This paper proposes formulating the belief space planning problem as a convex program that is appropriate for receding horizon control. This is important for two reasons. First, since convex programs can be solved quickly and accurately in general, the method is fast enough for receding horizon control. Second, solutions to convex programs are guaranteed to be globally optimal. This is important because it enables us to guarantee that receding horizon control converges in the limit to a goal region in belief space as long as a path to the goal region exists. In this paper, we constrain ourselves to considering only linear systems with state-dependent observation noise. Although we are assuming a linear system, it should be noted that this problem setting is fundamentally different from the linear quadratic Gaussian (LQG) setting [2] because observation noise is state dependent. This one difference means that the certainty equivalence principle [2] does not apply. To our knowledge, this is the first belief space control strategy for systems with state dependent noise that can be guaranteed to converge to a goal region in belief space¹. After introducing the basic convex formulation, this paper introduces an approach to using mixed integer programming to solve non-convex problems comprised of a small number of convex “parts”. Although it can take significantly longer to solve a mixed integer program than a convex program, this does not affect our on-line performance since the mixed integer program may be solved ahead of time and the integer variables

¹ Unlike our methods, local optimization methods such as iLQG [14] do not make guarantees about global optimality of the resulting solutions.

fixed. We simulate our algorithm in the context of standard mobile robot localization and navigation problems where features in the environment make observations more or less noisy in different parts of the environment.

2 Problem Statement

We consider a class of Markov partially observable control problems where state, $x \in \mathbb{R}^n$, is not observed directly. Instead, on each time step, the system takes an action, $u \in \mathbb{R}^l$, and perceives an observation, $z \in \mathbb{R}^m$. The process dynamics are linear-Gaussian of the form,

$$x_{t+1} = Ax_t + Bu_t + v_t, \quad (1)$$

where A and B are arbitrary, v_t is zero-mean Gaussian noise with covariance V , and t denotes the time index. The observation dynamics are of the form,

$$z_t = x_t + q_t(x_t), \quad (2)$$

where $q_t(x_t)$ is zero-mean Gaussian noise with state-dependent covariance, $Q(x_t)$. Notice that the presence of non-uniform observation noise invalidates the certainty equivalence principle [2] and makes this problem fundamentally different than the standard linear quadratic Gaussian (LQG) problem. In order to solve this problem using the convex formulation proposed in this paper, we require $Q(x)^{-1}$ to be piecewise matrix convex in x^2 . The objective of control is to reach a radius around a goal state, x_g , with high probability. Let $b_t(x) = P(x|u_{1:t-1}z_{1:t})$ denote *belief state* at time t , a probability distribution over system state that represents the state of knowledge of the system and incorporates all prior control actions, $u_{1:t-1}$, and observations, $z_{1:t}$. Our objective is to reach a belief state, b , such that

$$\Theta(b, r, x_g) = \int_{\delta \in B_n(r)} b(\delta + x_g) \geq \omega, \quad (3)$$

where $B_n(r) = \{x \in \mathbb{R}^n, x^T x \leq r^2\}$ denotes the r -ball in \mathbb{R}^n for some $r > 0$, and ω denotes the minimum probability of reaching the goal region. In addition, we require the system to adhere to a chance constraint that bounds the probability that the system collides with an obstacle on a given time step. Let $O_1, \dots, O_q \subset \mathbb{R}^n$ be a set of q polyhedral regions of state space that describe the obstacles. The probability that the system is in collision with an obstacle at time t is constrained to be less than θ :

$$P(x_t \in \cup_{n=1}^q O_n) \leq \theta. \quad (4)$$

² Matrix convexity implies that $a^T Q(x)^{-1} a$ is a convex function for any constant vector, a . Piecewise convexity means that the function can be partitioned into multiple convex regions.

3 Belief Space Planning

3.1 Non-Gaussian Belief Space Planning

We build upon a framework for non-Gaussian belief space control introduced in our prior work [8]. On each time step, a new T -horizon plan is created. For planning purposes, belief state, $b(x)$, will be represented by a set of k support points $x^{1:k} = (x^1, \dots, x^k)$ and the corresponding un-normalized weights, $w^{1:k}$:

$$b(x) \approx \sum_{i=1}^k w^i \delta(x - x^i), \quad (5)$$

where $\delta(x)$ denotes the Dirac delta function of x . As the number of samples, k , goes to infinity, this approximation becomes exact. Plans are created using a version of Bayesian filtering known as sequential importance sampling (SIS³) [1]. The SIS update occurs in two steps: the process update and the measurement update. The process update samples the support points at the next time step from the process dynamics, $x_{t+1}^i \sim N(\cdot | Ax_t^i + Bu_t, V)$, where $N(x|\infty, \Sigma)$ denotes the normal distribution with mean, ∞ , and covariance matrix, Σ , evaluated at x . Given a new observation, z , the measurement update adjusts the weight of each point according to the observation dynamics: $w_{t+1}^i = w_t^i N(z|x_{t+1}, Q(x_{t+1}))$ (recall the unit observation dynamics of Equation 2). In this paper, we will track un-normalized weights and write the above measurement update as:

$$w_{t+1}^i = w_t^i \exp\left(-\|z - x_{t+1}\|_Q^2\right), \quad (6)$$

where $\|x\|_Q^2 = x'Q^{-1}x$ denotes the L2 norm of x weighted by Q^{-1} .

Input : initial belief state, b , goal state, x_g , planning horizon, T , and belief-state update, G .

```

1 while  $\Theta(b_t, r, x_g) < \omega$  do
2    $x^1 \leftarrow \arg \max_{x \in \mathbb{R}^n} b(x);$ 
3    $x^i \sim b(x) : b(x^i) \geq \varphi, \forall i \in [2, k];$ 
4    $u_{1:T-1} = \text{ConvexPlan}(x^{1:k}, w^{1:k}, x_g, T);$ 
5   execute action  $u_1$ , perceive observation  $z$ ;
6    $b \leftarrow G(b, u_1, z);$ 
7 end

```

Algorithm 1. Receding horizon belief space control algorithm.

³ Compared to the (more standard) SIR version of particle filtering, SIS requires an exponentially large number of samples because it omits the resampling step. This makes SIS a poor choice for tracking belief state over *long* time horizons. However, we defend the use of SIS for planning over *short* horizons (assuming that T is small relative to how quickly belief state changes) because we expect that samples drawn from the prior distribution will still represent posterior belief states relatively well.

On each time step, we plan over a discrete time horizon, T . Let $t \in [1, T]$ denote time going T steps into the future. In order to plan in belief space, it is necessary to make assumptions regarding the content of future observations. Following our prior work [8], we assume, for planning purposes only, that future observations will be generated as if the system were actually in the currently most likely sampled state. At planning time ($t = 1$), the most likely sample is

$$x_1^\zeta = \arg \max_{i \in [1, k]} x_1^i. \quad (7)$$

Denote this same state integrated forward to time t as x_t^ζ . Therefore, the predicted observation is $z_t = x_t^\zeta$. Later in the paper, we show that this assumption enables us to prove that the receding horizon control algorithm converges in the limit to a goal region in belief space.

Algorithm 1 illustrates the complete replanning algorithm. Following our prior work in [8], we use two different representations of belief state in the algorithm. For planning purposes, we use the sample-based representation of Equation 5. However, for the purpose of tracking the distribution during execution, we allow the system designer to use any approximately accurate representation of belief state. Let G denote the implementation of Bayes filtering used by the designer to track belief state. G computes the next belief state, b_{next} , given that action u is taken from belief state, b and observation z is made: $b_{\text{next}} = G(b, u, z)$. Given the prior belief state, b , Steps 2 and 3 select the support points with one support point at the maximum. Step 4 computes a belief space plan, $u_{1:T-1} = (u_1, \dots, u_{T-1})$, as described in the next section. Step 5 executes the first step in the plan. Finally, Step 6 updates the belief state tracking distribution. The outer *while* loop executes until the belief space goal is satisfied. Later in this section, we give conditions under which this is guaranteed to occur.

3.2 Convex Formulation

The question remains how to solve the planning problem. We identify a class belief space trajectory planning problems that can be expressed as convex programs and solved using fast, globally optimal methods. Upon first approaching the problem, one might identify the following problem variables: x_t^i , w_t^i , and u_t , for all $t \in [1, T]$ and $i \in [1, k]$. However, notice that the weight update in Equation 6 is a non-convex equality constraint. Since we are interested in identifying a convex version of the problem, we express the problem in terms of the log-weights, $y_t^i = \log(w_t^i)$ rather than the weights themselves. Equation 6 becomes (recall Equation 7):

$$y_{t+1}^i = y_t^i - \|x_{t+1}^\zeta - x_{t+1}^i\|_{Q(x_{t+1}^i)}^2. \quad (8)$$

The second term above appears to be bi-linear in the variables. However, because we have assumed linear process dynamics, we have:

$$x_t^\zeta - x_t^i = A^t(x_1^\zeta - x_1^i),$$

where x_1^i are the support points for the prior distribution at time t and therefore constant. As a result, the second term of Equation 8 is convex when $Q(x)^{-1}$ is matrix concave in x . However, in order to express the problem in convex form, all convex constraints must be inequality constraints, not equality constraints. Therefore, we relax the constraint to become:

$$y_{t+1}^i \geq y_t^i - \|A^{t+1}(x_1^\zeta - x_1^i)\|_{Q(x_t^i)}^2. \quad (9)$$

The objective is to reach a belief state at time T such that Equation 3 is satisfied. This is accomplished by minimizing the average log weight,

$$J(T) = \frac{1}{k} \sum_{i=1}^k y_T^i. \quad (10)$$

The problem becomes:

Problem 1

$$\begin{aligned} \text{Minimize} \quad & \frac{1}{k} \sum_{i=1}^k y_T^i + \alpha \sum_{t=1}^{T-1} u_t^T u_t \\ \text{subject to} \quad & x_{t+1}^i = Ax_t^i + Bu_t, i \in [1, k] \\ & y_{t+1}^i \geq y_t^i - \|A^{t+1}(x_1^\zeta - x_1^i)\|_{Q(x_t^i)}^2 \\ & x_1^i = x^i, w_1^i = 1, i \in [1, k] \\ & x_T^\zeta = x_{goal}. \end{aligned} \quad (11)$$

α is intended to be a small value that adds a small preference for shortest-path trajectories. It turns out that because the $\frac{1}{k} \sum_{i=1}^k y_T^i$ term is part of the objective, the relaxed inequality constraint on the log-weights (Equation 9) is always active (i.e. it is tight). As a result, there is effectively no relaxation. All solutions found to Problem 1 will satisfy the equality constraint in Equation 6. Problem 1 is a convex program when $Q(x)^{-1}$ is matrix concave. When Algorithm 1 executes the convex plan in step 5, we mean that it solves an instance of Problem 1.

3.3 Examples: Single Beacon Domains

For example, Figure 1 illustrates the solution to Problem 1 for the a variant on the “light-dark” domain introduced in [11]. The cost function in this problem

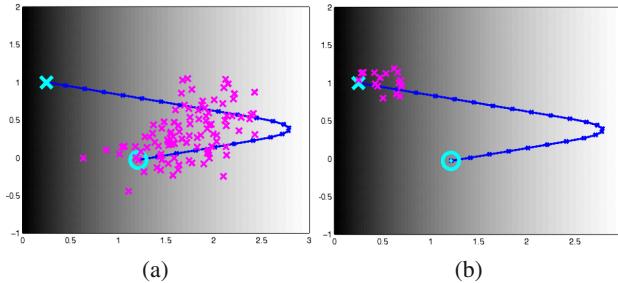


Fig. 1 Solution to the light-dark problem found using convex optimization. The magenta “x”’s denote the support points in the initial (a) and final (b) distributions that had above average weight. The distribution is represented by 1000 support points. Entropy of the distribution is clearly much lower in (b) than it is in (a).

(Equation 11) uses $\alpha = 0.4$. In this problem, the process dynamics have unit transition matrices, $A = I$ and $B = I$ in Equation 1, with Gaussian process noise with a variance of 0.01. The observation dynamics are defined in Equation 6 with the following hyperbolic state dependent noise, $Q(x) = 1/(1+2x)$, $x \geq 0$. The observation noise is illustrated by the shading in Figure 1. Noise is relatively high on the left side of state space and relatively small on the right. The system starts in an initial belief state described by a mixture of two Gaussians with means $(1.75, 0)$ and $(2, 0.5)$ and equal variances of 0.0625. The distribution is represented by a set of 1000 support points. The solid blue line in Figure 1 denotes the planned trajectory of the hypothesis sample starting at the cyan circle and ending in the cyan “x” mark. For the hyperbolic observation noise given above, Problem 1 becomes a quadratic program. We solved it (parametrized by 1000 support points) using the IBM ILOG CPLEX Optimizer version 12.3 in 0.08 seconds on a dual-core 2.53GHz Intel machine with 4G of RAM. The magenta “x” marks show the locations of the importance samples with weights greater than or equal to the average. Initially (Figure 1(a)), the sampled distribution has a high entropy. However, entropy drops significantly so that by the end of the trajectory (Figure 1(b)), entropy is much lower.

The single-beacon domain is another example of a convex belief space planning problem. In this problem, the covariance of observation noise is a function of the L2 distance from a single point in state space (*i.e.* the beacon). Figure 2 illustrates the solution to this problem. The beacon is located at $x_{\text{beacon}} = (2, 1.5)^T$. The covariance of the observation noise varies as a hyperbolic function of the L2 distance from the beacon center: $Q(x) = 1/(6 - \|x - x_{\text{beacon}}\|_2)$, $\|x - x_{\text{beacon}}\|_2 < 6$. For this noise function, Problem 1 can be expressed as a second order cone program where the constraint on the domain of x is interpreted as a second order cone constraint on the positions of all the support points. We solved this program using CPLEX in less than one second for a distribution with 100 support points.

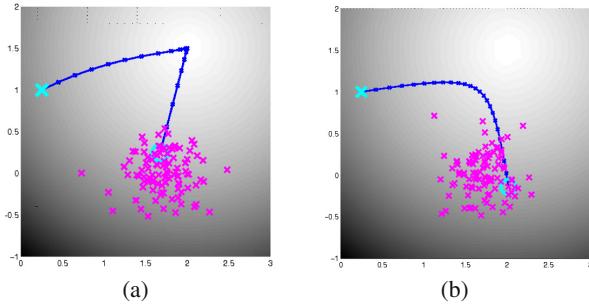


Fig. 2 Single-beacon problem solved for two different cost functions. (a) shows a solution found using a cost function (Equation 11) with $\alpha = 1$. (b) shows the solution found with a cost parameter $\alpha = 5$. The magenta “x”’s denote the support points in the initial distribution.

3.4 Analysis of Algorithm Convergence

For systems with deterministic process dynamics, it is possible to show that Algorithm 1 reaches a goal region of belief space in the limit as long as a feasible horizon- T path with cost $J < 0$ exists from any starting configuration, $x_1 \in \mathbb{R}^n$, and as long as the implementation of Bayesian filtering, G , used in the algorithm is sufficiently accurate. Let $F_{\mathbf{u}}(x) = (x_1, \dots, x_T)$ denote the sequence of T states reached after taking actions, $\mathbf{u} = (u_1, \dots, u_{T-1})$, starting from state $x_1 = x$. Let

$$q_{x,\mathbf{u}}(\mathbf{z}) = N(\mathbf{z}|F_{\mathbf{u}}(x), \mathbb{Q}(x)) \quad (12)$$

be the probability of making the observations, $\mathbf{z} = (z_1, \dots, z_T)$, given that the system starts in state x and takes actions, \mathbf{u} , where $\mathbb{Q}_{\mathbf{u}}(x) = \text{diag}(\mathbb{Q}(x_1), \dots, \mathbb{Q}(x_T))$, $(x_1, \dots, x_T) = F_{\mathbf{u}}(x)$ denotes the block diagonal matrix comprised of the covariance matrices of observation noise on each time step. Let

$$p_{b,\mathbf{u}}(\mathbf{z}) = \int_{x \in \mathbb{R}^n} b(x) q_{x,\mathbf{u}}(\mathbf{z}) \quad (13)$$

be the probability of the observations given \mathbf{u} marginalized over b . Our analysis is based on the following two lemmas that are adapted from [10] and stated here without proof.

Lemma 1. *Suppose we are given an arbitrary sequence of $T - 1$ actions, \mathbf{u} , an arbitrary initial state, $x_1 \in \mathbb{R}^n$, and an arbitrary initial distribution, b_1 . Then, the expected probability of x_T , where $(x_1, \dots, x_T) = F_{\mathbf{u}}(x)$, found by recursively evaluating the deterministic Bayes filter update $T - 1$ times is*

$$\mathbb{E}_{\mathbf{z}} \left\{ \frac{b_T(x_T)}{b_1(x_1)} \right\} \geq \exp(D_1(q_{\kappa,\mathbf{u}}, p_{b_1,\mathbf{u}}) - D_1(q_{\kappa,\mathbf{u}}, q_{x,\mathbf{u}})),$$

where κ denotes the true state at time t and D_1 denotes the KL divergence between the arguments.

It is possible to bound the KL divergence between the two distributions, $q_{\kappa, \mathbf{u}}$ and $p_{b_1, \mathbf{u}}$ using the following lemma.

Lemma 2. Suppose we are given a partially observable system with state dependent noise that satisfies Equation 2, an arbitrary sequence of T actions, \mathbf{u} , and a distribution, b_1 . Suppose $\exists \Lambda_1, \Lambda_2 \subseteq \mathbb{R}^n$ such that $\forall x_1, x_2 \in \Lambda_1 \times \Lambda_2, \|F_{\mathbf{u}}(x_1) - F_{\mathbf{u}}(x_2)\|_{\mathbb{Q}_{\mathbf{u}}(x_2)}^2 \geq \varepsilon^2$ and $\int_{x \in \Lambda_1} b_1(x) \geq \gamma, \int_{x \in \Lambda_2} b_1(x) \geq \gamma$. Then

$$\min_{y \in \mathbb{R}^n} D_1(q_{y, \mathbf{u}}, p_{b_1, \mathbf{u}}) \geq 2\eta^2 \gamma^2 \left(1 - e^{-\frac{1}{2}\varepsilon^2}\right)^2,$$

where $\eta = 1/\sqrt{(2\pi)^n |\mathbb{Q}_{\mathbf{u}}(x_2)|}$ is the Gaussian normalization constant.

In order to apply Lemma 2, it is necessary to identify regions, $\Lambda_1, \Lambda_2 \subseteq \mathbb{R}^n$ and a ε such that $\forall x_1, x_2 \in \Lambda_1 \times \Lambda_2, \|F_{\mathbf{u}}(x_1) - F_{\mathbf{u}}(x_2)\|_{\mathbb{Q}_{\mathbf{u}}(x_2)}^2 \geq \varepsilon^2$. Below, we show that these regions always exist for action sequences with a sufficiently small associated cost, $J(t+T)$ (Equation 10).

Lemma 3. Given a linear-Gaussian system with state dependent noise that satisfies Equations 1 and 2 and a set of k initial support points and weights, $x_1^i, i \in [1, k]$ and $w_1^i, i \in [1, k]$, let \mathbf{u} be a sequence of $T-1$ actions with cost $J = \frac{1}{k} \sum_{i=1}^k \log w_T^i$. Then, $\exists j \in [1, k]$ such that $\forall r \in \mathbb{R}^+, \forall \delta_1, \delta_2 \in B_n(r)$:

$$\|F_{\mathbf{u}}(x_1^i + \delta_1) - F_{\mathbf{u}}(x_1^j + \delta_2)\|_{\mathbb{Q}_{\mathbf{u}}(x_1^j)}^2 \geq \left[\sqrt{-J} - 2r\sqrt{\lambda} \right]^2,$$

where $B_n(r) = \{x \in \mathbb{R}^n; x^T x \leq r^2\}$, and λ denotes the maximum eigenvalue of $G^T \mathbb{Q}_{\mathbf{u}}(x_1^i)^{-1} G$ where $G = (I(A)^T (A^2)^T \dots (A^{T-1})^T)^T$.

Proof. A cost J implies that there is at least one support point, $j \in [1, k]$, such that $\log w_T^j \leq J$. This implies that $\|G(x_1^{\xi} - x_1^j)\|_{\mathbb{Q}_{\mathbf{u}}(x_1^j)}^2 \geq J$. Also, we have that $\forall \delta_1, \delta_2 \in B_n(r), \|G(\delta_1 + \delta_2)\|_{\mathbb{Q}_{\mathbf{u}}(x_1^j)}^2 \leq 4r^2\lambda$. Applying Lemma 4 in [9] gives us the conclusion of the Lemma.

In order to show convergence of Algorithm 1, we show that the likelihood of a region about the true state increases by a finite amount on each iteration of the outer *while* loop. Our proof only holds for deterministic systems (however, experimental evidence suggests that the approach is applicable to stochastic systems).

Theorem 1. Suppose we have:

1. a system described by Equations 1 and 2 where the process dynamics are required to be deterministic;
2. a prior distribution, b_1 ;
3. a trajectory, $\mathbf{u}_{\tau-1}$, defined over $k \geq 2$ support points with cost, $J_{\tau-1}$.

If an exact implementation of Bayesian filtering were to track state while executing $\mathbf{u}_{1:\tau-1}$ resulting in a distribution at time τ of b_τ , then the probability of all states within a ball of radius r about the true state is expected to increase by a factor of

$$\exp \left[2\eta^2 \gamma^2 \left(1 - e^{-\frac{1}{2}(\sqrt{-\log J} - 2r\sqrt{\lambda})^2} \right) \right] \quad (14)$$

relative to its value at time 1, where $\eta = 1/\sqrt{(2\pi)^n |\mathbb{Q}_\mathbf{u}(x_1^1)|}$ is the Gaussian normalization constant, $\gamma = \varepsilon \text{Vol}_n(r)$, $\text{Vol}_n(r)$ is the volume of the r -ball in n dimensions, and λ is the maximum eigenvalue of $G^T \mathbb{Q}_\mathbf{u}(x_1^1)^{-1} G$ where $G = (I(A)^T (A^2)^T \dots (A^{T-1})^T)^T$.

Proof. Lemma 3 gives us two samples, x^i and x^1 such that $\|F_\mathbf{u}(x_1^1 + \delta_1) - F_\mathbf{u}(x_1^i + \delta_2)\|_{\mathbb{Q}_\mathbf{u}(x_1^j)}^2$ is lower-bounded. Lemma 2 gives us a lower bound of $D_1(q_y, p)$ by setting $\varepsilon = \sqrt{-\log J} - 2r\sqrt{\lambda}$. Lemma 1 gives us the conclusion of the Theorem by noting that $D_1(q_\kappa, q_x) = 0$ when $x = \kappa$.

Since Theorem 1 shows that the probability of a region about the true state increases on each iteration of the *while* loop of Algorithm 1, we know that the algorithm eventually terminates in the belief space goal region, as stated in the following Theorem.

Theorem 2. Suppose we have the system described by Equations 1 and 2 where the process dynamics are deterministic. Suppose that $\forall x, x_g \in \mathbf{R}^n$, there always exists a solution to Problem 1 from x to x_g such that $J_T < 0$. Then Algorithm 1 reaches the goal region of belief space and terminates with probability 1 when a sufficiently accurate implementation of Bayesian filtering is used to track belief state.

Proof. Since the convex planner is globally optimal, we know that it will find a feasible plan with $J_t < 0$ if one exists. According to Theorem 1, this implies that the likelihood of an r -ball for some $r \in \mathbb{R}^+$ about the true state is expected to increase by a finite amount on each iteration of the *while* loop. Eventually, the goal region must contain at least ω probability mass and the algorithm terminates in the goal region of belief space.

4 Non-convex Observation Noise

The fact that some belief space planning problems can be solved efficiently as a convex program is important because it suggests that it might also be possible to solve efficiently non-convex problems comprised of a small number of convex parts. One way to accomplish this is to use mixed integer programming. Although integer programming (IP) is an NP-complete problem, there are branch-and-bound methods available that can be used to efficiently solve problems with “small” numbers of integer variables with good average-case running times. Moreover, since the complexity of the IP is insensitive to the dimensionality of the underlying state space or the belief space representation, this method can be used to solve high dimensional planning problems that cannot be solved in other ways.

4.1 Mixed Integer Formulation

Suppose that state space contains q non-overlapping convex polyhedral regions, $R_1 \cup \dots \cup R_q \subset \mathbb{R}^n$ such that $Q(x)^{-1}$ is matrix concave in each region. In particular, suppose that,

$$Q(x) = \begin{cases} Q_1(x) & \text{if } x \in R_1 \\ \vdots \\ Q_q(x) & \text{if } x \in R_q \\ Q_b(x) & \text{otherwise} \end{cases}$$

where $Q_j^{-1}, j \in [1, k]$ is the convex noise function in region R_j and Q_b is the “background” noise function. For this noise function, Problem 1 is non-convex. To solve the problem using mixed integer programming, we define $2 \times q \times (T - 1)$ binary variables: γ_t^j and $\lambda_t^j, \forall j \in [1, q], t \in [1, T - 1]$. These binary variables can be regarded as bit-vectors, $\gamma_{1:T-1}^j$ and $\lambda_{1:T-1}^j$, that denote when the hypothesis state enters and leaves region j . Each bit-vector is constrained to be all zero, all one, or to change from zero to one exactly once. This representation of the transition time from one region to the next uses extra bits but will make it easier to express the constraints in the following. The time step on which $\gamma_{1:T-1}^j$ transitions from zero to one denotes the time when the hypothesis state enters region j . Similarly, the time step on which $\lambda_{1:T-1}^j$ transitions from zero to one denotes the time when the hypothesis state leaves region j . These constraints on the form of the bit-vectors are expressed: $\gamma_{t+1}^j \geq \gamma_t^j, \forall j \in [1, q], t \in [1, T - 1]$ and $\lambda_{t+1}^j \geq \lambda_t^j, \forall j \in [1, q], t \in [1, T - 1]$. We also constrain the system to enter and leave each region exactly once and that the times when the hypothesis state is within different regions must be non-overlapping. This constraint is: $\lambda_{1:T-1}^j \leq \gamma_{1:T-1}^j, \forall i \in [1, q]$. The constraint that the regions are non-overlapping in time (the hypothesis state may not be in two regions at once) is: $\sum_{j=1}^q (\gamma_{1:T-1}^j - \lambda_{1:T-1}^j) \leq 1$.

Now that the constraints on the bit-vectors themselves have been established, we need to make explicit the relationship between the bit-vectors and the distribution, $b_t(x)$, encoded by the sample points and weights. First, we need to constrain the hypothesis state to be within the region R_j when $\gamma_t^j - \lambda_t^j = 1$. Suppose that each region, R_j , is defined by a set of ∞_j hyperplanes, $r_j^1, \dots, r_j^{\infty_j}$, such that $x \in r_j$ iff $(r_j^m)^T x \leq b_j, \forall m \in [1, \infty_j]$. When $\gamma_t^j - \lambda_t^j = 1$, we enforce at $x_t^j \in R_j$ using the so-called “big-M” approach [13]:

$$\forall m \in [1, \infty_j], \quad (r_j^m)^T x_t^j \leq b_j + M(1 - (\gamma_t^j - \lambda_t^j)), \quad (15)$$

where M is defined to be a scalar large enough to effectively relax the constraint. Also, when the hypothesis state is in a given region, we need to apply the corresponding noise constraint. That is, when $\gamma_{1:T-1}^j - \lambda_{1:T-1}^j = 1$, we need to apply a

constraint of the form of Equation 9. This is also accomplished using the big-M approach:

$$y_{t+1}^i \geq y_t^i - \|A^t(x_1^\zeta - x_1^i)\|_{Q_j(x_{t+1}^i)} - M(1 - (\gamma_t^j - \lambda_t^j)). \quad (16)$$

When the hypothesis state is outside of all regions, then the background noise model is applied:

$$y_{t+1}^i \geq y_t^i - \|A^t(x_1^\zeta - x_1^i)\|_{Q_b(x_{t+1}^i)} - M\left(\sum_{j=1}^q (\gamma_t^j - \lambda_t^j)\right). \quad (17)$$

4.2 Example: Multi-beacon Domain

We have explored the mixed-integer approach in the context of a multi-beacon localization problem. This problem is essentially a combination of three convex single-beacon problems and is illustrated in Figure 2. The scenario is as follows: a maneuverable aerial vehicle (such as a mini quad-rotor) is flying through the environment. The objective is to reach a neighborhood around a designated position with high confidence. Localization information is provided by beacons scattered throughout the environment. At each time step, the beacons report an estimate of the vehicle's position. The covariance of these estimates is state dependent: noise is small near the beacons but large further away. The covariance noise function is similar to that used in the single beacon domain, but it uses an L1 norm instead of an L2 norm. If the L1 distance between the system and any beacon, $x_{beacon}^j, j \in [1, q]$, is less than one ($\|x - x_{beacon}^j\|_1 < 1$), then the covariance of observation noise is: $Q_j(x) = 1/(1 + \rho - \|x - x_{beacon}^j\|_1)$, where $\rho = 0.01$. When $\|x - x_{beacon}^j\|_1 > 1, \forall j \in [1, q]$, observation noise is constant and equal to $1/\rho$.

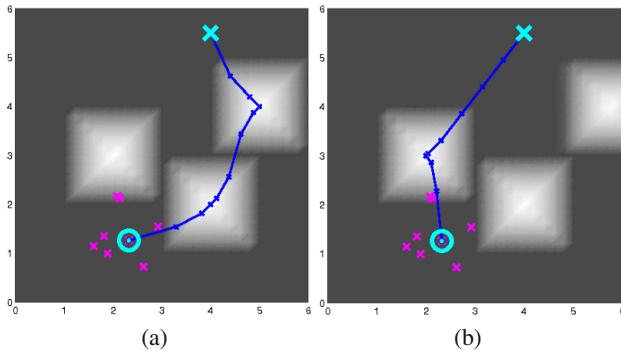


Fig. 3 Multi-beacon problem. Three beacons are located at the centers of each of the white regions. Observation noise is proportional to the shading in the figure: lower about the beacons and higher further away. (a) and (b) illustrate the trajectories found by the planner in two scenarios where the beacons are placed in slightly different configurations.

The sampled distribution was comprised of 8 samples. The prior distribution was sampled from a Gaussian with a standard deviation of 0.25. The trajectory was 24 time steps long. The cost function in this experiment used $\alpha = 0.25$. The experiment used a variation of the bit-vector strategy described above where each bit was associated with four adjacent time steps. As a result, each obstacle was associated with two bit-vectors that were each 6 bits long. In total, there were 36 binary variables in the problem. The resulting problem was a mixed integer quadratic program (MIQP). As in the light-dark experiment, we solved it using IBM CPLEX on a dual-core 2.53GHz Intel machine with 4G of RAM. It took roughly 30 second to solve each of the MIQPs in this section.

The results are illustrated in Figure 3. The shading in the figure denotes the observation noise about each beacon. We performed the experiment for two different contingencies where the beacons were arranged slightly differently. In the first, Figure 3(a), the upper right beacon was positioned closer to the goal whereas in the second, Figure 3(b), the upper right beacon was positioned further away. Notice the effect on the resulting solution. The position of this second beacon influences the direction of the entire trajectory and illustrates that the MIQP is finding globally optimal solutions to Problem 1.

5 Chance Constraints

Chance constraints are an important part of belief space planning. It is assumed that obstacles are present in the state space that are to be avoided. The probability that the planned trajectory will cause the system to collide with an obstacle on any given time step is constrained to be less than a desired threshold. One approach to finding feasible trajectories, introduced by [13], is to use mixed integer linear programming in the context of polyhedral obstacles. However, the large number of integer variables needed results in a large integer program that can be difficult to solve. A key innovation, known as iterative deepening [5], improves the situation dramatically by adding constraints and integer variables as needed rather than all at once. This helps because most constraints in obstacle avoidance problems are inactive – they can be removed without affecting the solution. Blackmore has used this idea to solve chance constrained problems where belief state is represented by unweighted support points and where there are no observations [3]. The key innovation of this section is to extend this approach to avoiding obstacles to partially observable problems where sensing is active.

Our problem is to find a plan such that on every time step, the probability that the system has collided with an obstacle is less than a threshold, θ . If there are q polyhedral obstacles, O_1, \dots, O_q , then this condition is (Equation 4):

$$P\left(x \in \bigcup_{j=1}^q O_j\right) \leq \theta.$$

Suppose that each obstacle, $j \in [1, q]$, is defined by v_j hyperplanes where the m^{th} hyperplane has a normal o_j^m and an intercept c_j^m . The condition that support point

i be outside the m hyperplane in obstacle j is $(o_j^m)^T x_t^i \leq c_j^m$. The condition that the normalized weight of support point i be less than θ/k is:

$$\frac{\exp(y_t^i)}{\sum_{n=1}^k \exp(y_t^n)} \leq \frac{\theta}{k}. \quad (18)$$

If the weight of each support point that is inside an obstacle is less than θ/k , then we are guaranteed that the total weight of the contained support points is less than θ . Unfortunately, since Equation 18 turns out to be a concave constraint, we cannot directly use it. Instead, we note that a simple sufficient condition can be found using Jensen's inequality:

$$y_t^i \leq \log(\theta) + \frac{1}{k} \sum_{n=1}^k y_t^n. \quad (19)$$

At this point, we are in a position to enumerate the set of constraints for all possible time steps, support points, obstacles, and obstacle hyperplanes. However, since each of these constraints is associated with a single binary variable, it is infeasible to solve the corresponding integer program for any problem of reasonable size. Instead, we use iterative deepening to add only a small set of active constraints.

Iterative deepening works by maintaining a set of active constraints. Initially, the active set is null. On each iteration of iterative deepening, the optimization problem is solved using the current active set. The solution is checked for particles that intersect an obstacle with a weight of at least θ/k . Violations of the obstacle chance constraint are placed in a temporary list that contains the time step and the sample number of the violation. The violations are ranked in order of the amount by which they penetrate the obstacle. Constraints (and the corresponding binary variables) are added corresponding to the most significant violation. If sample i collided with obstacle j at time t on a particular iteration of iterative deepening, we create $v_j + 1$ new binary variables, $\psi_t^{i,j,m}$, $m \in [1, v_j + 1]$. Essentially, these $v_j + 1$ binary variables enable us to formulate the following OR condition: either the sample is feasible for some hyperplane, $m \in [1, v_j]$ (indicating the point is outside of the obstacle), or Equation 19 is true (indicating a below-threshold weight). Using the “big-M” approach, we add the following constraints:

$$\sum_{m=1}^{v_j+1} \psi_t^{i,j,m} \geq 1, \quad (20)$$

$$(o_j^m)^T x_t^i \leq c_j^m + M(1 - \psi_t^{i,j,m}), m \in [1, v_j], \quad (21)$$

and

$$y_t^i \leq \log(\theta) + \frac{1}{k} \sum_{n=1}^k y_t^n + M(1 - \psi_t^{i,j,v_j+1}). \quad (22)$$

The constraint in Equation 20 forces at least one of the $v_j + 1$ binary variables to be one. This forces at least one of the constraints in Equations 21 and 22 to *not* be relaxed. Essentially, the support point cannot be inside *all* faces of the obstacle

while also having a large weight. This process of adding constraints via iterative deepening continues until no more violations are found. In principle, this procedure could become intractable if it is necessary to add all possible constraints in order to bring the violations list to null. In practice, however, this rarely happens.

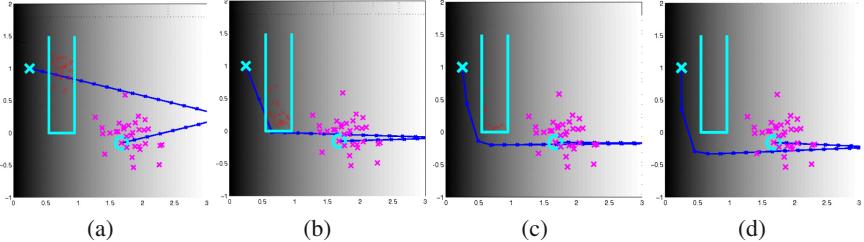


Fig. 4 Chance-constrained trajectories found on iterations 1 (a), 3 (b), 9 (c), and 25 (d) of iterative deepening. The obstacle is shown in cyan. The magenta “x”’s show the initial distribution. The red “x”’s in collision with the obstacle show the list of chance constraint violations at that particular iteration.

Figure 4 illustrates our approach to chance constraints in the presence of a single obstacle. With the exception of the obstacle, the domain is nearly identical to the light-dark domain in Section 3.3. The distribution is encoded using 32 weighted samples. There are 24 time steps. The cost function and the state-dependent noise are the same as those used in Section 3.3. The chance constraint threshold is $\theta = 0.1$ – the probability that the system may be in collision with the obstacle cannot exceed 0.1 on a single time step. (The effective chance constraint may be tighter because of the sufficient condition found using Jensen’s inequality.) The sequence of sub-figures in Figure 4 roughly illustrates progress on different iterations of iterative deepening. Even though this problem requires $32 \times 24 \times 3 = 2304$ binary variables in the worst case, we have found an optimal solution after adding only $25 \times 3 = 75$ binary variables.

6 Conclusions

In this paper, we express the problem of planning in partially observable systems as a convex program. Since convex programs can be solved quickly (on the order of milliseconds) and with guaranteed convergence to global optima, this makes the approach suitable for receding horizon control. In consequence, we are able to identify a new class of partially observable problems that can be solved efficiently and with provable correctness guarantees. We extend our approach to non-convex problems using mixed-integer programming.

References

1. Arulampalam, S., Maskell, S., Maskell, S., Gordon, N., Clapp, T.: A tutorial on particle filters for on-line non-linear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing* 50, 174–188 (2001)
2. Bertsekas, D.: *Dynamic Programming and Optimal Control*, 3rd edn. Athena Scientific (2007)
3. Blackmore, L., Ono, M., Bektassov, A., Williams, B.: A probabilistic particle-control approximation of chance-constrained stochastic predictive control. *IEEE Transactions on Robotics* (June 2010)
4. Bry, A., Roy, N.: Rapidly-exploring random belief trees for motion planning under uncertainty. In: *IEEE Int'l Conf. on Robotics and Automation* (2011)
5. Earl, M., D'Andrea, R.: Iterative milp methods for vehicle control problems. In: *IEEE Conf. on Decision and Control*, pp. 4369–4374 (December 2004)
6. Erez, T., Smart, W.: A scalable method for solving high-dimensional continuous POMDPs using local approximation. In: *Proceedings of the International Conference on Uncertainty in Artificial Intelligence* (2010)
7. Papadimitriou, C., Tsitsiklis, J.: The complexity of Markov decision processes. *Mathematics of Operations Research* 12(3), 441–450 (1987)
8. Platt, R., Kaelbling, L., Lozano-Perez, T., Tedrake, R.: Efficient planning in non-gaussian belief spaces and its application to robot grasping. In: *Int'l Symposium on Robotics Research* (2011)
9. Platt, R., Kaelbling, L., Lozano-Perez, T., Tedrake, R.: A hypothesis-based algorithm for planning and control in non-gaussian belief spaces. Technical Report CSAIL-TR-2011-039, Massachusetts Institute of Technology (2011)
10. Platt, R., Kaelbling, L., Lozano-Perez, T., Tedrake, R.: Non-gaussian belief space planning: Correctness and complexity. In: *IEEE Int'l Conference on Robotics and Automation* (2012)
11. Platt, R., Tedrake, R., Kaelbling, L., Lozano-Perez, T.: Belief space planning assuming maximum likelihood observations. In: *Proceedings of Robotics: Science and Systems, RSS* (2010)
12. Prentice, S., Roy, N.: The belief roadmap: Efficient planning in linear POMDPs by factoring the covariance. In: *12th International Symposium of Robotics Research* (2008)
13. Richards, A., How, J.: Aircraft trajectory planning with collision avoidance using mixed integer linear programming. In: *American Controls Conference* (2002)
14. Todorov, E., Li, W.: A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems. In: *IEEE Conference on Decision and Control* (2005)
15. van den Berg, J., Patil, S., Alterovitz, R.: Motion planning under uncertainty using differential dynamic programming in belief space. In: *Int'l Symposium on Robotics Research* (2011)

Planar Uncertainty Propagation and a Probabilistic Algorithm for Interception

Andrew W. Long, Kevin C. Wolfe, and Gregory S. Chirikjian

Abstract. Many robotics applications involve motion planning with uncertainty. In this paper, we focus on path planning for planar systems by optimizing the probability of successfully arriving at a goal. We approach this problem with a modified version of the Path-of-Probability (POP) algorithm. We extend the POP algorithm to allow for a moving target and to optimize the number of steps to reach the goal. One tool that we develop in this paper to increase efficiency of the POP algorithm is a second order closed-form uncertainty propagation formula. This formula is utilized to quickly propagate the mean and covariance of nonparametrized distributions for planar systems. The modified POP algorithm is demonstrated on a simple rolling disc example with a moving goal.

1 Introduction

A fundamental problem in robotics is the question *what is the best way to get from here to there?* This problem is known as *motion planning*, in which one tries to optimize the path for a robot to arrive at a goal. Motion planning is important for many applications such as autonomous cars, unmanned aerial vehicles, manufacturing robots, domestic robots and robotic surgery manipulators. If we had perfect knowledge of the robot and its environment, we could simply integrate the velocity commands from a known starting point to obtain an exact position of the robot at some future time. However, uncertainty in actuation, models and the environment will inevitably produce errors between the actual position and this integration making it difficult to follow a deterministic path. As a result, a more pertinent question is *what path has the greatest probability of reaching the goal?* A strategy that we

Andrew W. Long · Kevin C. Wolfe · Gregory S. Chirikjian
Johns Hopkins University,
Baltimore, MD
e-mail: {awlong, kevin.wolfe, gregc}@jhu.edu

employ in this paper is to represent all possible poses (position and orientation) with probability density functions (pdfs). The key then is to maintain and update these pdfs as the robot moves along a path. With this strategy, we can then begin to choose paths with higher probability of reaching a desired goal pose.

The main focus of this paper is with regards to motion planning for planar systems with uncertainty. Generally, planners try to minimize the execution time or minimize the final covariance. In this paper, we are concerned with maximizing the probability of successfully arriving at a goal. This objective has been considered by [1] and [2] with the stochastic motion road map (SRM). In our approach, we utilize a modified version of the Path-of-Probability (POP) algorithm to maximize this probability of success. In previous POP algorithms ([10], [18]), there was an assumption that the goal pose was fixed in time and that the planner knew how many discrete steps should be taken to reach the goal. We modify this algorithm to allow for a moving target and to optimize the number of steps.

Our algorithm relies on the ability to efficiently propagate uncertainty along a path. A tool that we develop in this paper is a set of closed-form recursive formulas for planar systems that can be utilized to quickly propagate the mean and covariance of nonparameterized distributions. Our formulas are nonparametric since they do not assume a specific distribution. The formulas are different than those of the Kalman methods because we use the theory of Lie algebras and Lie groups and propagate the error to second order.

The remainder of this section consists of related work and an outline of the paper.

1.1 Related Work

In the last two decades, there has been significant research into motion planning with uncertainty. Motion planning in general has been studied extensively as demonstrated in [14] and [9]. Several techniques that specifically incorporate uncertainty include back propagation [15], Linear Quadratic Gaussian (LQG) motion planning [22], stochastic dynamic programming [4], [3] and partially observable Markov decision processes (POMDPs) [21]. A bottleneck of these approaches is the computation time, especially as the length of the path increases. Another approach uses an extended space consisting of poses and covariances, allowing utilization of standard search techniques such as A^* or sampling-methods [6], [13], [19]. In the area of sampling-based methods, the well-known rapidly exploring random trees (RRTs) has been extended to the belief space in a search algorithm known as RRBT [5].

One problem that must be considered in these algorithms is how to propagate the uncertainty. Small errors can be propagated for linear systems with the Kalman filter [12] or for nonlinear systems with Jacobian-based methods such as in the classic work of [20]. For larger errors with unknown distributions, the standard technique is the particle filter [21], which consists of randomly selecting a large number of particles from the distribution and propagating these particles in time. The robot's

true position has a higher probability of occurring in the denser areas. However, the particle filter may require a large number of samples and can become computationally expensive. Recently, Wang and Chirikjian derived second-order propagation formulas for the Euclidean motion group $SE(3)$ [23] which can be used to efficiently propagate uncertainty without sampling. The propagation work presented in this paper is similar to their work but is focused on the planar motion group $SE(2)$.

The POP algorithm first appeared for manipulator arms to solve inverse kinematics problems by representing the reachable workspace with probability densities [10]. The idea of using reachable-state density functions was then applied to trajectory planning problems using the theory of Lie groups in [16]. A problem with the approach in [16] is that it is computationally expensive and is difficult to implement in real-time. With the closed-form propagation formulas for $SE(3)$ from [23], the POP algorithm was implemented more quickly and was applied to needle-steering in [18]. In this paper, we take a similar approach as [18] but remove the requirement of a fixed goal pose and a known number of steps to reach the goal.

1.2 Outline

In Sec. 2, we review important concepts and definitions from rigid-body motions, probability and statistics. We derive the second-order error propagation formulas for the planar motion group in Sec. 3. We describe the modified POP algorithm in Sec. 4. In Sec. 5, we demonstrate the propagation formulas and the POP algorithm on a simple rolling disc example. Our conclusions are discussed in Sec. 6.

2 Terminology and Notation

2.1 Rigid-Body Motions

The elements of the planar special Euclidean group, $SE(2)$, are the semidirect product of the plane, \mathbb{R}^2 with the special orthogonal group, $SO(2)$. $G = SE(2)$ is an example of a *matrix Lie group* where G is a set of square 3×3 square matrices and the group operation \circ is matrix multiplication. For a full review of Lie groups see [17] and [7]. The elements of $SE(2)$ and their inverses are given respectively as

$$g = \begin{pmatrix} R & \mathbf{t} \\ \mathbf{0}^T & 1 \end{pmatrix} \quad \text{and} \quad g^{-1} = \begin{pmatrix} R^T & -R^T \mathbf{t} \\ \mathbf{0}^T & 1 \end{pmatrix} \quad (1)$$

where R is the rotational part, \mathbf{t} is the translational part and $\mathbf{0}$ is a zero-vector.

In this paper, we also make use of $se(2)$, the Lie algebra associated with $SE(2)$. For a vector $x = [v_1, v_2, \alpha]^T$, we define the \wedge and \vee operators for $se(2)$ by

$$\hat{\mathbf{x}} = X = \begin{pmatrix} 0 & -\alpha & v_1 \\ \alpha & 0 & v_2 \\ 0 & 0 & 0 \end{pmatrix} \quad \text{and} \quad X^\vee = \mathbf{x}, \quad (2)$$

to allow us to map from \mathbb{R}^3 to $se(2)$ and back. We can obtain the group elements of $SE(2)$ by applying the matrix exponential $\exp(\cdot)$ to elements of the Lie algebra to give the following [7]

$$g(\mathbf{x}) = \exp(X) = \begin{pmatrix} \cos \alpha & -\sin \alpha & t_1 \\ \sin \alpha & \cos \alpha & t_2 \\ 0 & 0 & 1 \end{pmatrix}, \quad \text{where} \quad (3)$$

$$\mathbf{t} = \begin{pmatrix} t_1 \\ t_2 \end{pmatrix} = \frac{1}{\alpha} \begin{pmatrix} \sin \alpha & -(1 - \cos \alpha) \\ (1 - \cos \alpha) & \sin \alpha \end{pmatrix} \mathbf{v}, \quad (4)$$

and where $\mathbf{v} = [v_1, v_2]^T$. Since we use the matrix exponential to obtain the elements of $SE(2)$, we refer to the vector \mathbf{x} as exponential coordinates. With the matrix logarithm $\log(\cdot)$, we can obtain the vector \mathbf{x} from a group element $g \in SE(2)$ with

$$\mathbf{x} = (\log(g))^\vee. \quad (5)$$

We define two adjoint operators $Ad(g)$ and $ad(X)$ to satisfy

$$Ad(g)\mathbf{x} = (\log(g \circ \exp(X) \circ g^{-1}))^\vee, \quad \text{and} \quad ad(X)\mathbf{y} = ([X, Y])^\vee \quad (6)$$

where $g \in SE(2)$, $(X, Y) \in se(2)$ and $[X, Y] = XY - YX$ is the Lie bracket. These two adjoint operators are related by

$$Ad(\exp(X)) = \exp(ad(X)). \quad (7)$$

The adjoint matrices for $SE(2)$ and $se(2)$ are given explicitly as [7]

$$Ad(g) = \begin{pmatrix} R & M\mathbf{t} \\ \mathbf{0}^T & 1 \end{pmatrix}, \quad \text{and} \quad ad(X) = \begin{pmatrix} -\alpha M & M\mathbf{v} \\ \mathbf{0}^T & 0 \end{pmatrix} \quad \text{where} \quad M = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}. \quad (8)$$

The special Euclidean group $SE(2)$ is a connected unimodular matrix Lie group [17]. As a result, for any function $f : G \rightarrow \mathbb{R}$ for which the integral $\int_G f(g)dg$ exists the following properties hold for any fixed $h \in G$

$$\int_G f(g)dg = \int_G f(h \circ g)dg = \int_G f(g \circ h)dg = \int_G f(g^{-1})dg. \quad (9)$$

For a discussion of integration on groups see [7]. These properties of unimodular matrix Lie groups will be used extensively for changing coordinates in this paper.

2.2 Means, Covariances and Gaussians

A probability density function (pdf) for a Lie group (G, \circ) can be defined by

$$f(\mathbf{g}) \geq 0 \quad \forall \mathbf{g} \in G \quad \text{and} \quad \int_G f(\mathbf{g}) d\mathbf{g} = 1.$$

The definitions of the mean and covariance can be naturally extended to matrix Lie groups as in [23]. Given a group G , the mean $\infty \in G$ of a pdf $f(g)$ satisfies

$$\int_G [\log^\vee(\infty^{-1} \circ g)] f(g) dg = \mathbf{0}. \quad (10)$$

The covariance about the mean is defined as

$$\Sigma = \int_G \log^\vee(\infty^{-1} \circ g) [\log^\vee(\infty^{-1} \circ g)]^T f(g) dg. \quad (11)$$

A Gaussian in these coordinates is then given by

$$f(g; \infty, \Sigma) = \frac{1}{c(\Sigma)} \exp\left(-\frac{1}{2} [\log^\vee(\infty^{-1} \circ g)]^T \Sigma^{-1} [\log^\vee(\infty^{-1} \circ g)]\right) \quad (12)$$

where $c(\Sigma)$ is a normalizing factor. Although there is no closed-form formula for $c(\Sigma)$ in this Gaussian, when $\|\Sigma\|$ is small

$$c(\Sigma) \approx (2\pi)^{n/2} |\det \Sigma|^{\frac{1}{2}}. \quad (13)$$

We use this normalizing factor to allow us to use a closed-form formula for (12).

3 Propagation of Mean and Covariance of Pdfs on SE(2)

In this section, we are interested in developing closed-form formulas that can be used to propagate the mean and covariance of pdfs on the special Euclidean group $SE(2)$. Imagine that a robot has the option of performing Action 1 which has a pdf $f_1(g; \infty_1, \Sigma_1)$ or Action 2 which has a pdf $f_2(g; \infty_2, \Sigma_2)$. Given only the means (∞_1, ∞_2) and the covariances (Σ_1, Σ_2) of these two pdfs, we seek the mean and covariance of the resulting distribution of performing Action 1 then Action 2. This new distribution can be represented with the convolution of the two pdfs.

The convolution of two pdfs on groups is defined as [8]

$$(f_1 * f_2)(g) = \int_G f_1(h) f_2(h^{-1} \circ g) dh. \quad (14)$$

Let $\rho_i(g)$ be a unimodal pdf with mean at the identity. Then $\rho_i(\infty_i^{-1} \circ g)$ is a distribution with the same shape centered at ∞_i . We can then rewrite the definition of the convolution in (14) as

$$(f_1 * f_2)(g) = \int_G \rho_1(\infty_1^{-1} \circ h) \rho_2(\infty_2^{-1} \circ h^{-1} \circ g) dh. \quad (15)$$

If we make the change of coordinates $h \rightarrow \infty_1 \circ \infty_2 \circ k \circ \infty_2^{-1}$, then

$$(f_1 * f_2)(g) = \int_G \rho_1^{\infty_2}(k) \rho_2(k^{-1} \circ \infty_2^{-1} \circ \infty_1^{-1} \circ g) dk \quad (16)$$

$$= (\rho_1^{\infty_2} * \rho_2)(\infty_2^{-1} \circ \infty_1^{-1} \circ g), \quad (17)$$

where $\rho_1^{\infty_2}(g) = \rho_1(\infty_2 \circ g \circ \infty_2^{-1})$ is a transformed version of $\rho_1(g)$ with its argument conjugated by ∞_2 . It can be shown that this pdf $\rho_1^{\infty_2}(g)$ has its mean at the identity and covariance given by

$$\Sigma_{\rho_1^{\infty_2}} = Ad(\infty_2)^{-1} \Sigma_{\rho_1} Ad(\infty_2)^{-T}. \quad (18)$$

In [23], ‘order’ is defined to be the number of terms that were kept in the BCH expansion (see Appendix). We seek the second-order error propagation formulas using this same definition. No assumptions are made here about $f_i(g)$ except that

$$||\Sigma_i|| = O(v) \quad \text{where } v \ll 1 \text{ and that } ||\log \infty_i|| = O(1). \quad (19)$$

The condition in (19) means that the density is “highly focused.”

Theorem 1. *If $f_i(g)$ is a pdf on $SE(2)$ that has mean ∞_i and covariance Σ_i about the mean for $i = 1, 2$, then to second order, the mean and covariance of $(f_1 * f_2)(g)$ are*

$$\infty_{1*2} = \infty_1 \circ \infty_2 \quad \text{and} \quad (20)$$

$$\Sigma_{1*2} = A + B + F(A, B) \quad \text{where} \quad (21)$$

$$F(A, B) = \frac{1}{4} C(A, B) + \frac{1}{12} [A''B + (A''B)^T + B''A + (B''A)^T],$$

$$A = Ad(\infty_2^{-1}) \Sigma_1 Ad^T(\infty_2^{-1}), \quad \text{and} \quad B = \Sigma_2,$$

where the double-prime operator “” takes a matrix A and rearranges it to obtain

$$A'' = \begin{pmatrix} -a_{33} & 0 & a_{31} \\ 0 & -a_{33} & a_{32} \\ 0 & 0 & 0 \end{pmatrix}$$

and $C(A, B)$ is computed from

$$C(A, B) = \begin{pmatrix} c_{11} & c_{12} & 0 \\ c_{12} & c_{22} & 0 \\ 0 & 0 & 0 \end{pmatrix}, \quad \text{where} \quad \begin{aligned} c_{11} &= b_{33}a_{22} - b_{32}a_{23} - b_{23}a_{32} + b_{22}a_{33} \\ c_{12} &= -b_{33}a_{21} + b_{31}a_{23} + b_{23}a_{31} - b_{21}a_{33} \\ c_{22} &= b_{33}a_{11} - b_{31}a_{13} - b_{13}a_{31} + b_{11}a_{33}. \end{aligned}$$

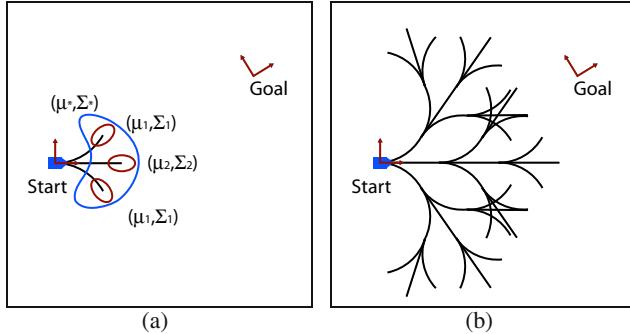


Fig. 1 (a) shows a planar robot with three noisy path options for a single step with means ∞_i and covariances Σ_i . The combined distribution of all possible trajectories for a single step has mean ∞_* and covariance Σ_* . (b) depicts an ideal tree of possible poses after three steps.

The proof of this theorem is similar to that in [23] except that it is for the planar case. The derivation of the covariance expression is shown in the Appendix.

4 Path-of-Probability (POP) Algorithm

We are interested in finding a path consisting of N intermediate steps that maximizes the probability of moving from a starting pose to a final pose. We assume that we have a finite set ζ of small trajectories that we can choose to use for a given step. For example, as shown in Fig. 1(a), we can choose from three different segments. Due to uncertainty, each segment has a distribution of possible poses with mean ∞_i and covariance Σ_i as illustrated by the red ellipses. We can then add multiple steps to begin to cover the environment as shown in Fig. 1(b) for three such steps. It can be computationally expensive to keep track of all possible trajectories and to propagate the uncertainty along each of these trajectories, especially as the number of choices in ζ and number of steps increases. In this section, we present the Path-Of-Probability (POP) algorithm, which is a computationally efficient way of determining which option we should choose for each step. The POP algorithm has been applied previously in slightly different manners for manipulator arms [10] and needle steering [18].

Let $g_i \in \zeta$ be the mean of the selected trajectory for the i th step. Suppose that the first $(i - 1)$ steps have been selected. In the POP algorithm, we want to select the step $g_i \in \zeta$ that has the highest probability of reaching the goal with the remaining $(N - i)$ steps. This can be written as

$$g_i = \arg \max_{g \in \zeta} f((g_1 \cdots g_{i-1} \circ g)^{-1} \circ g_{goal}; \infty_{N-i \cdots N}, \Sigma_{1 \cdots N}), \quad (22)$$

where ζ is the set of possible moves. If we have a way of exactly measuring ($\Sigma_{act} = 0$) the actual pose g_{act} before choosing a step, then the formulation is

$$g_i = \arg \max_{g \in \zeta} f((g_{act} \circ g)^{-1} \circ g_{goal}; \alpha_{N-i*...*N}, \Sigma_{N-i*...*N}). \quad (23)$$

Another variant of this algorithm could be to fuse uncertainty in measurements with the uncertainty in propagation. Here we restrict our attention to the problems in (22) and (23). The key in the POP algorithm is to calculate the convolved means and covariances of the remaining steps quickly.

In this paper, we assume that we do not have any prior knowledge of the type of path we would like to use for the remaining steps. Therefore, we combine all the individual distributions for a single step into a new distribution with mean α_* and covariance Σ_* as shown with the blue contour in Fig. 1(a). We can then use the propagation formulas in (20) and (21) recursively to propagate the distribution for the remaining steps. In a sense, this allows us to calculate the probability of arriving at the goal pose given all possible remaining trajectories.

We can sample data points from each individual distribution then use the following formulas to estimate the mean and covariance for a single step

$$\alpha_* = \alpha_* \circ \exp \left(\frac{1}{Q} \sum_{j=1}^Q \log(\alpha_*^{-1} \circ g_j) \right) \quad (24)$$

$$\Sigma_* = \frac{1}{Q} \sum_{j=1}^Q \log^\vee(\alpha_*^{-1} \circ g_j) [\log^\vee(\alpha_*^{-1} \circ g_j)]^T \quad (25)$$

where Q is the total number of data points from all trajectories in a single step. Depending on the number of samples, this can become computationally expensive, especially since the mean equation is recursive. However, if the individual distributions are highly focused around the individual means, we observed that we can obtain approximately the same results if we use

$$\alpha_* = \alpha_* \exp \left(\frac{1}{Z} \sum_{j=1}^Z \log(\alpha_*^{-1} \circ \alpha_j) \right) \quad \text{and} \quad (26)$$

$$\Sigma_* = \frac{1}{Z} \sum_{j=1}^Z \log^\vee(\alpha_*^{-1} \circ \alpha_j) [\log^\vee(\alpha_*^{-1} \circ \alpha_j)]^T \quad (27)$$

where Z is the total number of possible trajectories in one step. With the latter method, we do not need to sample from the individual distributions, we only need the means $\{\alpha_j\}$. If the set ζ is known beforehand, the mean α_* and Σ_* can be calculated numerically before beginning the POP algorithm.

The previous versions of the POP algorithm in [10] and [18] assume that the goal is fixed through out time and that the number of steps to reach the goal is known. We can remove these assumptions by using a two-part POP algorithm. In the first part, for step i with a maximum number of steps N , we estimate the best number of

remaining steps M to reach the target with

$$M = \arg \max_{m \in 1 \dots N-i} f((g_{act})^{-1} \circ g_{goal}((i+m)T); \infty_{1 \dots m}, \Sigma_{1 \dots m}) \quad (28)$$

where we use m convolutions of (∞_*, Σ_*) and T is the time of a single step. Note that the goal pose is evaluated at different discrete times in this part. After selecting the best number of steps to take, we choose a trajectory for a single step with

$$g_i = \arg \max_{g \in \zeta} f((g_{act} \circ g)^{-1} \circ g_{goal}((i+M)T); \infty_{i+2 \dots (M+i)}, \Sigma_{i \dots (i+M)}). \quad (29)$$

By breaking the problem into a two-part algorithm, we only have to evaluate $(N - i + Z)$ pdfs for each step i . We repeat this two-step process for each step until the current pose is within some tolerance of the goal pose. In the next section, we demonstrate this POP algorithm on a simple numerical example.

5 Example: Stochastic Kinematic Disc

In this section, we are interested in numerically testing a couple key aspects of this paper with the simple example of a disc that can roll but not slip in the plane. First, we verify that a Gaussian in exponential coordinates is a good fit. Second, we verify that the second-order propagation formulas can be used to calculate the mean and covariance of the convolution of two distributions. Finally, we demonstrate the POP algorithms outlined above.

The governing stochastic differential equation for this example is given by

$$(g^{-1}\dot{g})^\vee dt = \mathbf{h} dt + H d\mathbf{w} = \begin{pmatrix} v \\ 0 \\ \omega \end{pmatrix} dt + \begin{pmatrix} \sqrt{D_v} & 0 \\ 0 & 0 \\ 0 & \sqrt{D_\omega} \end{pmatrix} \begin{pmatrix} dw_v \\ dw_\omega \end{pmatrix}. \quad (30)$$

where $(g^{-1}\dot{g})^\vee$ are body fixed velocities [7], v and ω are translational and angular velocities, respectively, D_v and D_ω are noise coefficients and $d\mathbf{w} = [dw_v, dw_\omega]^T$ are unit-strength Wiener processes.

For small diffusion, the mean and covariance defined in (10) and (11) can be approximated with [18]

$$\infty(t) = \exp \left(\int_0^t \hat{\mathbf{h}} d\tau \right) \quad \text{and} \quad \Sigma(t) = \int_0^t A d^{-1}(\infty(\tau)) H H^T A d^{-T}(\infty(\tau)) d\tau, \quad (31)$$

which are essentially the deterministic path from integrating $\mathbf{h} dt$ and covariance propagation with only the A and B terms in (21) with infinitesimal steps. For an ideally straight trajectory ($\omega = 0$), the mean and covariance at time t is given by

$$\infty(t) = \begin{pmatrix} 1 & 0 & vt \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \text{and} \quad \Sigma(t) = \begin{pmatrix} D_v t & 0 & 0 \\ 0 & \frac{1}{2} D_\omega v^2 t^3 & \frac{1}{2} D_\omega v t^2 \\ 0 & \frac{1}{2} D_\omega v t^2 & D_\omega t \end{pmatrix}. \quad (32)$$

For a trajectory with constant angular and translational velocity, the disc ideally moves along a circular path. The mean and covariance of this case are

$$\boldsymbol{\alpha}(t) = \begin{pmatrix} \cos(\omega t) - \sin(\omega t) & \frac{v}{\omega} \sin(\omega t) \\ \sin(\omega t) & \cos(\omega t) & \frac{v}{\omega} (1 - \cos(\omega t)) \\ 0 & 0 & 1 \end{pmatrix} \quad \text{and} \quad \boldsymbol{\Sigma}(t) = \begin{pmatrix} \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \sigma_{12} & \sigma_{22} & \sigma_{23} \\ \sigma_{13} & \sigma_{23} & \sigma_{33} \end{pmatrix}, \quad (33)$$

where

$$\begin{aligned} \sigma_{11} &= \frac{1}{4\omega^3} [2t\omega(3D_\omega v^2 + D_v \omega^2) - 8D_\omega v^2 \sin(\omega t) + (D_w v^2 + D_v \omega^2) \sin(2\omega t)], \\ \sigma_{12} &= \frac{1}{\omega^3} [D_\omega v^2 - D_v \omega^2 - (D_\omega v^2 + D_v \omega^2) \cos(\omega t)] \sin^2\left(\frac{\omega t}{2}\right), \\ \sigma_{13} &= \frac{1}{\omega^2} D_\omega v (\omega t - \sin(\omega t)), \quad \sigma_{22} = \frac{1}{4\omega^3} (D_\omega v^2 + D_v \omega^2) (2\omega t - \sin(2\omega t)), \\ \sigma_{23} &= \frac{1}{\omega^2} D_\omega v (1 - \cos(\omega t)), \quad \text{and} \quad \sigma_{33} = D_\omega t. \end{aligned}$$

To verify that the integral version of the means and covariances are accurate, we numerically integrated the stochastic differential equation in (30) using a modified version of the Euler-Maruyama method [11] with a time step of $dt = 0.001$ for $T = 1$ for 25000 samples with noise levels of $D_v = 0.001$ and $D_\omega = 0.1$. We can calculate an estimate of the actual mean and covariance from (24) and (25), respectively, where Q is the number of sample data points.

For the ideal straight path with $v = 1$ and $\omega = 0$, we obtain from sample data

$$\boldsymbol{\alpha}_{\text{data}} = \begin{pmatrix} 1.000 & -0.001 & 1.000 \\ 0.001 & 1.000 & -0.000 \\ 0 & 0 & 1 \end{pmatrix} \quad \text{and} \quad \boldsymbol{\Sigma}_{\text{data}} = \begin{pmatrix} 0.001 & -0.000 & -0.000 \\ -0.000 & 0.034 & 0.050 \\ -0.000 & 0.050 & 0.100 \end{pmatrix};$$

from the integral propagation formula,

$$\boldsymbol{\alpha}_{\text{prop}} = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \text{and} \quad \boldsymbol{\Sigma}_{\text{prop}} = \begin{pmatrix} 0.001 & 0 & 0 \\ 0 & 0.033 & 0.050 \\ 0 & 0.050 & 0.100 \end{pmatrix}.$$

For an ideal circular path with $v = 1$ and $\omega = \frac{\pi}{2}$ we have from data

$$\boldsymbol{\alpha}_{\text{data}} = \begin{pmatrix} 0.000 & -1.000 & 0.637 \\ 1.000 & 0.000 & 0.637 \\ 0 & 0 & 1 \end{pmatrix} \quad \text{and} \quad \boldsymbol{\Sigma}_{\text{data}} = \begin{pmatrix} 0.010 & 0.012 & 0.023 \\ 0.012 & 0.021 & 0.041 \\ 0.023 & 0.041 & 0.101 \end{pmatrix}.$$

The integral propagation formulas provide

$$\boldsymbol{\alpha}_{\text{prop}} = \begin{pmatrix} 0 & -1 & 0.637 \\ 1 & 0 & 0.637 \\ 0 & 0 & 1 \end{pmatrix} \quad \text{and} \quad \boldsymbol{\Sigma}_{\text{prop}} = \begin{pmatrix} 0.010 & 0.013 & 0.231 \\ 0.013 & 0.021 & 0.041 \\ 0.023 & 0.041 & 0.100 \end{pmatrix}.$$

Given the sample data from each example, we can calculate normalized histogram contours and compare them to the contours generated from the Gaussian in (12) marginalized over the heading as shown in Fig. 2. Note that if we used the standard

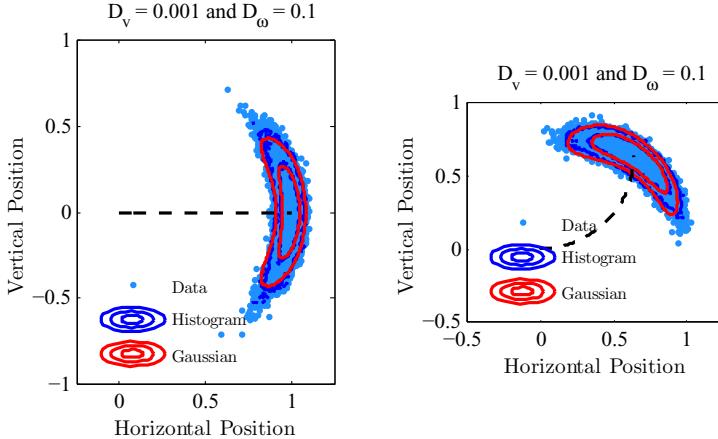


Fig. 2 25,000 sample points after integrating the stochastic differential equation of a noisy rolling disc for a straight path (left) and a circular path (right). Gaussian pdf contours marginalized over the heading from a propagated mean and covariance are compared to the contours from a histogram of the sample points.

formula for a Gaussian in \mathbb{R}^n we would have elliptical contours, which does not fit the data as well.

Now let's verify that the second-order propagation formulas are accurate for convolving two distributions. Here we assume the robot follows the straight example followed by the arc example with the same parameters as above. The mean and covariance from sample data following these paths were calculated to be

$$\hat{\boldsymbol{\alpha}}_{\text{data}} = \begin{pmatrix} -0.003 & -1.000 & 1.636 \\ 1.000 & -0.003 & 0.636 \\ 0 & 0 & 1 \end{pmatrix} \quad \text{and} \quad \boldsymbol{\Sigma}_{\text{data}} = \begin{pmatrix} 0.145 & 0.083 & 0.137 \\ 0.083 & 0.065 & 0.104 \\ 0.137 & 0.104 & 0.201 \end{pmatrix}.$$

By using the propagation formulas in (20) and (21) with the means and covariances from (32) and (33) we obtain

$$\hat{\boldsymbol{\alpha}}_{\text{prop}} = \begin{pmatrix} 0.000 & -1.000 & 1.638 \\ 1.000 & 0.000 & 0.637 \\ 0 & 0 & 1 \end{pmatrix} \quad \text{and} \quad \boldsymbol{\Sigma}_{\text{prop}} = \begin{pmatrix} 0.146 & 0.083 & 0.137 \\ 0.083 & 0.065 & 0.104 \\ 0.137 & 0.104 & 0.200 \end{pmatrix}.$$

Finally, we want to demonstrate the POP algorithm. Given a set ζ of ideally straight and circular trajectories, we can approximate the mean and covariance of each individual trajectory in closed form using the expressions in (32) and (33). Before applying the POP algorithm, we numerically calculate the mean $\hat{\boldsymbol{\alpha}}_*$ and covariance $\boldsymbol{\Sigma}_*$ for a single step. In the remaining examples, we use $D_v = 0.0001$ and $D_\omega = 0.01$ as well as a constant translational velocity $v = 1$ and a range of angular velocities $\omega \in [-\pi/3 : \pi/12 : \pi/3]$ for a step time of $T = 1$. When planning a path open loop

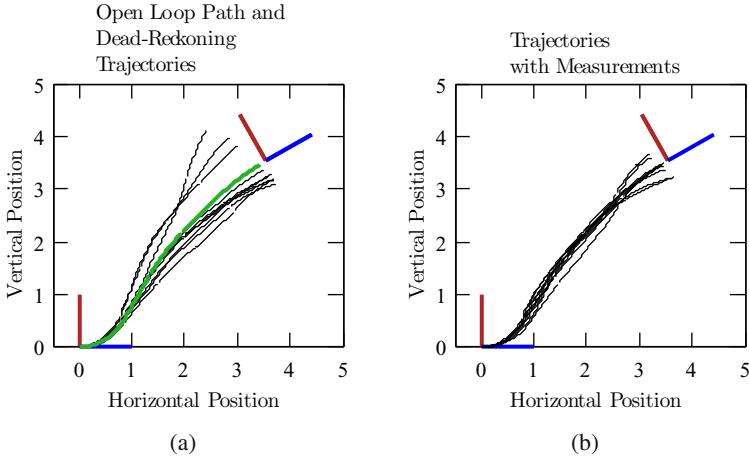


Fig. 3 (a) shows the open-loop optimal trajectory of the POP algorithm in green. Dead-reckoning trajectories are shown in black executing this sequence of steps. (b) shows ten paths executing the POP algorithm with absolute measurements.

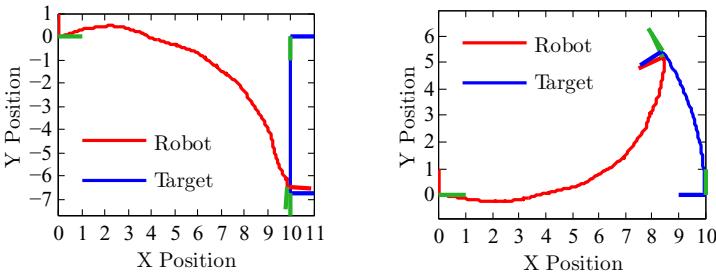


Fig. 4 The modified POP algorithm with absolute measurements implemented for two examples of a moving target and a variable number of steps to intercept

(no measurements) for a fixed goal, we use the POP algorithm with (22) to obtain the desired trajectories for each step. Fig. 3(a) shows an ideal path with five intermediate steps for an example goal pose with several dead-reckoning trajectories that try to follow this trajectory. Note that due to noise in the velocities the trajectories move away from the goal. When we have access to exact full pose information at each step, we select different step segments according to (23). Fig. 3(b) shows several actual trajectories using measurements at the end of each step.

The modified POP algorithm from (28) and (29) with measurements was tested with moving targets and a variable number of steps as shown in the two examples of Fig. 4 for approximately 25 steps with the same parameters as above. For both examples, the velocity of the moving target was chosen to be 50 percent of translational velocity of the robot to enable the robot to intercept it. This example demonstrates the propagation formulas as well as the modified POP algorithm.

6 Conclusion

In this paper, we focus on motion planning with uncertainty for planar systems. We represent all distributions with probability density functions. By using Lie group theory, we are able to derive closed-form expressions to efficiently propagate the mean and covariance of these distributions. We discuss the Path-of-Probability algorithm with these tools and extend this algorithm to include moving targets and a variable number of steps. This algorithm and propagation formulas were tested on a disc that can roll but not slip in the plane. In future work, we plan to introduce obstacles in the planning and to incorporate noisy measurements of both the robot and the target. We are also interested in applying these techniques to more complicated planar systems. Overall, these propagation formulas and the POP algorithm allow for quick and efficient motion planning of uncertain planar systems.

7 Appendix

7.1 Covariance Formula Proof

The proof of the expression for the mean to second order was shown in [23]. The proof of the covariance expression is slightly different than in [23] since we are working with $SE(2)$ instead of $SE(3)$. The covariance of the convolution $(f_1 * f_2)(g)$ about the mean ∞_{1*2} is given by

$$\Sigma_{f_1 * f_2} = \int_G \log^\vee(\infty_{1*2}^{-1} \circ g) [\log^\vee(\infty_{1*2}^{-1} \circ g)]^T (f_1 * f_2)(g) dg. \quad (34)$$

With a change of coordinates $g \rightarrow \infty_1 \circ \infty_2 \circ g$, this can be rewritten as

$$\Sigma_{f_1 * f_2} = \int_G \log^\vee(\infty_{\rho_1^{\infty_2} * \rho_2}^{-1} \circ g) [\log^\vee(\infty_{\rho_1^{\infty_2} * \rho_2}^{-1} \circ g)]^T (\rho_1^{\infty_2} * \rho_2)(g) dg = \Sigma_{\rho_1^{\infty_2} * \rho_2}. \quad (35)$$

Define the covariance C about the identity to be

$$C = \int_G \log^\vee(g) [\log^\vee(g)]^T f(g) dg. \quad (36)$$

If a pdf $f(g)$ has mean ∞ and covariance Σ , then expanding (36) with the Baker-Campbell-Hausdorff formula (BCH) (see next subsection)

$$C = \Sigma + \log^\vee(\infty) [\log^\vee(\infty)]^T + \frac{1}{2} (\Sigma ad^T(\log(\infty)) + ad(\log(\infty))\Sigma). \quad (37)$$

Since $\infty_{\rho_1^{\infty_2} * \rho_2}$ is second order, this means to second order

$$C_{\rho_1^{\infty_2} * \rho_2} = \Sigma_{\rho_1^{\infty_2} * \rho_2}. \quad (38)$$

The covariance C about the identity for $f(g) = (\rho_1^{\infty} * \rho_2)(g)$ is given by

$$C_{\rho_1^{\infty} * \rho_2} = \int_G \int_G \log^\vee(g) [\log^\vee(g)]^T \rho_1^{\infty}(h) \rho_2(h^{-1} \circ g) dh dg$$

Applying a change of coordinates $g \rightarrow h^{-1} \circ g$,

$$C_{\rho_1^{\infty} * \rho_2} = \int_G \int_G \log^\vee(h \circ g) [\log^\vee(h \circ g)]^T \rho_1^{\infty}(h) \rho_2(g) dh dg.$$

Let $X = \log(h)$ and $Y = \log(g)$. In the expansion of the log terms with the BCH, any terms in linear in X or Y will integrate to zero, which leaves the even terms as

$$\begin{aligned} & \left\{ (Z(X, Y))^\vee [(Z(X, Y))^\vee]^T \right\}_{\text{even}} \\ &= \mathbf{x}\mathbf{x}^T + \mathbf{y}\mathbf{y}^T + \frac{1}{4}ad(X)\mathbf{y}\mathbf{y}^T ad^T(X) \\ &+ \frac{1}{12}ad(X)ad(X)\mathbf{y}\mathbf{y}^T + \frac{1}{12}\mathbf{y}\mathbf{y}^T ad^T(X)ad^T(X) \\ &+ \frac{1}{12}ad(Y)ad(Y)\mathbf{x}\mathbf{x}^T + \frac{1}{12}\mathbf{x}\mathbf{x}^T ad^T(Y)ad^T(Y) + \dots \end{aligned} \quad (39)$$

Integrating the first two terms, we obtain the matrix A from

$$A = \int_G \mathbf{x}\mathbf{x}^T \rho_1^{\infty}(h) dh = \Sigma_{\rho_1^{\infty}} = Ad(\infty_2^{-1}) \Sigma_{\rho_1} Ad^T(\infty_2^{-1}) \quad (40)$$

and the matrix B from

$$B = \int_G \mathbf{y}\mathbf{y}^T \rho_2(g) dg = \Sigma_{\rho_2}. \quad (41)$$

Integration of the third term over g gives us

$$\int_G ad(X)\mathbf{y}\mathbf{y}^T ad^T(X) \rho_2(g) dg = ad(X)B ad^T(X)$$

and after integration over h we obtain

$$C(A, B) = \int_G ad(X)B ad^T(X) \rho_1^{\infty}(h) dh.$$

Additionally, we have

$$\int_G ad(X)ad(X)\rho_1^{\infty}(h) dh \int_G \mathbf{y}\mathbf{y}^T \rho_2(g) dg = A''B,$$

where the " operator rearranges the covariance matrix A . The remaining terms in the log expansion can be found similar to (7.1) by switching the order of A and B and using transposes.

7.2 Baker-Campbell-Hausdorff Formula

The *Baker-Campbell-Hausdorff formula* (BCH) [7] is a useful expression that will be used extensively in this paper to relate between the matrix exponential and the Lie bracket. The BCH is given by

$$\begin{aligned} Z(X, Y) &= \log(e^X e^Y) \\ &= X + Y + \frac{1}{2}[X, Y] + \frac{1}{12}([X, [X, Y]] + [Y, [Y, X]]) + \frac{1}{24}([X, [Y, [Y, X]]]) + \dots \end{aligned}$$

If the \vee operator is applied to this formula, we obtain

$$\begin{aligned} \mathbf{z} = &\mathbf{x} + \mathbf{y} + \frac{1}{2}ad(X)\mathbf{y} \\ &+ \frac{1}{12}(ad(X)ad(X)\mathbf{y} + ad(Y)ad(Y)\mathbf{x}) + \frac{1}{24}ad(X)ad(Y)ad(Y)\mathbf{x} + \dots \end{aligned}$$

Acknowledgments. This work was funded in part by NSF grant IIS-0915542 “RI: Small: Robotic Inspection, Diagnosis, and Repair” and Andrew Long’s NDSEG fellowship.

References

- [1] Alterovitz, R., Siméon, T., Goldberg, K.: The stochastic motion roadmap: A sampling framework for planning with markov motion uncertainty. In: *Robotics: Science and Systems*, pp. 246–253 (2007)
- [2] Alterovitz, R., Branicky, M., Goldberg, K.: Motion planning under uncertainty for image-guided medical needle steering. *Intl J. of Robotics Res.* 27(11-12), 1361–1374 (2008)
- [3] Barraquand, J., Ferbach, P.: Motion planning with uncertainty: The information space approach. In: *Proceedings of the IEEE Intl Conf. on Robotics and Automation*, vol. 2, pp. 1341–1348 (1995)
- [4] Bertsekas, D.: *Dynamic Programming and Optimal Control*, 3rd edn., vol. II. Athena Scientific, Nashua (2011)
- [5] Bry, A., Roy, N.: Rapidly-exploring random belief trees for motion planning under uncertainty. In: *Proceedings of the IEEE Intl Conf. on Robotics and Automation*, pp. 723–730 (2011)
- [6] Censi, A., Calisi, D., De Luca, A., Oriolo, G.: A Bayesian framework for optimal motion planning with uncertainty. In: *Proceedings of the IEEE Intl Conf. on Robotics and Automation*, pp. 1798–1805 (2008)
- [7] Chirikjian, G.: *Stochastic Models, Information Theory, and Lie Groups*, vol. 2. Birkhäuser (2012)
- [8] Chirikjian, G., Kyatkin, A.: *Engineering Applications of Noncommutative Harmonic Analysis*. CRC Press, Boca Raton (2001)
- [9] Choset, H., Lynch, K., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, L., Thrun, S.: *Principles of robot motion: theory, algorithms, and implementation*. MIT Press (2005)

- [10] Ebert-Uphoff, I., Chirikjian, G.: Inverse kinematics of discretely actuated hyper-redundant manipulators using workspace densities. In: Proceedings of the IEEE Intl Conf. on Robotics and Automation, vol. 1, pp. 139–145 (1996)
- [11] Higham, D.: An algorithmic introduction to numerical simulation of stochastic differential equations. *SIAM Review*, 525–546 (2001)
- [12] Kalman, R.: A new approach to linear filtering and prediction problems. *J. of Basic Eng.* 82, 35–45 (1960)
- [13] Lambert, A., Gruyer, D.: Safe path planning in an uncertain-configuration space. In: Proceedings of the IEEE Intl Conf. on Robotics and Automation, vol. 3, pp. 4185–4190 (2003)
- [14] LaValle, S.: Planning algorithms. Cambridge University Press (2006)
- [15] Lazanas, A., Latombe, J.: Landmark-based robot navigation. *Algorithmica* 13(5), 472–501 (1995)
- [16] Mason, R., Burdick, J.: Trajectory planning using reachable-state density functions. In: Proceedings of the IEEE Intl Conf. on Robotics and Automation, vol. 1, pp. 273–280 (2002)
- [17] Murray, R., Li, Z., Sastry, S.: A mathematical introduction to robotic manipulation. CRC (1994)
- [18] Park, W., Wang, Y., Chirikjian, G.: The path-of-probability algorithm for steering and feedback control of flexible needles. *Intl J. of Robotics Res.* 29(7), 813–830 (2010)
- [19] Pepy, R., Lambert, A.: Safe path planning in an uncertain-configuration space using rrt. In: Proceedings of the IEEE/RSJ Intl Conf. on Intelligent Robots and Systems, pp. 5376–5381 (2006)
- [20] Smith, R., Self, M., Cheeseman, P.: Estimating uncertain spatial relationships in robotics. In: Proceedings of the Second Annual Conf. on Uncertainty in Artificial Intelligence, pp. 435–461 (1986)
- [21] Thrun, S., Burgard, W., Fox, D.: Probabilistic robotics. MIT Press (2006)
- [22] Van Den Berg, J., Abbeel, P., Goldberg, K.: LQG-MP: Optimized path planning for robots with motion uncertainty and imperfect state information. *Intl J. of Robotics Res.* 30(7), 895 (2011)
- [23] Wang, Y., Chirikjian, G.S.: Nonparametric second-order theory of error propagation on motion groups. *Intl J. of Robotics Res.* 27(11-12), 1258–1273 (2008)

Intention-Aware Motion Planning

Tirthankar Bandyopadhyay, Kok Sung Won, Emilio Frazzoli, David Hsu,
Wee Sun Lee, and Daniela Rus

Abstract. As robots venture into new application domains as autonomous vehicles on the road or as domestic helpers at home, they must recognize human intentions and behaviors in order to operate effectively. This paper investigates a new class of motion planning problems with uncertainty in human intention. We propose a method for constructing a practical model by assuming a finite set of unknown intentions. We first construct a motion model for each intention in the set and then combine these models together into a single Mixed Observability Markov Decision Process (MOMDP), which is a structured variant of the more common Partially Observable Markov Decision Process (POMDP). By leveraging the latest advances in POMDP/MOMDP approximation algorithms, we can construct and solve moderately complex models for interesting robotic tasks. Experiments in simulation and with an autonomous vehicle show that the proposed method outperforms common alternatives because of its ability in recognizing intentions and using the information effectively for decision making.

1 Introduction

Motion planning is a critical capability for autonomous robots. The key issues of motion planning—geometry, kinematics, dynamics, uncertainties in robot control and sensing, *etc.* [13]—were identified many years ago. At the time, autonomous robots were mostly confined to tightly controlled environments, such as

Tirthankar Bandyopadhyay
Singapore-MIT Alliance for Research and Technology, Singapore
e-mail: tirtha@smart.mit.edu

Kok Sung Won · David Hsu · Wee Sun Lee
National University of Singapore, Singapore
e-mail: {koksung, dyhsu, leews}@comp.nus.edu.sg

Emilio Frazzoli · Daniela Rus
Massachusetts Institute of Technology, Cambridge, MA, USA
e-mail: frazzoli@mit.edu, rus@csail.mit.edu



Fig. 1 Our experimental platform, a robot golf cart, during an encounter with pedestrians in a parking lot. Each pedestrian is walking towards one of several goals. Unaware of the pedestrians' intentions in advance, the vehicle must pass over them as quickly as possible and avoid accidents.

manufacturing factory floors, where they seldom actively interact with humans. As robots venture into new application domains as autonomous vehicles on the road or as domestic helpers at home, they must recognize human intentions and behaviors in order to operate effectively.

Recognizing human intention is difficult. In principle, estimating intention is similar to estimating other more common quantities such as the robot position and velocity. The true state is inferred from sensor data with some uncertainty. However, recognizing intention is often more difficult, because of the diversity and subtlety of human behaviors and the lack of a powerful “intention sensor”. Further, a robot may actively gather information for intention recognition by taking sensing actions, but must balance such actions against those contributing directly to the goal of motion planning. The robot's ultimate goal is to complete the specified tasks and not to recognize intention. It thus should not gather more information than necessary.

We propose to treat intention-aware motion planning as planning under uncertainty and model it as a partially observable Markov decision process (POMDP) [9, 19], in fact, a recently introduced variant called the *mixed observability Markov decision process* (MOMDP) [15]. Our MOMDP model assumes that the robot interacts with an intentional agent, *e.g.*, a human. The agent has a finite set of intentions, each embodied in an observable behavior. Given the intention, the agent's behavioral dynamics is modeled in advance and known to the robot. The agent's intention is then the primary source of uncertainty for motion planning and the main partially observable state variable in the MOMDP. The remaining state variables, which specify the agent's and the robot's dynamics, may be either fully or partially observable. Consider, for example, an autonomous robot vehicle in an encounter with pedestrians (Fig. 1). Each pedestrian is walking towards one of several goals, some of which may lead him to cross the road. The robot does not know the pedestrian intention, in this case, the goal in advance and must infer it based on observed pedestrian behavior. The robot then acts accordingly. By modeling the encounter as a MOMDP and solving it, we obtain a conditional plan that enables the robot to act optimally (with respect to the model) despite uncertainty on the pedestrians' intentions.

This work introduces the intention of an interacting agent into motion planning and proposes the MOMDP as a model for it. The MOMDP formulation of intention-aware motion planning provides several key advantages:

- The MOMDP is a rich probabilistic model that captures uncertainties in intention as well as robot control and sensing.
- It divides the complex task of recognizing agent intentions into two simpler parts. We first construct a model of the agent's behavioral dynamics for each intention and solve the resulting MOMDP model for a plan. The execution of the plan performs inference over a finite number of intentions already modeled, based on observed agent behavior. This approach simplifies both modeling and inference.
- Our MOMDP model treats intention as a single partially observable state variable and limits uncertainty over intention to a small portion of the state space. The latest MOMDP algorithm exploits this modeling feature to achieve dramatic computational efficiency gain over standard POMDP algorithms [12, 15]. The scalability of MOMDPs makes them useful for modeling moderately complex robotic tasks.
- Similar to its POMDP counterpart, the MOMDP model provides a principled general approach to planning under uncertainty and optimally balances information-gathering and task completion actions.

We evaluated our approach on two navigation tasks both in simulation and on a robot golf cart: pedestrian interaction and interaction navigation. In the first task, intentions represent pedestrian goals. In the second task, the autonomous vehicle navigates through an uncontrolled intersection and encounters another vehicle. Here, intention reflects the preference of the approaching vehicle's driver, *e.g.*, the level of caution. Experimental results show that the MOMDP approach outperforms the more common approaches such as reactive planning and Bayesian intention inference in terms of safety and navigation efficiency.

2 Background

2.1 Related Work

Our work touches on several distinct lines of research in the literature. Motion planning is a vast field [13]. Over the years, many important issues have been identified, including the geometry of robots and environments, the kinematics and dynamics of robots, uncertainties in robot control and sensing, *etc.*. Our work introduces a new aspect, the intention of an interacting agent, into motion planning.

Both plan recognition and activity recognition deal with agent intentions (see, *e.g.*, [10, 21]). While they solve the state estimation problem of identifying an agent's plan or activity, we solve a control problem, in which a robot's goal is to complete specified tasks and resolve the intention only when necessary. One distinguishing feature is the need to balance actions that gather information for intention recognition and actions that contribute directly to task completion.

Our work is more closely related to the Hidden Goal Markov Decision Process (HGMDP) and the Helper Action Markov Decision Processes (HAMDP) [6]. Both model a helper agent trying to recognize another agent’s intention and perform assistive actions. Solving HGMDPs exactly is PSPACE-hard [6]. The earlier work does not provide a practical way of solving HGMDPs, but it introduces HAMDPs, which place restrictions on how the two agents interact, for computational efficiency.

We model intention-aware motion planning as MOMDPs, which are structured variants of POMDPs. POMDPs provide a general framework for planning under uncertainty. Although solving POMDPs exactly is computationally intractable in the worst case [16], point-based approximation algorithms have greatly improved the speed of POMDP planning in recent years [12, 17, 20]. Today the fastest algorithms, such as HSVI [20] and SARSOP [12], can solve POMDPs with hundreds of thousands states in reasonable time. MOMDPs specify additional structural information on the corresponding POMDPs and dramatically improve efficiency of point-based approximation algorithms under suitable conditions [15].

2.2 Preliminaries on MDPs and POMDPs

A Markov decision process (MDP) models a system taking a sequence of actions under uncertainty to maximize its total rewards. Formally, an infinite-horizon discrete MDP is a tuple $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$, where \mathcal{S} and \mathcal{A} denote the system’s state space and action space, respectively. At each time step, the system takes an action $a \in \mathcal{A}$ and moves from the current state $s \in \mathcal{S}$ to a new state $s' \in \mathcal{S}$ with a conditional probability function $T(s, a, s') = p(s'|s, a)$, modeling the system dynamics.

At each time step, the system receives a real-valued reward $R(s, a)$ that depends on its state s and action a . The goal of the system is to choose a sequence of actions that maximizes the expected total reward $E(\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t))$, where s_t and a_t denote the system’s state and action at time t , and $\gamma \in (0, 1)$ is a discount factor that reflects the preference of immediate rewards over future ones.

Solving an MDP, produces a *policy* $\pi: \mathcal{S} \rightarrow \mathcal{A}$, prescribing an action $a \in \mathcal{A}$ for each system state $s \in \mathcal{S}$. An optimal policy π^* maximizes the agent’s expected total reward.

As MDP does not address observation uncertainty, due to, *e.g.*, imperfect sensors, we need the POMDP, which adds two additional elements: the observation space \mathcal{O} and the observation function Z . At each time step, the system receives an observation o after taking action a and arriving in state s' . The observation function is again specified as a conditional probability function $Z(s', a, o) = p(o|s', a)$, which models observation uncertainty.

In a POMDP, the system state is not known exactly and is represented as a probability distribution $b(s)$ over \mathcal{S} , commonly called a *belief*. Suppose that the beliefs of a discrete POMDP are represented as vectors and $|\mathcal{S}|$ is the number of states in the POMDP. The space \mathcal{B} of all possible beliefs then forms an $(|\mathcal{S}| - 1)$ -dimensional simplex, as the probabilities over \mathcal{S} must sum up to 1.

In contrast to an MDP policy, a POMDP policy is a mapping $\pi: \mathcal{B} \rightarrow \mathcal{A}$, which prescribes a action $a \in \mathcal{A}$ for each belief $b \in \mathcal{B}$. The policy π induces a *value function* $V_\pi: \mathcal{B} \rightarrow \mathbb{R}$. The *value* of b with respect to π is the system's expected total reward of executing π with initial belief b : $V_\pi(b) = E\left(\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid \pi, b\right)$. The value function V^* for an optimal policy π^* can be approximated arbitrarily closely by a piecewise-linear convex function

$$V(b) = \max_{\alpha \in \Gamma} \sum_{s \in S} \alpha(s)b(s), \quad (1)$$

where each $\alpha \in \Gamma$ is represented as a vector and called an α -vector. Each α -vector defines a hyperplane $h(b) = \sum_{s \in S} \alpha(s)b(s)$ over \mathcal{B} . Fast discrete-state POMDP algorithms [12, 17, 20] exploit the α -vector representation.

The value function V can be then represented as a finite set of hyperplanes. Most of the fastest discrete-state POMDP algorithms [12, 17, 20] represent a policy by its value function and exploit the α -vector representation for efficient computation. As the value function is defined over \mathcal{B} , a high-dimensional belief simplex is a major obstacle to computational efficiency.

Each α -vector is associated with an action. Once a value function is computed, the corresponding policy can be executed by selecting the action associated with the best α -vector at the current belief b , using (1).

3 Intention-Aware Motion Planning as a MOMDP

Consider a robot interacting with an intentional agent. We divide the complex task of recognizing agent intentions and acting optimally into two stages. In the off-line stage, we construct a motion model for each agent intention (Section 3.1) and solve the resulting MOMDP model for a policy (Section 3.2). In the on-line stage, the policy enables the robot to make an inference over a finite set of agent intentions and act accordingly, based on observed agent behavior Section 3.3.

3.1 Modeling

Let \mathcal{X} and \mathcal{A} denote the robot's state space and action space, respectively. Let \mathcal{Y} denote the agent's state space. The robot's motion is governed by the probabilistic transition function $T_x(x, a, x') = p(x'|x, a)$, where $x, x' \in \mathcal{X}$ are the robot's current and next state and $a \in \mathcal{A}$ is an admissible robot action (Fig. 2). The robot may observe both its own and the agent's state, and the probabilistic observation function is given by $Z(x, y, o) = p(o|x, y)$, for a robot state $x \in \mathcal{X}$, an agent state $y \in \mathcal{Y}$, and an observation $o \in \mathcal{O}$.

The agent's motion is governed by another probabilistic transition function $T_y(y, a', y') = p(y'|y, a')$ for $y, y' \in \mathcal{Y}$ and some agent action $a' \in \mathcal{A}'$. To relate the agent's action a' to its intention $g \in \Theta$, we further assume that a' is the result of the agent executing a policy $\rho: \mathcal{X} \times \mathcal{Y} \times \Theta \rightarrow \mathcal{A}'$, which chooses a' based on the current robot state x , the current agent state y , and the agent's intention θ . Recall our earlier example in which a robot vehicle encounters a pedestrian. It is

reasonable to expect that the pedestrian chooses an action based on the robot's and his own position and velocity as well as his intention, in this case, the goal location. The form of ρ indicates that the agent has perfect information on the robot's and its own state. Although these assumptions may not hold exactly, it provides a reasonable trade-off between model fidelity and computational complexity, as we show in Section 4.

There are various ways to construct the policy ρ . One possibility is to specify ρ manually. This potentially provides high infidelity in reproducing the agent's behavior, but becomes tedious for an agent with complex behavior. Another possibility is to compute ρ by solving a simplified MDP model. We illustrate this approach with our vehicle-pedestrian encounter example. Assume that the pedestrian approaches the goal location by optimizing an objective function, e.g., following the shortest path and avoiding collision with the vehicle. Assume also that the pedestrian's actions may be imperfect. All these can be captured in an MDP, which is then solved to generate a policy ρ for the pedestrian. Now substituting $a' = \rho(x, y, \theta)$ into T_y , we obtain $T_y(y, \rho(x, y, \theta), y')$, which clearly shows the dependence of the agent's motion on its intention. For simplicity, we assume that y' does not depend on x' , but the dependence can be added if necessary.

To summarize, our MOMDP model is a tuple $(\mathcal{X}, \mathcal{Y}, \Theta, \mathcal{A}, \mathcal{O}, T_x, T_y, Z, R, \gamma)$, which consists of the joint state space $\mathcal{X} \times \mathcal{Y} \times \Theta$, the robot action space \mathcal{A} , the robot observation space \mathcal{O} , the transition functions T_x and T_y , the observation function Z , a reward function $R: \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}$, and a discount factor γ . To complete the model, we must also specify which variables are fully or partially observable. In general, the intention is always partially observable. Other variables, which describe the robot's or the agent's state, may be partially observable as well.

3.2 Policy Computation

To compute a policy, we apply SARSOP [12, 15], a leading point-based approximation algorithm, to our MOMDP model. We give a brief description of the algorithm here for completeness. SARSOP samples incrementally a set of points from the belief space \mathcal{B} and maintains a set of α -vectors, which represents a piecewise-linear lower-bound approximation \underline{V} to the optimal value function V^* . We can view the sampling process as building a *belief search tree* rooted at a given initial belief $b_0 \in \mathcal{B}$. The nodes of the tree correspond to beliefs in \mathcal{B} , and the edges correspond to action-observation pairs. A child node b' is connected to its parent b by an edge (a, o) , if performing the action a and receiving the observation o update the belief b to a new belief b' . Thus every belief in the tree is reachable from b_0 through a sequence of action and observation updates, and the approximation \underline{V} is constructed over a subset $\mathcal{R}(b_0)$ of beliefs reachable from b_0 . To compute \underline{V} , SARSOP uses value iteration [18], which is based on the idea of dynamic programming.

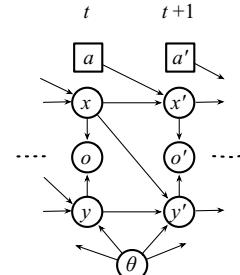


Fig. 2 A MOMDP model for intention-aware motion planning

Exploiting the fact that V^* must satisfy the Bellman equation, value iteration starts with an initial set of α -vectors and performs backup operations on the α -vectors at the sampled points by iterating on the Bellman equation, until the iteration converges.

Compared with the standard POMDP model, a major advantage of our MOMDP model is computational efficiency. A POMDP models all components of a system's state in a single variable, thus forcing the state variable to be partially observable even if only one component is partially observable. To simplify the presentation, let us assume that in the rest of this section, intention θ is the only partially observable component in our model. In this case, the dimensionality of \mathcal{B} is nevertheless $|\mathcal{X}||\mathcal{Y}||\Theta| - 1$ for the POMDP model (see Section 2.2). The belief search tree must be constructed in this high-dimensional space. Furthermore, the primitive objects in SARSOP—the beliefs and α -vectors—must be represented and computed over this space as well. All these increase computational cost, unnecessarily.

The MOMDP model separates the fully and partially observable state components through a factored model. If θ is the only partially observable variable, a belief is represented as $b = (x, y, b_\theta)$, where x and y are the fully observable robot and agent states and b_θ is a belief over the partially observable agent intentions. The belief space \mathcal{B} then becomes a union of subspaces: $\mathcal{B} = \{\mathcal{B}_\theta(x, y) \mid x \in \mathcal{X}, y \in \mathcal{Y}\}$. For each x and y , $\mathcal{B}_\theta(x, y)$ is a space of beliefs over the intentions, and its dimensionality is only $|\Theta| - 1$, a drastic reduction compared with the POMDP model. Correspondingly, a MOMDP value function $V(b) = V(x, y, b_\theta)$ is represented as a collection of α -vector sets: $\{\Gamma_\theta(x, y) \mid x \in \mathcal{X}, y \in \mathcal{Y}\}$, where for each x and y , $\Gamma_\theta(x, y)$ is a set of α -vectors defined over $\mathcal{B}_\theta(x, y)$. Geometrically, each α -vector set $\Gamma_\theta(x, y)$ represents a *restriction* of V to the subspace $\mathcal{B}_\theta(x, y)$, obtained by restricting the domain of V from \mathcal{B} to $\mathcal{B}_\theta(x, y)$. In the MOMDP policy computation, SARSOP computes only these restrictions in the lower-dimensional subspaces $\mathcal{B}_\theta(x, y)$ for $x \in \mathcal{X}$ and $y \in \mathcal{Y}$, because there is no uncertainty over x or y .

3.3 Policy Execution

After computing a MOMDP policy, represented as a value function $V(x, y, b_\theta)$, we execute the policy by repeating two steps. The first step selects an action for the current belief (x, y, b_θ) . To evaluate $V(x, y, b_\theta)$, we use x and y as an index to find the right α -vector set and then find the maximum α -vector from the set:

$$V(x, y, b_\theta) = \max_{\alpha \in \Gamma_\theta(x, y)} \{\alpha \cdot b_\theta\}. \quad (2)$$

We then execute the action associated with the chosen α -vector. The second step updates the belief. Suppose that after the robot takes action a , the new robot state is x' and the new agent state is y' . The robot receives an observation o , which is exactly (x', y') , as the robot state and the agent state are fully observable. The new belief is then (x', y', b'_θ) , where

$$b'_\theta(\theta) = \eta T_y(x, y, \theta, y') b_\theta(\theta) \quad (3)$$

and η is a normalizing constant.

In general, some of the constituent variables in x and y may be partially observable as well. Let s_f denote all the fully observable variables, and let s_p denote all the partially observable variables, including, in particular, the intention θ . Let \mathcal{S}_f and \mathcal{S}_p be the subspaces in \mathcal{S} for s_f and s_p , respectively. We then have

$$V(s_f, b_{\mathcal{S}_p}) = \max_{\alpha \in \Gamma_{\mathcal{S}_p}(s_f)} \{\alpha \cdot b_{\mathcal{S}_p}\}. \quad (4)$$

$$b_{\mathcal{S}_p}(s'_p) = \eta Z(x', y', o) \sum_{\theta \in \Theta} T_x(x, a, x') T_y(x, y, \theta, y') b_{\mathcal{S}_p}(s_p). \quad (5)$$

Equations (2) and (3) are merely the special case when $s_f = (x, y)$ and $s_p = \theta$. The MOMDP model's computational advantage decreases with an increasing number of partially observable variables. However, it is always at least as efficient as the corresponding POMDP and significantly reduce the computational cost when a system has only a few partially observable variables.

4 Experiments

We now present experimental results on our approach evaluated on two autonomous vehicle navigation tasks: pedestrian interaction (Sections 4.1–4.2) and intersection navigation (Section 4.3). There has been significant interest in safe navigation of autonomous vehicles in recent years (see, e.g., [1, 4, 7]). However, the main objective of our work here is to propose a general approach to intention-aware motion planning. A comparison with these alternative approaches in the specific context of autonomous vehicle navigation will be explored in future work.

4.1 Pedestrian Interaction

Recall the pedestrian avoidance task from Section 1. We model it as a MOMDP. The robot vehicle and the pedestrian traverse in an environment discretized into a uniform grid. Each grid cell has size $1 \text{ m} \times 1 \text{ m}$. The robot has three velocity levels: 0 m/s , 1 m/s , and 2 m/s . The robot's state consists of its position and velocity. It has three actions that control the acceleration: ACCELERATE (1 m/s^2), MAINTAIN (0 m/s^2), and DECELERATE (-1 m/s^2). The robot motion may be noisy. We assume that the robot can sense its own position and the pedestrian's position perfectly, as the laser range finder on-board our robot golf cart (Fig. 1) is sufficiently accurate, with respect to the discretized environment. The robot gets a reward for safely passing over the pedestrian and gets penalties for collision with the pedestrian, speeding, and time delay. Each time step has duration of 1 second.

To model the pedestrian behavior, we use a simplified version of Helbing *et al.*'s pedestrian motion model [8], which has been carefully validated on empirical data. This model resembles the potential field method [11] for motion planning: the pedestrian is attracted to a goal and is repelled by the robot vehicle, which is treated as a quasi-static obstacle. The pedestrian movement may be noisy as well.

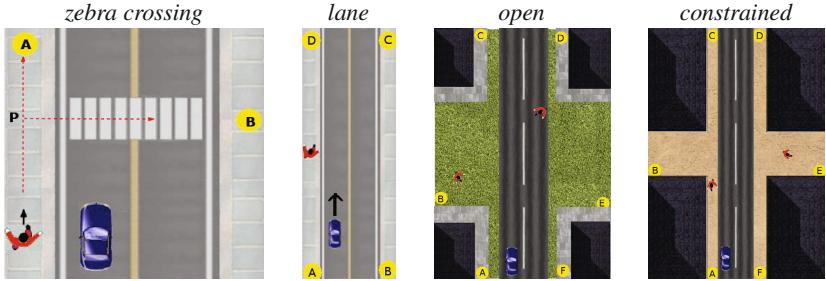


Fig. 3 Four test environments. The letters in yellow circles mark pedestrian goals. The dashed curves in the leftmost environment mark the pedestrian paths towards the goals.

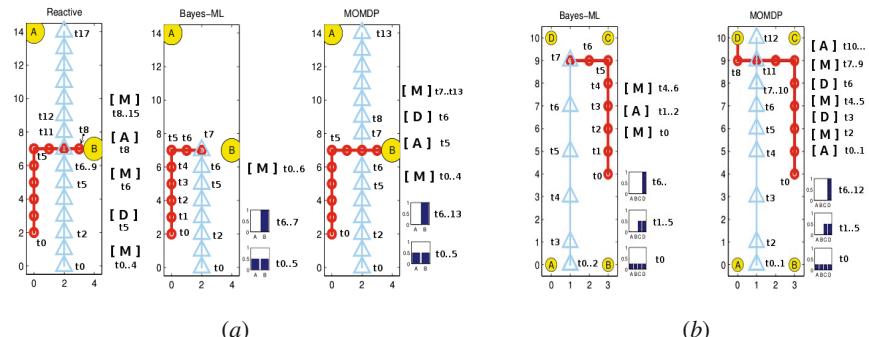


Fig. 4 Simulation runs comparing MOMDP policies with two other methods in the (a) zebra crossing and the (b) lane environments. Red curves mark the pedestrian's trajectories. Light blue curves mark the robot's trajectories. The histograms show the robot's belief on the pedestrian's goal over time. “A”, “M”, and “D” indicate the robot's three actions.

The hidden intention is the pedestrian's goal location, which the robot does not know in advance and cannot sense directly.

For comparison, we consider two common pedestrian avoidance methods. The first one is simple reactive planning. At each time step, if the pedestrian is within a predefined distance from the robot according to the current observation, the robot slows down to a full stop and waits until the pedestrian passes by. The second method, Bayes-ML, pre-computes a set of MDP policies, each assuming a known pedestrian goal. It then performs Bayesian inference on the pedestrian's goal based on the received observations. Just as our MOMDP policy, Bayes-ML maintains a belief over pedestrian goals by incorporating the observation at each time step (implementing the same update Eqn. 3). However, it chooses the action by determining a most likely goal from the current belief and looking up the action from the pre-computed MDP policy for this goal.

We compared these methods in four environments in simulation (Fig. 3). In the *zebra crossing* environment, the robot vehicle encounters a pedestrian, who may follow two possible trajectories, one proceeding straight ahead and one crossing the

Table 1 Performance comparison of Bayes-ML and MOMDP in four test environments

| Environment | Noise | Bayes-ML | | MOMDP | |
|----------------|-------|-------------|----------|-------------|----------|
| | | Time | Accident | Time | Accident |
| zebra crossing | zero | 14.8 (0.4) | 4.8% | 14.7 (0.5) | 0% |
| | high | 17.0 (3.6) | 2.8% | 16.8 (3.5) | 2.1% |
| lane | zero | 7.6 (1.0) | 2.5% | 7.7 (1.2) | 0.3% |
| | low | 9.1 (1.8) | 3.1% | 9.1 (2.2) | 1.9% |
| | med | 8.6 (1.8) | 6.1% | 8.6 (1.8) | 5.9% |
| | high | 10.1 (4.0) | 5.8% | 10.7 (4.4) | 5.0% |
| open | zero | 11.3 (0.9) | 0.8% | 11.4 (1.1) | 0.02% |
| | low | 13.5 (2.6) | 1.7% | 13.6 (2.6) | 1.3% |
| | med | 14.3 (3.8) | 2.5% | 14.5 (4.0) | 2.2% |
| | high | 14.6 (4.5) | 3.0% | 14.5 (4.4) | 2.7% |
| constrained | zero | 11.3 (0.8) | 1.6% | 11.5 (1.2) | 0.07% |
| | low | 13.5 (2.5) | 1.7% | 13.7 (2.5) | 0.9% |
| | med | 14.6 (4.0) | 3.6% | 15.1 (4.3) | 3.3% |
| | high | 18.4 (10.4) | 4.0% | 21.9 (13.5) | 3.2% |

road over the zebra stripes. The *lane* environment is similar, but has four pedestrian goals. Furthermore, as there is no zebra marking, the pedestrian can “jaywalk” towards the goals directly. The *open* environment emulates a road going through open space, *e.g.*, in a park. The *constrained* environment emulates a narrow lane between city blocks with pedestrian walkways across. The *open* and the *constrained* environments look similar. However, the latter is more difficult, because many pedestrian trajectories overlap due to the constrained space, making it challenging to recognize the pedestrian’s true goal. The performance statistics are shown in Table 1, and some sample simulation runs are shown in Fig. 4.

Fig. 4a shows that that for reactive planning, the robot slows down to a full stop. It waits until the pedestrian crosses the road and is sufficiently far. It then proceeds. If the pedestrian stands at the location P (see Fig. 3, leftmost) for a prolonged time, the robot must wait, as the pedestrian is within the predefined distance. This is clearly unacceptable and is an inherent limitation of simple reactive planning, which does not look ahead and plan for the future.

We now focus on the comparison between Bayes-ML and our MOMDP policy, which both perform probabilistic inference on the pedestrian’s goal. For each test environment, we varied the system noise level, and constructed and solved a MOMDP model. For the *zebra crossing* environment, we varied the noise level in robot motion. In the other three environments, we varied pedestrian movement noise, which makes intention inference more difficult. Each data entry in Table 1 is the result over 4,500 simulation runs. Columns 3 and 5 of the table report the average time and the standard deviation for the robot to pass over the pedestrian and reach the goal. Columns 4 and 6 report the rate of accident, which occurs when

the robot and pedestrian are within a predefined distance. For easy comparison, we tuned the MOMDP reward functions so that the time to clear is roughly the same for Bayes-ML and the MOMDP policies. The statistics show that the MOMDP policies consistently have lower accident rate, sometimes substantially.

To understand the reasons behind MOMDP policies' better performance, let us look at some simulation runs (Fig. 4). In the *zebra crossing* environment (Fig. 4a), Bayes-ML maintains a belief over the pedestrian's goal based on the observations. However, the initial stretches of the two pedestrian trajectories towards *A* and *B* overlap. The robot has no information to distinguish the pedestrian's goal until time step $t = 5$ s, when the pedestrian reaches the location *P*. The histogram shows that the belief at $t = 5$ s is roughly uniform, but not exactly because of noise in pedestrian movement. Bayes-ML forces the robot to act according to the most likely goal and ignores the alternative completely. If goal *A* happens to have slightly higher probability, the robot will maintain its current velocity. At $t = 6$ s, the new observation changes the belief and shows clearly that the pedestrian is crossing the road towards *B*. However, it is too late by then. Because of the robot's dynamics, no action can prevent an accident. It cannot stop in time with DECELERATE. Neither can it overtake the pedestrian fast enough with ACCELERATE. Bayes-ML applies MAINTAIN simply because it incurs the lowest cost. One main weakness of Bayes-ML is its failure to exploit the full information in the belief and instead to choose the action based solely on the mostly likely estimate. It is overly confident and does not hedge against all possible situations.

When faced with the same roughly uniform belief at $t = 5$ s, the MOMDP policy reasons about the future effects of both goals *A* and *B* and realizes the danger if the pedestrian crosses the road. It chooses ACCELERATE so that the robot passes over the pedestrian before any dangerous situation occurs. At $t = 6$ s, the robot is assured of safely overtaking the pedestrian at its current velocity. The policy then chooses DECELERATE in order to avoid the penalty incurred in high-velocity states. It is interesting to note that the key action, ACCELERATE, is decided at $t = 5$ s, when the pedestrian's true goal is still largely unresolved (see the belief histogram in Fig. 4a).

The simulation run in the *lane* environment (Fig. 4b) provides more information on the MOMDP policies' behavior. Initially, the robot's belief on the pedestrian's goal is uniform. At $t = 1$ s, the robot observes the pedestrian's forward movement. The belief now peaks on goals *C* and *D*. The robot then chooses ACCELERATE. As the robot catches up with the pedestrian, it chooses DECELERATES at $t = 3$ s to stay a safe distance behind, in case the pedestrian suddenly crosses the road. At $t = 6$ s, the pedestrian steps off the curb. The updated belief concentrates almost entirely on the goal *D*, and the pedestrian's intention is now resolved. The robot decelerates until it fully stops, waits for the pedestrian to cross the road, and then proceeds.

Comparison of the MOMDP policies' behavior in Fig. 4a and Fig. 4b clearly demonstrates the MOMDP's unique ability in optimally balancing exploration and exploitation. It gathers information and resolves the uncertainty in the pedestrian's intention only when necessary.

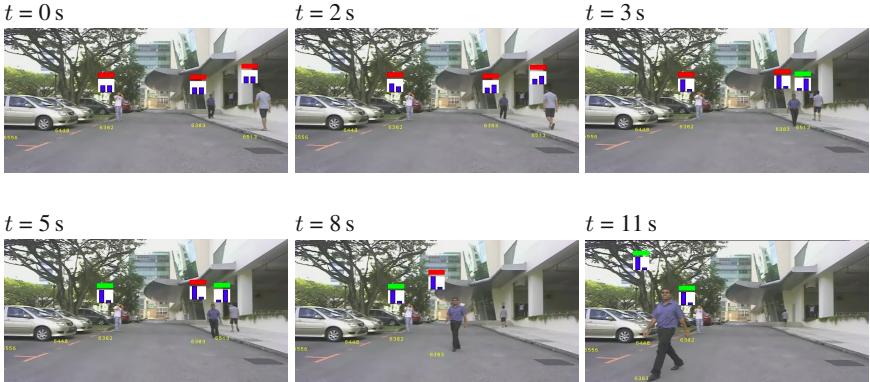


Fig. 5 A test run of the robot golf cart under the control of a MOMDP policy. The pictures are taken from a camera on-board the vehicle. A full view of the test environment is shown in Fig. 1. There are three pedestrians. For each pedestrian, the blue histogram shows the summary belief of whether the pedestrian will stay on the left or right side of the road. The color bar indicates the action chosen with respect to the pedestrian. Red means to decelerate from high speed or maintain low speed. Green means to accelerate from low speed or maintain high speed.

4.2 A Robot Golf Cart Interacting with Multiple Pedestrians

We implemented and tested a MOMDP controller on a robot golf cart (Fig. 1), presented in [5]. The robot is equipped with a perception module to localize and detect pedestrians with sufficient accuracy in urban environments.

In order to handle multiple pedestrians, we instantiate a MOMDP controller for each pedestrian detected. These controllers operate independently. We then combine their output actions by choosing the most conservative one.

Fig. 5 shows a test run of the vehicle in a parking lot. There are three pedestrians. Initially the beliefs on all the pedestrians' goals are uniform. The time step $t = 2$ s is quite interesting. As a result of inherent stochastic noise in human movement, the beliefs seem to indicate that none of the pedestrians will cross the road. In this case, Bayes-ML would choose to accelerate. The red color bars in the picture show that for every pedestrian, the MOMDP controller chooses to either maintain low speed or decelerate. The reason is that the probability that a pedestrian suddenly crosses the road is still substantial and the penalty for an accident is high. With the additional observation received at $t = 3$ s, the belief indicates that the rightmost pedestrian will not cross; the MOMDP controller decides that it is safe to move on with respect to him (see the green bar in the picture). Then, the belief at $t = 5$ s indicates that the leftmost pedestrian will also not cross, and it is safe to move on. However, the middle pedestrian is crossing the road. The action chosen with respect to him is to decelerate or maintain low speed (see the red bar). This is the most conservative action and is actually executed. At $t = 8$ s, the rightmost pedestrian is far away and is no longer detected as a potential threat. At $t = 11$ s, the middle pedestrian has

crossed. The green bars show that the actions chosen now for all pedestrians are to accelerate. The vehicle moves on and safely pass over the pedestrians.

This test uses the same model as that for the *lane* environment with pedestrian movement noise. The result shows that our approach is robust under uncertainty in robot control and sensing, and is scalable with respect to multiple pedestrians.

4.3 Intersection Navigation

The intersection navigation task is motivated by a near-miss accident in the 2007 DARPA Urban Challenge [14]. Two autonomous vehicles, R and A , approach an uncontrolled traffic intersection (Fig. 6). R comes to a stop and then resumes its forward motion after checking for precedence. A wants to make a left turn, but fails to yield. Without understanding A 's intention, R continues its forward motion. The two vehicles got so close that an emergency mechanism was activated to stop both. We want to model this scenario as intention-aware motion planning and evaluate whether a MOMDP policy can potentially improve safety and efficiency when an autonomous vehicle navigates through an intersection. In the special case of two autonomous vehicles, intentions can in principle be communicated directly. However, in the foreseeable future, autonomous vehicles must interact with other vehicles with human drivers. Such direct communication will not be possible then.

Following the approach in Section 3, we built a MOMDP model for vehicle R (Fig. 7). The state of R consists of its position and velocity. The environment is discretized into a uniform grid. Each grid cell has size $2.5 \text{ m} \times 2.5 \text{ m}$. There are five velocity levels from 0 m/s to 4 m/s . Each time step has duration $\Delta t = 0.25 \text{ s}$. Vehicle R has four actions: ACCELERATE (1 m/s^2), MAINTAIN (0 m/s^2), DECELERATE (-1 m/s^2), EMERGENCYSTOP (-3 m/s^2). The actions are noisy with probability 0.05 of failing to achieve the intended outcomes. We use probabilistic transition to compensate for the coarse discretization by matching the *expected* distance of travel per time step. For example, if the vehicle's speed is 1 m/s , it then travels a distance of 0.25 m per time step. Let x and x' be two adjacent grid cells along the vehicle's path. We set $p(x'|x) = 0.1$ and $p(x|x) = 0.9$ so that the expected distance that vehicle travels is also $2.5 \times 0.1 + 0 \times 0.9 = 0.25 \text{ m}$ in our model. Vehicle R receives observations on its own state and vehicle A 's state, but the observations may be noisy. It gets a reward for crossing the intersection safely. It gets a penalty for time delay and a high penalty for collision with A .

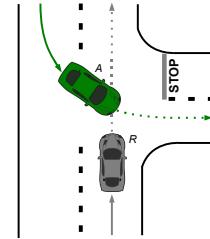


Fig. 6 A near-miss accident during the 2007 DARPA Urban Challenge

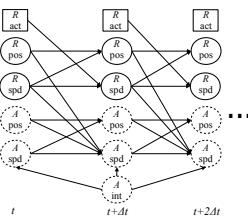


Fig. 7 A MOMDP model for vehicle R to recognize the intention of vehicle A and navigate safely through the intersection. Observation variables are omitted to avoid clutter.

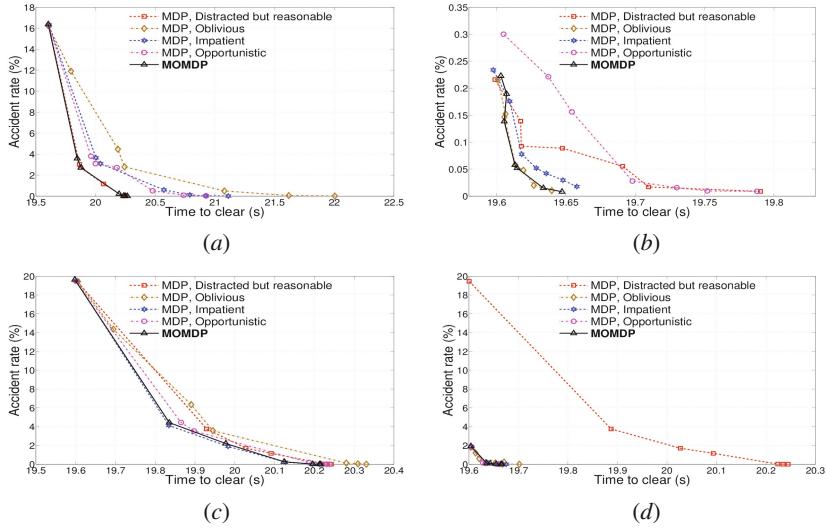


Fig. 8 Performance of the robot vehicle when faced with another vehicle that is (a) reasonable but distracted, (b) oblivious, (c) impatient, and (d) opportunistic but rational

By adjusting reward and penalty values, we can trade-off accident rate and time to clear the intersection, an additional practical advantage of the MOMDP formulation.

Vehicle A has a motion model similar to R 's. Its intention is unknown to R . We assume that A exhibits one of four driving behaviors, each reflecting an intention:

- *Reasonable but distracted.* This driver usually slows and stops before the intersection, but with probability 0.1, he may not stop.
- *Oblivious.* This driver increases his speed to 2 m/s and maintains it, totally ignoring the presence of other vehicles.
- *Impatient.* This driver seeks to cross the intersection as fast as possible. He reacts to the speed of vehicle R , increasing his speed if R slows down and vice versa. He never comes to a complete stop at the intersection.
- *Opportunistic.* Similar to the impatient driver, this driver increases his speed if R slows down and vice versa. However, he will come to a complete stop at the intersection to avoid a collision.

Although not exhaustive, the list includes some common behaviors encountered in intersection navigation, and more can be added if desired.

We built several MOMDP models by varying the reward function, in order to trade off accident rate and time to clear the intersection. We evaluated the resulting policies in SUMO, an established open-source package for microscopic road traffic simulation [3]. For comparison, we also computed and evaluated four MDP policies, which assume that the intended behavior of vehicle A is known. The MDP policies must perform better than our MOMDP policies, as they know A 's intention in advance. The results are shown in Figs. 8 and 9, which plot the performance of

vehicle R measured in accident rate versus time to clear the intersection. Each data point was obtained from 10,000 simulation runs.

The results in Fig. 8 assume that vehicle R has perfect observations. Each plot shows the performance of the MOMDP and MDP policies, when vehicle A has a particular driving behavior. The plots show that the MOMDP policy consistently performs almost as well as the MDP policy that knows the true underlying driving behavior. The performance gap between the MOMDP and MDP policies is very small. In contrast, the MDP policies with the wrong assumption of vehicle A 's behavior usually perform poorly. This suggests that identifying A 's underlying behavior is crucial and the MOMDP policies do so successfully.

We performed further testing by adding noise to the observations. The noise causes the position of vehicle A to be observed at the adjoining grid cells with probability 0.3 rather than its actual cell. Due to space limitation, Fig. 9 shows the results for two of the four driving behaviors only. Observation noise causes moderate performance degradation of the MOMDP policies. This is expected, because with observation noise, it is more difficult to infer vehicle A 's true behavior. However, the performance gaps between the MOMDP policy and the best POMDP policy is still relatively small.

5 Conclusion

This paper introduced the intention of an interacting agent into motion planning. We treat intention-aware motion planning as planning under uncertainty and model it as a MOMDP. The MOMDP is a structured variant of the more common POMDP. It is a rich probabilistic model that can accommodate not only uncertainty in the intention of an interacting agent but also uncertainty in robot control and sensing. It provides a principled general approach to planning under uncertainty and optimally balances information-gathering and task completion actions. By separating the fully and partially observable variables through a factored model, the MOMDP significantly improves computational efficiency for policy computation under suitable conditions, making it a practical tool for modeling interesting robotic tasks.

Our current model assumes that the interacting agent's intention is fixed and does not change over time. To allow changing intentions, we can generalize our MOMDP model by adding a new component for the intention dynamics.

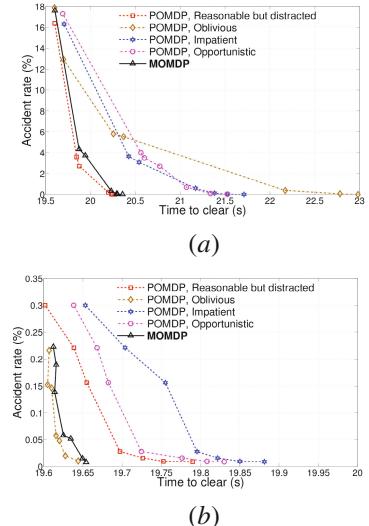


Fig. 9 Performance of the robot vehicle with noisy observations, when faced with another vehicle that is (a) reasonable but distracted and (b) oblivious

SARSOP, the MOMDP solver used in our implementation, assumes a discrete state. We are investigating the possibility of using a continuous POMDP solver [2] to remove this modeling restriction.

Acknowledgments. We thank Leslie Kaelbling and Tomás Lozano-Pérez from MIT for many insightful discussions on POMDPs. This work is supported in part by SMART IRG grant R-252-000-447-592, MoE AcRF grant 2010-T2-2-071, and MDA GAMBIT grant R-252-000-398-490.

References

1. Aoude, G.S., Luders, B.D., Levine, D.S., How, J.P.: Threat-aware path planning in uncertain urban environments. In: Proc. IEEE/RSJ Int. Conf. on Intelligent Robots & Systems (2010)
2. Bai, H., Hsu, D., Lee, W.S., Ngo, V.A.: Monte Carlo Value Iteration for Continuous-State POMDPs. In: Hsu, D., Isler, V., Latombe, J.-C., Lin, M.C. (eds.) Algorithmic Foundations of Robotics IX. STAR, vol. 68, pp. 175–191. Springer, Heidelberg (2010)
3. Behrisch, M., Bieker, L., Erdmann, J., Krajzewicz, D.: Sumo – simulation of urban mobility: An overview. In: Proc. Int. Conf. on Advances in System Simulation, pp. 63–68 (2011)
4. Bennewitz, M., Burgard, W.: Adapting navigation strategies using motion patterns of people. In: Proc. IEEE Int. Conf. on Robotics & Automation (2003)
5. Chong, Z.J., et al.: Autonomous personal vehicle in crowded campus environments. In: Proc. IEEE/RSJ Int. Conf. on Intelligent Robots & Systems, Workshop on Perception and Navigation for Autonomous Vehicles in Human Environment (2011)
6. Fern, A., Tadepalli, P.: A computational decision theory for interactive assistants. In: Advances in Neural Information Processing Systems, NIPS (2010)
7. Fulgenzi, C., Tay, C., Spalanzani, A., Laugier, C.: Probabilistic navigation in dynamic environment using rapidly-exploring random trees and Gaussian processes. In: Proc. IEEE/RSJ Int. Conf. on Intelligent Robots & Systems (2008)
8. Helbing, D., Buzna, L., Johansson, A., Werner, T.: Self-organized pedestrian crowd dynamics and design solutions: Experiments, simulations and design solutions. Transportation Science 39(1), 1–24 (2005)
9. 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)
10. Kautz, H., Allen, J.F.: Generalized plan recognition. In: Proc. AAAI Conf. on Artificial Intelligence, vol. 19, p. 86 (1986)
11. Khatib, O.: Real-time obstacle avoidance for manipulators and mobile robots. Int. J. Robotics Research 5(1), 90–98 (1986)
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. Latombe, J.C.: Robot Motion Planning. Kluwer Academic Publishers, Boston (1991)
14. Leonard, J., et al.: A perception driven autonomous urban vehicle. J. Field Robotics 25(10), 727–774 (2008)
15. Ong, S.C.W., Png, S.W., Hsu, D., Lee, W.S.: Planning under uncertainty for robotic tasks with mixed observability. Int. J. Robotics Research 29(8), 1053–1068 (2010)

16. Papadimitriou, C., Tsiplakis, J.N.: The complexity of Markov decision processes. *Mathematics of Operations Research* 12(3), 441–450 (1987)
17. 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)
18. Russell, S., Norvig, P.: *Artificial Intelligence: A Modern Approach*. Prentice Hall (2003)
19. Smallwood, R.D., Sondik, E.J.: The optimal control of partially observable Markov processes over a finite horizon. *Operations Research* 21, 1071–1088 (1973)
20. Smith, T., Simmons, R.: Point-based POMDP algorithms: Improved analysis and implementation. In: Proc. Uncertainty in Artificial Intelligence (2005)
21. Turaga, P., Chellappa, R., Subrahmanian, V.S., Udrea, O.: Machine recognition of human activities: A survey. *IEEE Trans. on Circuits & Systems for Video Technology* 18(11), 1473–1488 (2008)

Point-Based Policy Transformation: Adapting Policy to Changing POMDP Models

Hanna Kurniawati and Nicholas M. Patrikalakis

Abstract. Motion planning under uncertainty that can efficiently take into account changes in the environment is critical for robots to operate reliably in our living spaces. Partially Observable Markov Decision Process (POMDP) provides a systematic and general framework for motion planning under uncertainty. Point-based POMDP has advanced POMDP planning tremendously over the past few years, enabling POMDP planning to be practical for many simple to moderately difficult robotics problems. However, when environmental changes alter the POMDP model, most existing POMDP planners recompute the solution from scratch, often wasting significant computational resources that have been spent for solving the original problem. In this paper, we propose a novel algorithm, called *Point-Based Policy Transformation (PBPT)*, that solves the altered POMDP problem by *transforming* the solution of the original problem to accommodate changes in the problem. PBPT uses the point-based POMDP approach. It transforms the original solution by modifying the set of sampled beliefs that represents the belief space B , and then uses this new set of sampled beliefs to revise the original solution. Preliminary results indicate that PBPT generates a good policy for the altered POMDP model in a matter of minutes, while recomputing the policy using the fastest offline POMDP planner today fails to find a policy with similar quality after two hours of planning time, even when the policy for the original problem is reused as an initial policy.

1 Introduction

Motion planning under uncertainty that can efficiently take into account changes in the environment is critical for robots to operate reliably in our living spaces.

Hanna Kurniawati
School of Information Technology & Electrical Engineering,
University of Queensland
e-mail: hannakur@uq.edu.au

Nicholas M. Patrikalakis
Department of Mechanical Engineering, Center for Ocean Engineering,
Massachusetts Institute of Technology
e-mail: nmp@mit.edu

Changes in our living spaces are unavoidable. New furnitures are added, water current changes direction, etc. Despite these changes, robots need to reliably accomplish the given tasks in the midst of various control and sensing errors. For example, imagine an Autonomous Underwater Vehicle (AUV) navigating around highly cluttered offshore oil platforms. Water currents that highly accentuates the AUV's control error, changes throughout the day. Many of these changes are very significant, e.g., a 180^0 changes in the currents direction, making an optimal motion strategy at a particular time may actually be a bad strategy when the AUV operates at a different time. Despite the criticality of these changes, most changes are often limited to localized regions. In this preliminary work, we are interested in local changes and assume that the changes are known prior to execution.

To handle uncertainty, PBPT uses the Partially Observable Markov Decision Process (POMDP), which is a systematic and general framework for motion planning under uncertainty. Due to uncertainty, a robot never knows its exact state. Therefore, a POMDP planner computes the best action to perform with respect to a set of states that are consistent with the available information so far. Each set of states is represented as a distribution over the state space, called a belief, and the set of all beliefs is called the belief space B . The action to perform is encoded as a mapping from beliefs to actions, called a policy. Once generated, the optimal policy can be used as a feedback controller for the robot. It is true that solving a POMDP exactly is computationally intractable [15]. However by trading optimality with approximate optimality for speed, point-based POMDP approach [16] has tremendously sped-up POMDP planning, enabling it to be practical for many simple to moderately difficult robotics problems [11, 22].

Now, when the environment changes, the POMDP model changes too. Methods that can be used to handle changes in the POMDP model can be classified into two extreme approaches. First is replanning. Off-line replanning that recomputes the policy from scratch, using existing off-line POMDP planners, may waste significant computation time, especially when the changes are small but significant, and the problem is difficult such that even the fastest offline POMDP solvers today require hours to solve it. Recent replanning methods, such as [7, 8, 17, 21], are fast, as they are designed for on-line planning. But, these methods do not perform global planning in the belief space. Hence, they may cause the robot to fall into a catastrophic state when such state exists. This is undesirable for highly critical tasks. The second approach hedges over all possible changes [20]. It models all possible changes again as a POMDP problem. Each parameter that may change is modeled as a state variable of an enlarged POMDP problem. The enlarged POMDP problem can be solved using existing POMDP planners and the generated policy is optimal over all possible changes in the POMDP model. But, the enlarged problem can quickly become very large, beyond the capability of even the best POMDP solver today.

In this paper, we propose a novel algorithm, called *Point-Based Policy Transformation (PBPT)*, that transform a pre-computed POMDP policy according to changes in the POMDP model. By doing so, PBPT does not enlarge the POMDP problem that needs to be solved, and hence is faster than the hedging approach. Although slower than on-line replanning strategies. PBPT performs global planning in a restricted

part of B and finds the best policy in a restricted class of policies with high probability, making it more suitable than on-line replanning for highly critical tasks.

PBPT uses the point-based POMDP approach. Key to point-based approach is that it represents the belief space B using a representative set of sampled beliefs. And then plans, i.e., performs Bellman updates, on only this set of beliefs, instead of the entire B . Now, when the POMDP model changes, the representative set of beliefs is likely to change too. PBPT uses the difference between the new and the original POMDP models to transform the representative set of beliefs. It identifies beliefs that are affected by the changes and are unlikely to be part of a representative belief set in the new problem. It replaces such beliefs by re-sampling B . Once the set of representative beliefs is fixed, PBPT revises the policy by performing Bellman updates at selected beliefs to avoid unnecessary update operation.

Suppose a policy class Π is the set of all possible policies for the new POMDP model, where the mapping from unaffected beliefs is the same as the mapping from these beliefs in the original policy. Then, PBPT converges to a good approximation of the best policy in Π or better, with high probability. Furthermore, preliminary results indicate that PBPT generates a good policy for the revised POMDP model in a matter of minutes, while recomputing the policy using the fastest offline POMDP planner today fails to find a policy with similar quality after two hours of planning time, even when the original policy is reused as an initial policy.

2 Background and Related Work

2.1 POMDP Background

A POMDP is specified as a tuple $\langle S, A, O, T, Z, R, b_0, \gamma \rangle$, where S is the set of states, A is the set of actions, and O is the set of observations. In each step, the agent is in a state $s \in S$, takes an action $a \in A$, and moves from s to an end state s' . Due to the uncertainty in action, the end state s' is modeled as a conditional probability density function $T(s, a, s') = f(s'|s, a)$. The agent may then receive an observation. Due to the uncertainty in observation, the observation result $o \in O$ is again modeled as a conditional probability density function $Z(s', a, o) = f(o|s', a)$. In each step, the agent receives a reward $R(s, a)$, if it takes action a from state s . 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 specify a discount factor $\gamma \in (0, 1)$ so that the total reward is finite and the problem is well defined.

A POMDP planner computes an *optimal policy* that maximizes the agent's expected total reward. A POMDP policy $\pi: B \rightarrow A$ prescribes an action a , given the agent's belief b . A policy π induces a value function $V(b, \pi)$ which specifies the expected total reward of executing policy π , and is computed as

$$V(b, \pi) = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) | b, \pi\right] \quad (1)$$

A policy can be represented by various representations, e.g., policy-graph [2], α -function [18], or pairs of belief and action [25] for continuous state space.

To execute a policy π , an agent executes action selection and belief update repeatedly. For example, if the agent's current belief is b , it selects the action referred to by $a = \pi(b)$. After the agent performs action a and receives an observation o according to the observation function Z , it updates b to a new belief b' given by

$$b'(s') = \tau(b, a, o) = \eta Z(s', a, o) \int_{s \in S} T(s, s, s') ds \quad (2)$$

where η is a normalization constant.

2.2 Related Work

Many motion planners have been proposed to handle changes in the environment, e.g., [4, 9, 14, 23]. However, they do not consider partial observability of the system. The work in [13] considers a limited partial observability property, i.e., partial predictability of the environment, but they do not take into account errors in the robot's control and sensing.

POMDP is a systematic and general approach for planning under uncertainty. Although solving a POMDP exactly is computationally intractable [15], in the past few years, different approaches have been proposed to make POMDP planning more practical. Several work restricts the beliefs to be Gaussian, e.g., [3, 19].

Point-based POMDP does not restrict its distribution and has tremendously advanced POMDP planning. It reduces the complexity of planning in B by representing B as a set of sampled beliefs and planning with respect to this set only. To generate a policy, most point-based POMDPs use value iteration, utilizing the fact that the optimal value function satisfies Bellman equation. They start from an initial policy, represented as a value function V . And iteratively perform Bellman backup on V at the sampled beliefs, i.e., $V(b) = \max_{a \in A} (R(b, a) + \gamma \sum_{o \in O} \tau(b, a, o) \hat{V}^*(\tau(b, a, o)))$ where $\hat{V}^*(b')$ is the current best value of b' . The iteration is performed until it converges. Over the past few years, impressive progress have been gained by improving the strategy for sampling B [11, 16, 22]. Although these planners were designed for discrete state space, they can be extended to handle continuous state space, by replacing their policy representation and backup operation with policy representation and backup operation designed for continuous state space, e.g., [2, 18, 25]. However, most work in point-based POMDP does not handle changes in the model. We propose a point-based POMDP planner to handle changes in the POMDP model, where the changes are known prior to execution, by modifying a pre-computed optimal policy of the original problem.

The idea of policy modification resembles path deformation, e.g., [12]. But, most work in path deformation do not consider partially observable systems.

Modifying an optimal policy of one POMDP problem to solve another POMDP problem, can be considered as a transfer learning problem. However, most work in transfer learning focus on finding the mapping from one problem to another or from one solution to another [24]. We propose an algorithm to identify parts of the policy that needs to be modified and how to modify the policy, based on the differences between two POMDP models.

3 Overview

3.1 PBPT's Goal

Before defining PBPT's goal more formally, let's first discuss the model changes and their effect on beliefs more formally. Suppose $P_o = \langle S, A, O, T_o, Z_o, R_o, b_0, \gamma \rangle$ is the original POMDP model. PBPT assumes that the state space S is a metric space, and can be continuous or discrete. It assumes that the action space A and observation space O are discrete. Suppose to accommodate environmental changes, the original model P_o is revised to a new POMDP model $P_n = \langle S, A, O, T_n, Z_n, R_n, b_0, \gamma \rangle$. PBPT can handle any changes in the POMDP model, as long as there is a bijection from the state, action, and observation spaces in the original model to their corresponding spaces in the revised model. To simplify notation, here we assume that the bijection is an identity map and use the same notation for these spaces in P_o and P_n . Furthermore, we assume that the initial belief b_0 does not change.

Changes from P_o to P_n affect a subset of S , which we denote as $S_{ch}(P_o, P_n) \subseteq S$ and define as,

$$\begin{aligned} S_{ch}(P_o, P_n) = \{s \in S \mid & \exists_{s' \in S, a \in A} T_o(s, a, s') \neq T_n(s, a, s') \vee \\ & \exists_{a \in A, o \in O} Z_o(s, a, o) \neq Z_n(s, a, o) \vee \exists_{a \in A} R_o(s, a) \neq R_n(s, a)\}. \end{aligned}$$

For writing compactness, we use S_{ch} to refer to $S_{ch}(P_o, P_n)$. PBPT assumes that S_{ch} consists of one or more connected components, where each connected component is a closed set, forming a simple polytope.

Now, we can use the notion of affected states to discuss affected beliefs. Let π_o^* be the optimal policy that has been pre-computed for P_o . Key to PBPT in revising a policy is to first revise the set of representative beliefs. The set of representative beliefs is sampled from the set $\mathcal{R}_o^*(b_0)$ of beliefs that are reachable from the initial belief $b_0 \in B$ under π_o^* in P_o . For writing compactness, we use \mathcal{R}_o^* to refer to $\mathcal{R}_o^*(b_0)$. To identify which beliefs in the original set of representative beliefs need to be replaced, we classify \mathcal{R}_o^* into three classes, i.e., *directly affected*, *indirectly affected*, and *unaffected*.

A belief $b \in \mathcal{R}_o^*$ is in the *directly affected* class, denoted as B_{ch} , whenever its support intersects S_{ch} . A belief $b \in \mathcal{R}_o^*$ is in the *indirectly affected* class, denoted as B'_{ch} , whenever its support does not intersect S_{ch} but at least one of the beliefs reachable from b under π_o^* is directly affected. More formally, the set of indirectly affected beliefs is $B'_{ch} = \{b \in \mathcal{R}_o^* \mid \text{support}(b) \cap S_{ch} = \emptyset \wedge \text{support}(\mathcal{R}_o^*(b)) \cap S_{ch} \neq \emptyset\}$ where $\text{support}(\mathcal{R}_o^*(b)) = \bigcup_{b' \in \mathcal{R}_o^*(b)} \text{support}(b')$. Beliefs in $\mathcal{R}_o^*(b_0)$ that are not directly affected nor indirectly affected belong to the *unaffected* class, denoted as B_u .

This classification reflects the effect of model changes on the value function, too. Let $V_o(b, \pi_o^*)$ be the value of executing π_o^* from b in P_o , and $V_n(b, \pi_o^*)$ be the value of executing π_o^* from b in P_n . Then, the relation can be defined more formally as,

Lemma 1. Suppose $P_o = \langle S, A, O, T_o, Z_o, R_o, b_0, \gamma \rangle$ changes to $P_n = \langle S, A, O, T_n, Z_n, R_n, b_0, \gamma \rangle$. For any $b \in B$, if $\text{support}(b) \cap S_{ch} = \emptyset$ and $\text{support}(\mathcal{R}_o^*(b)) \cap S_{ch} = \emptyset$, then $V_n(b, \pi_o^*) = V_o(b, \pi_o^*)$.

Proof. The proof is straightforward by using induction on the planning horizon. For the base case (planning horizon 1), $V_n(b, \pi_o^*) = V_o(b, \pi_o^*)$ because the reward functions of performing any action from any state in $\text{support}(b)$ remain the same. Let's assume that $V_n(b, \pi_o^*) = V_o(b, \pi_o^*)$ for planning horizon h . For planning horizon $h+1$, $V_n(b, \pi_o^*) = R(b, \pi_o^*(b)) + \gamma \sum_{o \in O} \tau(b, a, o) V(\tau(b, a, o)) do$ where $V(\tau(b, a, o))$ is the value of $\tau(b, a, o)$ computed using h planning horizon. Since b is not affected by any changes, $R(b, \pi_o^*(b))$ and $\tau(b, a, o)$ for any observation $o \in O$ remain the same. Using the assumption on planning horizon h , $V_n(b, \pi_o^*) = V_o(b, \pi_o^*)$. \square

Using the above classification, we can define PBPT's goal more formally. Suppose $\Pi(\pi_o^*, P_o, P_n) = \left\{ \pi \in \Pi_n \mid \forall b \in B \setminus (B_{ch} \cup B'_{ch}) \pi(b) = \pi_o^*(b) \right\}$, where Π_n is the set of all possible POMDP policies for P_n . Then, PBPT's goal is to find the best policy, i.e., $\pi_n^* = \arg \max_{\pi \in \Pi(\pi_o^*, P_o, P_n)} V_n(b_0, \pi)$, from the policy class $\Pi(\pi_o^*, P_o, P_n)$.

3.2 Overview of the Algorithm

Algorithm 1. Point-Based Policy Transformation (π_o^*, P_o, P_n)

- 1: Initialize T with a sampled representation of \mathcal{R}_o^* .
 - 2: Classify the nodes of T into indirectly affected, directly affected, and unaffected nodes.
 - 3: π_n = Initialize with π_o^* .
 - 4: **while** Termination condition not satisfied **do**
 - 5: Sample an indirectly affected node b from T .
 - 6: Let $(b_0, b_1, \dots, b_n, b)$ be the path from b_0 to b in T .
 - 7: NodeStackToBeBackup.clear();
 - 8: **for** $i = 0$ to n **do**
 - 9: NodeStackToBeBackup.push(b_i).
 - 10: NodeStackToBeBackup.push(b).
 - 11: **while** b is not unaffected AND expansion still likely to improve $V_n(b_0, \pi_n)$ **do**
 - 12: Select an action $a \in A$ and an observation $o \in O$.
 - 13: Let $b' = \tau(b, a, o)$.
 - 14: Insert b' as a child of b in T .
 - 15: NodeStackToBeBackup.push(b').
 - 16: $b = b'$.
 - 17: **while** NodeStackToBeBackup is not empty **do**
 - 18: $b = \text{NodeStackToBeBackup.pop}()$
 - 19: Backup(b, π_n).
-

Key to PBPT is to transform a policy by revising the set of sampled beliefs that represents B . Given an optimal policy π_o^* for the original problem P_o , PBPT starts by constructing a representative set of sampled beliefs for P_o (line 1 of Algorithm 1). To this end, PBPT samples the set \mathcal{R}_o^* of beliefs reachable under π_o^* in P_o and represents them as a belief tree T . Each node of T represents a sampled belief and the root is b_0 . PBPT represents each belief as unweighted particles. In this paper, we refer to the nodes of T and their corresponding beliefs interchangeably, and use the same

notation. Each edge bb' represents a pair of action–observation $a\text{--}o$ where $a \in A$ and $o \in O$, such that $b' = \tau(b, a, o)$. Details on how PBPT samples \mathcal{R}_o^* are in Section 4.1.

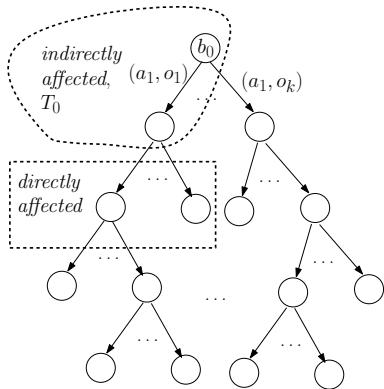


Fig. 1 The initial T . All out-edges of the same node would have the same action label, as the initial T is a sampled representation of \mathcal{R}_o^* . The nodes outside the rectangle and bounding curve are unaffected beliefs.

vance, as knowing this is the same as knowing π_n^* . But we can use the observation that, even when the changes from P_o to P_n decrease the reward of all states in S_{ch} by much, an indirectly affected belief b may still have a high value, and hence lies in \mathcal{R}_n^* , if after reaching b , there is a strategy for the robot to maneuver, to avoid reaching beliefs whose supports intersect S_{ch} . Utilizing this observation, PBPT selects an indirectly affected node to start further expansion of T . Details on how to choose an indirectly affected node for expansion and how a deep sampling is performed are in Section 4.4.

When an unaffected belief is sampled, a deep sampling is stopped (line 11 of Algorithm 1). By doing so, PBPT avoids recomputing the policy at beliefs where the mapping from the original policy does not need to be revised.

To approximate the set B_u of unaffected beliefs, PBPT constructs an end-game region around each unaffected node of T . An end-game region is the set of points that lie within a threshold distance away from an unaffected node of T . When a newly sampled belief falls inside the end-game region, the deep sampling is terminated, as we can and will reuse the mapping from π_o^* . The details on end-game region construction are discussed in Section 4.3.

To revise the policy, PBPT performs repeated backup operations starting from the newly inserted nodes all the way to the root (line 17–19 of Algorithm 1). By doing so, PBPT avoids unnecessary backup operations.

Suppose P_n is the new problem. PBPT uses the difference between P_o and P_n to classify the nodes in T into the directly affected, indirectly affected, and unaffected nodes (Figure 1), as discussed in Section 3.1 (line 2 of Algorithm 1). Details on how PBPT classifies the nodes are in Section 4.2.

Once the classification is done, PBPT revises T by selecting an indirectly affected node to be expanded (line 5 of Algorithm 1), and performs deep sampling from the selected node (line 11–16 of Algorithm 1). Suppose \mathcal{R}_n^* is the set of beliefs reachable from b_0 under an optimal policy π_n^* for the new problem P_n . Then to revise T , ideally we want to replace nodes of T that are not in \mathcal{R}_n^* with beliefs that lie in \mathcal{R}_n^* , and for efficiency, we want to replace the smallest number of nodes possible. Of course, we do not know \mathcal{R}_n^* in ad-

4 Point-Based Policy Transformation

4.1 Initializing the Belief Tree T

To initialize the belief tree T using a sampled representation of \mathcal{R}_o^* , PBPT simulates executing π_o^* from b_0 in P_o multiple times. The required number of simulation runs is discussed in Section 5. Remember that due to uncertainty, the robot does not know the actual state it visited. Therefore, in each simulation run, PBPT maintains two traces. One is a state trace, representing the sequence of states visited by the robot. Another is a belief trace, representing the sequence of beliefs that estimates the sequence of states visited by the robot.

To start a simulation run, PBPT samples a state $s_0 \in S$ from b_0 , and sets s_0 as the actual starting state of the robot. Then, it simulates the robot performing action $\pi_o^*(b_0)$. To simulate the action, PBPT samples the robot's next state $s \in S$ from $T(s_0, \pi_o^*(b_0), s)$, samples an observation $o \in O$ perceived by the robot from $Z(s, \pi_o^*(b_0), o)$, assigns the reward $R(s_0, \pi_o^*(b_0))$ to the robot, and computes the robot's next belief as $b = \tau(b_0, \pi_o^*(b_0), o)$. If the belief b is not yet a child of b_0 in T , PBPT inserts b as a child of b_0 . Regardless of whether b is just inserted or is already available, the sampled state s is added to $S_B(b)$, a set of states that represent b 's support. Next, PBPT continues the simulation and insertion process iteratively from b and s . This process is iteratively repeated for subsequent beliefs and states until a large number of simulation steps is performed.

4.2 Classifying the Nodes of T

To identify nodes of T that are directly affected, PBPT finds the nodes whose support sets intersect S_{ch} . The main consideration here is the ability to handle many different changes to the original POMDP model P_o , efficiently. For this purpose, PBPT takes the union $\bigcup_{b \in T} S_B(b)$ of the support states, and structures them in a range tree data structure. PBPT also computes the bounding hyper-rectangle of each connected subset of the affected state space regions S_{ch} . For each bounding hyper-rectangle, PBPT queries the range tree. Then, it tests each state s in the query result whether s is in the connected subset bounded by the hyper-rectangle. For each s in S_{ch} , PBPT sets each belief b of T that has s in its $S_B(b)$ as a directly affected belief.

Once all directly affected nodes in T have been identified, the indirectly affected nodes are then all ancestors of each directly affected node that are not themselves directly affected. Nodes that are not directly nor indirectly affected are identified as unaffected nodes.

The above strategy requires additional time to construct the range tree, but once the range tree is constructed, finding directly affected nodes of T can be done fast. Suppose T contains N nodes, and the set $S_B(b)$ of each node b in T has size m . To construct the range tree structure, PBPT requires $O(Nm \log^{d-1}(Nm))$ time [5], where d is the dimension of S . Now, suppose we are given S_{ch} that consists of M connected subset. For each subset, PBPT constructs a bounding hyper-rectangle. The time to query the states that lie inside a hyper-rectangle dominates the time for

testing if a state s in a query result lies in S_{ch} and the time for finding the nodes of T that has s in its support. Therefore, the total time to find directly affected nodes is $O(M(\log^d(Nm) + k))$ where k is the largest number of query result.

4.3 Constructing the End-Game Region

Recall that the end-game region is the set of beliefs within a given threshold distance from at least one of the unaffected nodes of T . Before discussing the distance threshold, let's first discuss the metric we use.

To compute distance in B , PBPT uses the Earth Mover's Distance (EMD). EMD between two beliefs computes the minimum expected state space distance, where the minimum is taken over all possible joint distribution whose marginals are the two beliefs. More precisely,

$$D_B(b, b') = \inf_f \left\{ \int_{s \in S} \int_{s' \in S} D_S(s, s') f(s, s') ds ds' \mid b = \int_{s'} f(s, s') ds', b' = \int_s f(s, s') ds \right\}$$

where $D_B(b, b')$ is the EMD between b and b' in B , $D_S(s, s')$ is the distance between s and s' in S , and f is a joint density function.

Most point-based POMDP solvers do not use EMD, instead they use L1 distance as metric in B . The main difference between the two metrics is that EMD is dependent on the underlying state space metric, while L1 is not. This difference makes EMD a better indication of the value difference between two beliefs compared to L1, when the state space is a metric space and the behavior of the system at nearby states are similar, which is often the case in robot motion planning. The value function of a belief (eq. (1)) is computed based on two components, the belief and the expected total discounted reward. EMD incorporates both components when the reward function is Lipschitz continuous, while L1 can only incorporate one of them.

Using EMD in end-game region construction requires that nearby beliefs, under EMD, would have similar optimal value, such that with a small error, one belief can be used to represent other nearby beliefs. This requirement is indeed true, as the value function of beliefs in B with EMD satisfies Lipschitz condition, i.e.,

Theorem 1. Suppose (S, D_S) and (B, D_B) are metric spaces, and for any state $s, s' \in S$ and any action $a \in A$, $|R(s, a) - R(s', a)| \leq CD_S(s, s')$. If $D_B(b, b') \leq \delta$, then $|V^*(b) - V^*(b')| \leq C\delta$ for any belief $b, b' \in B$.

Proof. Let's first define $D'_S(s, s') = CD_S(s, s')$. When the metric in S is D'_S , the EMD of B becomes $D'_B(b, b') = CD_B(b, b')$.

Wlog, suppose the optimal policy is represented by a set Γ of α -functions. Let $Q_b(s, a) = R(s, a) + \gamma \int_{s' \in S} T(s, a, s') \sum_{o \in O} Z(s', a, o) \alpha_{b,a,o}^*(s') ds'$ where $\alpha_{b,a,o}^* = \arg \max_{\alpha \in \Gamma} \int_{s \in S} \alpha(s) \cdot \tau(b, a, o)(s) ds$. Then, the optimal value of $b \in B$ can be written as $V^*(b) = \max_{a \in A} (\int_{s \in S} Q_b(s, a) \cdot b(s) ds)$ and the value difference, is

$$\begin{aligned} |V^*(b) - V^*(b')| &= \max_{a \in A} \left(\int_{s \in S} Q_b(s, a) \cdot b(s) ds \right) - \max_{a \in A} \left(\int_{s \in S} Q_{b'}(s, a) \cdot b'(s) ds \right) \\ &\leq \max_{a \in A} \left(\int_{s \in S} Q_b(s, a) \cdot b(s) ds - \int_{s \in S} Q_b(s, a) \cdot b'(s) ds \right) \end{aligned} \quad (3)$$

We can bound $Q_b(s, a)$ from above, such that $Q_b(s, a) \leq R(s, a) + \gamma^{\frac{R_{max}}{1-\gamma}}$. Using this bound in eq. (3) gives us

$$|V^*(b) - V^*(b')| \leq \max_{a \in A} \left(\int_{s \in S} R(s, a) \cdot b(s) ds - \int_{s \in S} R(s, a) \cdot b'(s) ds \right) \quad (4)$$

Using the Kantorovich distance [6], i.e., $K(b, b') = \sup_{g \in Lip_1} (\int_{s \in S} g(s) b(s) ds - \int_{s \in S} g(s) b'(s) ds)$ where Lip_1 is the set of all 1-Lipschitz functions over S , we can bound eq. (4). Using metric D'_S , the reward function R satisfies 1-Lipschitz condition for any action $a \in A$, and so we have $|V^*(b) - V^*(b')| \leq K(b, b')$.

Since the above Kantorovich distance is the dual of EMD D'_B [6], $|V^*(b) - V^*(b')| \leq K(b, b') = D'_B(b, b') = CD_B(b, b')$. \square .

In our prior work [10], we have shown Lipschitz condition for value function in B with metric EMD. But, the requirement in the above theorem is much more relaxed than the one in our prior work.

Since EMD is based on the metric in S , we can use heuristics on S to set the threshold distance for constructing end-game region in B . Constructing reasonable heuristics in S is much easier than in B . One heuristics that can be used is based on the intuition that the maximum expected future rewards are similar when the POMDP agent starts from nearby states where the transition and reward functions are the same. Using this heuristics, the threshold distance for a belief b is the maximum between the expected nearest distance to states with different transition and reward function and the expected threshold distance in S .

4.4 Sampling New Beliefs

To select which node of T to start deep sampling from, we want to choose a combination of starting nodes that improves the value of b_0 the most in the least time possible. Of course, we do not know which combination of nodes would generate such improvement. Therefore, PBPT uses Exp3.S, an adaptive selection strategy based on sampling history that is guaranteed to be close to the best combination [1].

Suppose $T_0 = \{b_1, b_2, \dots, b_K\}$. Using Exp3.S, at iteration- t PBPT selects a node $b_i \in T_0$ with probability

$$p_t(b_i) = (1 - \beta) \frac{w_t(b_i)}{\sum_{j=1}^K w_t(b_j)} + \frac{\beta}{K} \quad (5)$$

where $w_t(b_i)$ is a weight that reflects how much starting deep sampling from b_i has improved the value of b_0 in the past and $\beta \in (0, 1)$ is a fixed constant. Once a node b_i is sampled, PBPT starts a deep sampling from b_i . After the deep sampling is terminated, PBPT backups the nodes in the path traversed during deep sampling, starting from the last node inserted to T all the way to the initial belief b_0 . Next, PBPT updates the probability in eq. (5) according to how much the value of b_0 improves. It updates the weight as,

$$w_{t+1}(b_i) = w_t(b_i) e^{\beta \frac{x_t(b_i)}{K}} + \frac{e}{MK} \sum_{i=1}^K w_t(b_i)$$

where $x_t(b_i) = \frac{V_2^t(b_0, \pi_2) - V_2^{t-1}(b_0, \pi_2)}{|R_{\max}|/(1-\gamma)}$ if b_i was selected at iteration- t and 0 otherwise, $V_2^t(b_0, \pi_2)$ is $V_2(b_0, \pi_2)$ after iteration- t . Note that due to backup operation, $V_2^t(b_0, \pi_2) \geq V_2^{t-1}(b_0, \pi_2)$, and so $x_t(b_i) \geq 0$. The notation M denotes the total number of deep sampling PBPT performs.

PBPT's deep sampling strategy is similar to SARSOP. SARSOP maintains an upper and lower bounds for each node in T . The upper bound is the expected value assuming deterministic action, while the lower bound is the value of the belief in the current policy. Given b , PBPT chooses an action $a \in A$ with the highest upper bound, and an observation $o \in O$ that is expected to reduce the gap between the upper and lower bound the most. A new belief b' is computed as $\tau(b, a, o)$ and then inserted to T . Then, the expansion continues starting from b' until either the gap between the upper and lower bound becomes too small to cause improvement on gap reduction on the root. PBPT stops the expansion that starts from b when the newly sampled belief is unlikely to reduce the gap at the root, or when the newly sampled belief is in the end-game region. When the newly sampled belief is in the end-game region, the mapping from belief to action from the original policy can be used.

5 Convergence

Now, the question is whether PBPT converges to the best policy in $\Pi(\pi_o^*, P_o, P_n)$. To discuss convergence, we need to consider three main components of PBPT. First, the strategy to exclude nodes in the end-game region from expansion and backup operations. Second, the strategy for selecting a starting node for deep sampling. Last, the strategy for deep sampling.

The main concern in excluding beliefs in the end-game region from expansion and backup operations is that, PBPT may misclassify a sampled belief to be in the end-game region, while it is actually not. Suppose B_e is the set of unaffected nodes in the initial T . To identify if a belief b in the initial T is in B_e or not, PBPT samples a set of state traces (Section 4.1), starting from a state sampled from the support of b . If state traces that intersects S_{ch} exist, but PBPT fails to sample even one of them, b will be wrongly identified as unaffected node, and hence wrongly included in the end-game region. Since PBPT will not improve the value of beliefs in the end-game region, this misclassification may cause PBPT to fail to converge to the best policy in $\Pi(\pi_o^*, P_o, P_n)$.

However, it turns out that with a small number of independently sampled state traces, the effect of misclassification in the end-game region can be kept low.

Theorem 2. Suppose p_i is the probability that a state trace sampled from $\mathcal{R}_o^*(b_i)$ for any b_i in the initial T intersects S_{ch} , and $p_{\min} = \min_{b_i \in B_{ch}} p_i$. Assume that each state trace from a node b in the initial T starts from a state in $\text{support}(b)$ and that the starting state is sampled independently from the same distribution b . If $\mathcal{R}_o^*(b)$ of each belief b in the initial T is represented by $n \geq \frac{\ln(\varepsilon\delta)}{\ln(1-p_{\min})}$ state traces, then $P(\% \text{ nodes in the initial } T \text{ that are misclassified as unaffected} < \varepsilon) \geq (1 - \delta)$, where $\varepsilon, \delta \in (0, 1]$ is a small constant.

Proof. Let's first compute the smallest number n of sampled state traces needed to ensure that $P(b \text{ is misclassified}) \leq m$ for any belief $b \in B_e$ and $m \in [0, 1]$.

Assuming that each state trace starts from a state in $\text{support}(b)$ and is sampled independently from the same distribution b , we can compute n using binomial distribution with success probability p_{\min} . The result is that $n \geq \frac{\ln(m)}{\ln(1-p_{\min})}$ state traces are needed to guarantee that $P(b \text{ is misclassified}) \leq m$.

Now, let X be the random variable that denotes the number of beliefs in T that are misclassified as unaffected nodes. We can represent X as a binomial distribution with success probability at most m , and computes

$$\begin{aligned} P(X < \varepsilon N) &= 1 - P(X \geq \varepsilon N) \\ &\geq 1 - \frac{Nm}{\varepsilon N} \end{aligned}$$

where N is the total number of nodes in the initial T . The last inequality is computed using Markov inequality. To ensure that $P(X < \varepsilon N) \geq (1 - \delta)$, we need $m = \varepsilon \delta$. Inserting this value of m to the lower bound of n that ensures $P(b \text{ is misclassified}) \leq m$ yields the desired result. \square

The above theorem computes the smallest number of traces to guarantee small misclassification in B_e with high probability. But, the end-game region is larger than B_e , as it is the set of all beliefs within a threshold distance from at least one belief in B_e . However, since the value function satisfies Lipschitz condition (Theorem 1), when the threshold distance is small, the difference between $V_2(b, \pi_n^*)$ and $V_2(b', \pi_n^*)$, where b is in the end-game region and $b' \in B_e$ is b 's nearest belief in B_e , is small too. Hence, the effect of misclassified beliefs in the end-game region to PBPT convergence depends mostly on the misclassification in B_e .

Opposite to the above discussion, some beliefs that are supposed to be in the end-game region may not be sampled in the initial T . Hence, these beliefs are not identified as part of the end-game region. When they are sampled during expansion of T , PBPT continues expanding and performing backups from them. As a result, PBPT may converge to a policy that is better than the best policy in $\Pi(\pi_o^*, P_o, P_n)$.

Convergence to better than the best policy in $\Pi(\pi_o^*, P_o, P_n)$ may also happens due to luck. When we use a global policy representation, such as α -functions, the value of beliefs in the end-game region may improve because of the backup operations at some farther beliefs that are not in the end-game region.

The last two components of PBPT do not worsen the convergence results. PBPT deep sampling strategy is the same as SARSOP, which converges to the optimal policy. But unlike SARSOP, PBPT starts sampling from an indirectly affected node in the initial T , instead of from b_0 . Starting deep sampling not from b_0 may cause the sampling to be incomplete, in the sense that it may never cover the beliefs that are critical to generate an optimal policy. However, T_0 always contains b_0 and the strategy PBPT uses to sample a starting node from T_0 always assigns non-zero probability for each belief in T_0 . Hence, PBPT sampling is complete, which means that PBPT strategies in the first two steps does not change the above convergence results.

Since PBPT fails to converge only when there are misclassified beliefs in the end-game region, Theorem 2 shows that with a small number of state trace samples, PBPT converges to a good approximation of the best policy in $\Pi(\pi_o^*, P_o, P_n)$ or better, with high probability.

6 Experiments

6.1 Scenarios

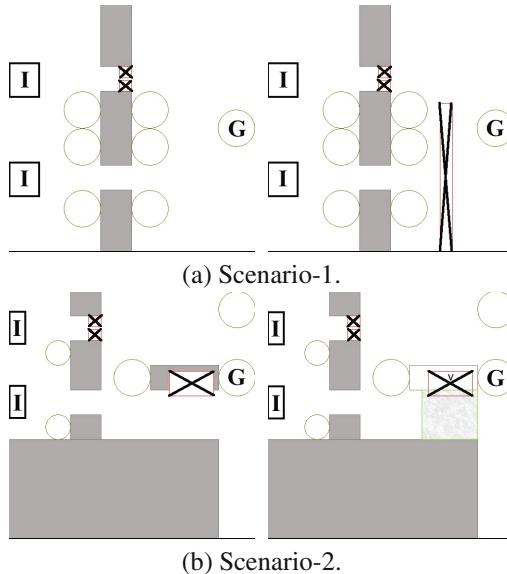


Fig. 2 Scenarios. In each scenario, left: original problem, right: modified problem. The rectangles labeled 'I' is possible initial position of the AUV. The circle labeled 'G' is the goal region. The circle without labels are regions where the AUV can localize well. Rectangles marked with crosses are bounding rectangles of vortex center regions. In the right-most picture, rectangle colored light grey indicates the region significantly affected by a vortex, while v in the large crossed rectangle denotes the vortex center .

250m × 250m, populated by obstacles (colored dark grey). The initial state of the AUV is not known exactly. It is uniformly distributed inside the rectangles marked with 'I'. The action space is discretized into 5 actions, Move-North, Move-East, Move-South, Move-Northeast, and Move-Southeast. Due to motion error, whenever the AUV performs an action, it never reaches its intended next state, it is always off by within 2.5m radius from its intended destination. For AUV localization, several transponders with various maximum range are placed in the environment. Whenever the AUV is within a transponders' coverage (inside a circle with no label), it can localize itself with 100% accuracy. Otherwise, the AUV does not receive any observation. The AUV receives a reward of 100,000 when it reaches the goal region (circles labeled with 'G'). The environment contains local vortices whose center regions are in the rectangles marked with crosses. These vortex center regions

We tested PBPT on two scenarios (Figure 2) of an AUV navigating around offshore platforms, where the environment around the platform changes prior to the AUV deployment. In both scenarios, the AUV is set to operate at a particular depth. Hence, these are 2D navigation under uncertainty problems. Although this problem seems simple, the highly accentuated control error due to water current and vortex, unavailability of GPS underwater, and the highly cluttered environments make the problem difficult. Most AUVs handle uncertainty and environmental changes using variants of reactive greedy. But reactive greedy often fails in highly cluttered environments, as the planning horizon is often too short. State of the art POMDP planner requires significant time to generate reasonable policy for these problems.

The first scenario is shown in Figure 2(a). The state space S is a continuous 2D space of size

should be avoided by the AUV as it may damage the vehicle. Therefore, if the AUV passes through the crossed rectangle, we assign a penalty of -25,000. Each move is penalized by -1, as it takes energy. The original problem is in the left picture of Figure 2(a). Due to changes in the water flow, new vortices appear as shown in the right picture of Figure 2(a). Here, the changes are in the reward function of the states inside the new crossed rectangle.

The second scenario is shown in Figure 2(b). The AUV model, including its control and observation model, the reward function, and the legends are the same as in Scenario-1. But here, the changes is in the region affected by the vortex. Due to changes in the speed of the water flow, the vortex on the right becomes much stronger. As a result, when the AUV is at a state s inside the rectangle colored light grey in the right picture of Figure 2(b), the AUV drifts as much as $|\frac{1}{2}(v - s)|$ towards the vortex center, denoted as v . Here, the changes are in the transition function of the states inside the light grey rectangle, but not the reward function.

6.2 Experimental Setup

We implemented PBPT in C++, and use MCVI for policy computation from sampled beliefs. We compared PBPT with off-line replanning and policy reuse strategy. Both replanning and policy reuse use SARSOP [11] with MCVI [2] policy computation. In Policy reuse, we use the optimal policy of the original problem as an initial policy. All experiments were conducted in a PC with 2.27GHz Intel processor and 1.5GB RAM. Below is our experimental setup for each problem scenario.

To test PBPT, we first generate 30 different policies for the original problem, using SARSOP+MCVI with 2 hours of planning time. For each policy, we ran PBPT $10 \times$ to solve the modified problem. For each original policy, 10 modified policies are generated after every 10 minutes interval, for up to 2 hours. To compute the reward level reached by each modified policy, we ran 500 simulation runs and compute the average total discounted reward of the runs.

To test the performance of policy reuse, we use the same 30 original policies used to test PBPT. For each policy, we ran SARSOP+MCVI with the original policy as the initial policy, for $10 \times$ to solve the modified problem. Similar to PBPT, for each original policy, 10 new policies is generated after every 10 minutes interval, for up to 2 hours. And, for each new policy, we ran 500 simulation runs and computed the average total discounted reward. To test the performance of replanning, we generate 30 policies for the modified problems using SARSOP+MCVI, after every 10 minutes interval, for up to 2 hours. To compute the reward level reached by a policy, we ran 500 simulation runs, and computed the average total discounted reward.

6.3 Results

The results are in Figure 3. They indicate that PBPT can generate a good policy for the modified problems much faster than replanning and policy reuse strategies.

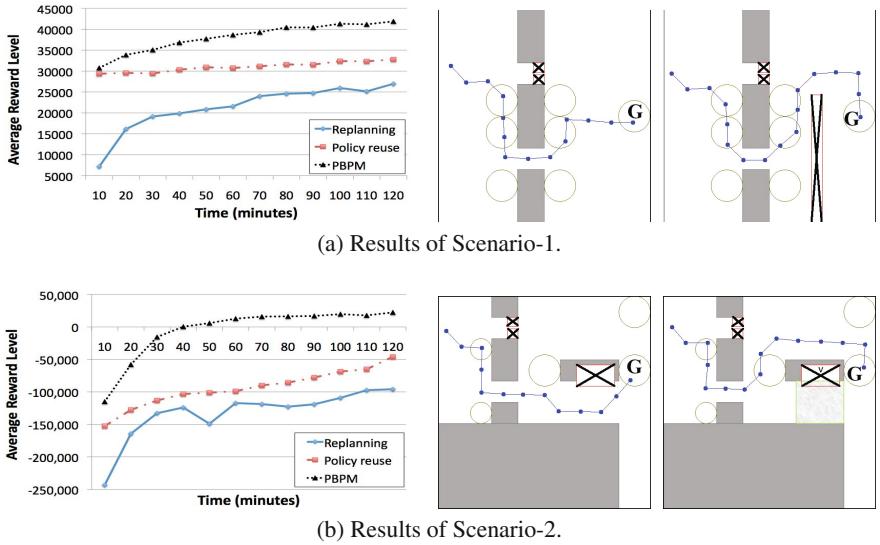


Fig. 3 Simulation results. Left: The average reward level of the policies. Middle: A typical simulation run of the original policy. Right: A typical simulation run of the policy generated by PBPT.

It is interesting to compare PBPT with policy reuse, as policy reuse reuses the policy of the original problem too. The main difference between the two is that in addition to the original policy, PBPT reuses the difference between the original and modified problems to guide belief space sampling. By doing so, PBPT can quickly construct a representative sampled representation of \mathcal{R}_n^* and focus on modifying critical parts of the original policy. Although policy reuse does use the original policy as an initial lower bound in SARSOP planning, and hence uses the original policy to indirectly guide its belief space sampling, policy reuse ignores information about the changes in the POMDP model. As a result, it is not as focused as PBPT in its belief space sampling, and hence is much slower in generating a good policy for the modified problems.

The middle and right most pictures in Figure 3 show typical simulation runs of the original policy and the modified policy generated by PBPT. The simulation runs indicate that in both scenarios, a good strategy for passing through the channels in the left should not change, as there are no change in that part of the environment. PBPT utilizes this information and focus more on modifying the strategy for moving after the AUV passes the left channel.

7 Conclusion

We propose a point-based algorithm, called Point-Based Policy Transformation (PBPT), that modifies a pre-computed policy according to changes in the POMDP

model. PBPT uses the difference between the original and modified POMDP models to identify subset of \mathcal{R}_o^* that is affected by the changes in the model. It uses this information to guide belief space sampling, which is a critical component for point-based POMDP planners. We show that PBPT converges to a good approximation of the best policy in $\Pi(\pi_o^*, P_o, P_n)$ or better, with high probability. Furthermore, preliminary results indicate that PBPT can generate a good policy for the modified POMDP problems much faster than recomputing the policy using the fastest POMDP planner today, even when the policy for the original problem is reused as an initial policy.

Many avenues are open for future work. We are currently working on speeding up PBPT, so that it can perform the modification on-line.

Acknowledgments. The authors thank Leslie P. Kaelbling and Tomas Lozano-Perez for fruitful discussion, David Hsu for cluster computing usage, and the AdaComp group at SoC, NUS for providing the MCVI code. This work is funded by the Singapore NRF through SMART, CENSAM.

References

1. Auer, P., Cesa-Bianchi, N., Freund, Y., Schapire, R.E.: The non-stochastic multi-armed bandit problem. *SIAM Journal on Computing* 32(1), 48–77 (2003)
2. Bai, H., Hsu, D., Lee, W.S., Ngo, V.A.: Monte Carlo Value Iteration for Continuous-State POMDPs. In: Hsu, D., Isler, V., Latombe, J.-C., Lin, M.C. (eds.) *Algorithmic Foundations of Robotics IX*. STAR, vol. 68, pp. 175–191. Springer, Heidelberg (2010)
3. van den Berg, J., Abbeel, P., Goldberg, K.: LQG-MP: Optimized Path Planning for Robots with Motion Uncertainty and Imperfect State Information. In: RSS (2010)
4. van den Berg, J., Overmars, M.: Roadmap-based motion planning in dynamic environments. *IEEE TRO* 21(5), 885–897 (2005)
5. de Berg, M., Cheong, O., van Kreveld, M., Overmars, M.: *Computational Geometry: Algorithms and Applications*. Springer (2000)
6. Dudley, R.M.: *Real Analysis and Probability*. Cambridge University Press (2002)
7. Hauser, K.: Randomized Belief-Space Replanning in Partially-Observable Continuous Spaces. In: Hsu, D., Isler, V., Latombe, J.-C., Lin, M.C. (eds.) *Algorithmic Foundations of Robotics IX*. STAR, vol. 68, pp. 193–209. Springer, Heidelberg (2010)
8. He, R., Brunskill, E., Roy, N.: PUMA: planning under uncertainty with macro-actions. In: AAAI (2010)
9. Jaillet, L., Siméon, T.: A PRM-based motion planner for dynamically changing environments. In: IROS (2004)
10. Kurniawati, H., Bandyopadhyay, T., Patrikalakis, N.M.: Global motion planning under uncertain motion, sensing, and environment map. In: RSS (2011)
11. Kurniawati, H., Hsu, D., Lee, W.S.: SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In: RSS (2008)
12. Lamiraux, F., Bonnafous, D., Lefebvre, O.: Reactive path deformation for nonholonomic mobile robots. *IEEE TRO* 20(6), 967–977 (2004)
13. LaValle, S.M., Sharma, R.: On motion planning in changing, partially-predictable environments. *IJRR* 16(6), 775–805 (1997)
14. Leven, P., Hutchinson, S.: Real-time path planning in changing environments. *IJRR* 21(12), 999–1030 (2001)

15. Papadimitriou, C.H., Tsitsiklis, J.N.: The Complexity of Markov Decision Processes. *Math. of Operation Research* 12(3), 441–450 (1987)
16. Pineau, J., Gordon, G., Thrun, S.: Point-based value iteration: An anytime algorithm for POMDPs. In: IJCAI, pp. 1025–1032 (2003)
17. Platt, R., Tedrake, R., Lozano-Perez, T., Kaelbling, L.P.: Belief space planning assuming maximum likelihood observations. In: RSS (2010)
18. Porta, J.M., Vlassis, N., Spaan, M.T.J., Poupart, P.: Point-Based Value Iteration for Continuous POMDPs. *JMLR* 7, 2329–2367 (2006)
19. Prentice, S., Roy, N.: The Belief Roadmap: Efficient Planning in Linear POMDPs by Factoring the Covariance. In: ISRR (2007)
20. Ross, S., Chaib-draa, B., Pineau, J.: Bayes-adaptive POMDPs. In: NIPS (2007)
21. Ross, S., Pineau, J., Paquet, S., Chaib-draa, B.: Online planning algorithms for POMDPs. *JAIR* 32, 663–704 (2008)
22. Smith, T., Simmons, R.: Point-based POMDP algorithms: Improved analysis and implementation. In: UAI (July 2005)
23. Stentz, A.: The Focussed D* Algorithm for Real-Time Replanning. In: IJCAI (1995)
24. Taylor, M.E., Stone, P.: Transfer learning for reinforcement learning domains: A survey. *JMLR* 10(1), 1633–1685 (2009)
25. Thrun, S.: Monte carlo POMDPs. In: NIPS, pp. 1064–1070 (2000)

From Formal Methods to Algorithmic Implementation of Human Inspired Control on Bipedal Robots

Shishir Nadubettu Yadukumar, Murali Pasupuleti, and Aaron D. Ames

Abstract. This paper presents the process of translating formal theory and methods to efficient algorithms in the context of human-inspired control of bipedal robots, with the end result being experimentally realized robust and efficient robotic walking with AMBER. We begin by considering human walking data and find outputs (or virtual constraints) that, when calculated from the human data, are described by simple functions of time (termed canonical walking functions). Formally, we construct a torque controller, through model inversion, that drives the outputs of the robot to the outputs of the human as represented by the canonical walking function; while these functions fit the human data well, they do not *a priori* guarantee robotic walking (due to the physical differences between humans and robots). An optimization problem is presented that determines the best fit of the canonical walking function to the human data, while guaranteeing walking for a specific bipedal robot; in addition, constraints can be added that guarantee physically realizable walking. We consider a physical bipedal robot AMBER and define a simple voltage based control law—utilizing only the human outputs and canonical walking function with parameters obtained from the optimization—for which we obtain walking in simulation. Since this controller does not require model inversion, it can be implemented efficiently in software. Moreover, applying this methodology to AMBER experimentally results in robust and efficient “human-like” robotic walking.

1 Introduction

Humans intrinsically display the following five major characteristics during walking: efficiency, naturalism, stability, simplicity, and versatility. Though human walking is a result of complex neuro-muscular interactions, it seems that the aforementioned high-dimensional behavior can be characterized by low-dimensional

Shishir Nadubettu Yadukumar · Murali Pasupuleti · Aaron D. Ames
Texas A&M University, College Station, TX 77843
e-mail: {shishirny, muralikris, aames}@tamu.edu

representation; for example, flat ground human walking behavior appears to be controlled by central pattern generators in spinal cord [11, 16]. This special property motivates the construction of a human-inspired controller for bipeds, which can help robots achieve human-like walking and thus paving way for the transition of bipeds from research labs to real environments. The philosophy behind our work is “simplicity implies robustness”. So the main objective of this paper is to develop a framework which can seamlessly integrate human walking data to design control algorithms which are simple, computationally tractable and easily realizable in physical robots.

Numerous approaches which aim to find the underlying “simplicity” in bipedal walking have been explored. Some of the first fundamental work in this area was by Marc Raibert, with the idea of achieving locomotion through the use of inverted pendulum models to create single-legged hoppers [20], and Tad Mcgeer who introduced the concept of passive walking [14] (which has also been realized on robots with efficient actuation [7]). Passive walking lead to the notion of controlled symmetries which allows for low energy walking [21], and the Spring Loaded Inverted Pendulum (SLIP) models [10, 19] for running robots. Walking has also been looked as a learning process [15] where the learning algorithm determines an optimal control policy by going through a collection of training sets. This method inherently requires several successful walking trials or good training examples in order to learn walking. In addition to these approaches, several methods have been proposed to directly bridge the gap between biomechanics and control theory by looking at human walking data to build models for bipedal robotic walking (see [8, 22] to name a few). Finally, combining many of the above approaches, significant strides have been made in underactuated bipedal walking (without feet) by using the idea of virtual constraints and hybrid zero dynamics (HZD) [23, 13], which resulted in amazingly robust walking even on rough terrain. HZD’s has indeed represented bipedal walking in a very elegant fashion, but implementing a HZD controller on a biped involves the determination of the parameters of the robot through identification experiments [17] which are not only very exhaustive and time consuming but are also not scalable to changes in hardware or robot structure.

This paper attempts to overcome the limitations posed by a HZD controller by using outputs and canonical walking functions which intrinsically capture the major characteristics of human walking behavior; this human-inspired control approach thus aims to further bridge the gap between robotics and control by using human walking data to formally design controllers (as first discussed in [3]). Specifically, by considering human walking data obtained through motion capture of subjects walking on flat ground, we find that certain outputs (or virtual constraints) of the human as calculated from this data can be represented by a special class of functions, termed canonical walking functions, characterized as time response of a linear spring-mass-damper system. Thus, humans appear to act like linear spring-mass-damper systems when walking on flat ground. By forming an optimization algorithm, where the cost is the least squares fit of the canonical walking functions to the human walking data, we obtain parameters for a human-inspired controller that provably results in stable underactuated robotic walking that is as close as possible to human walking. The

goal of this paper is to translate these formal methods in human-inspired control to efficient algorithms which are experimentally realizable on a physical robot; and specifically AMBER, a 2D underactuated bipedal robot (see Fig. 1).

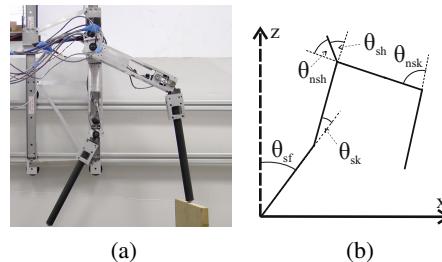


Fig. 1 The bipedal robot AMBER (a) and the angle convention used (b)

The main idea behind algorithmically implementing the formal ideas underlying the human-inspired control approach to bipedal robotic walking is that the inherent robustness present in the human-outputs that are chosen can be utilized to create simple and efficient feedback control strategies. Using the parameters of outputs obtained from the formal optimization problem that provably results in stable robotic walking, we define a simple voltage-based proportional (P) feedback control law on the human-inspired outputs (similar ideas have been explored for robotic manipulators [6, 12]). Since the actuators of AMBER are powered by DC motors, this naturally lends itself to simple implementation on the physical robot. The end result is that the voltage applied to the motors is directly proportional to the error between the desired and actual outputs of the robot, as represented by the canonical walking functions. The algorithmic implementation of this controller is, therefore, very efficient requiring less than 100 lines of pseudo-code. Therefore, the authors conclude that this simplistic algorithm becomes the foundation for this walking robot.

Implementing the algorithms developed on AMBER experimentally results in bipedal robotic walking that is efficient, robust and “human-like.” We argue that this is a direct result of the combination of formal methods with simple realization through the proportional voltage control on outputs that are directly inspired by human walking. In particular, we find good agreement between the simulation and experimental data. This indicates a direct connection between the formal methods and implementation. The experimental output data of the robot can also be related back to the human output data from which the controller was derived, for which there is a strong similarity showing that “human-like” walking is achieved. In addition the robot exhibits robustness in walking even under the influence of a wide variety of disturbances like push-pull, knee strike, tripping, obstacles (as high as 6cm) and even with hits from wooden blocks (see [2]). Moreover, this was achieved by using extremely low power DC motors (11 W). Hence simplicity of the voltage-based P-control on human-inspired outputs adheres closely to the philosophy that “simplicity implies robustness”, thereby rendering our walking algorithm, and the resulting robotic walking, efficient and robust.

2 Formal Methods for Bipedal Robot Modeling and Control

AMBER (short for **A & M** Bipedal Experimental Robot) is a 2D bipedal robot with 5 links (2 calves, 2 thighs and a torso, see Fig. 2). AMBER is 61cm tall with a total mass of 3.3kg (see Table 3b). It is made from aluminum with carbon fiber calves, powered by 4 DC motors and controlled through LabView software by National Instruments. The robot has point feet, and is thus underactuated at the ankle. In addition, since this robot is built for only 2D walking, it is supported in the lateral plane via a boom; this boom does *not* provide support to the robot in the sagittal plane. This means that the torso, through which the boom supports the robot, can freely rotate around the boom. The boom is fixed rigidly to a sliding mechanism (see Fig. 2), which allows the boom and consequently the biped, to move its hip front, back, up and down with minimum friction. The sliding mechanism is rested on a pair of parallel rails.

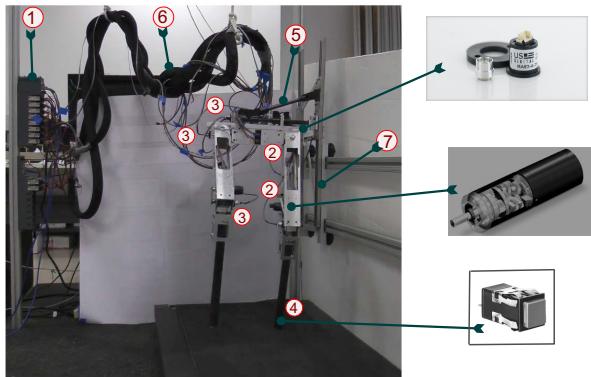


Fig. 2 Amber Experimental Setup. Parts marked are (1): NI cRIO, (2): Maxon DC Motors located in the calf and the torso, (3): Encoders on boom and the joints, (4): Contact switch at the end of the foot, (5): Boom, (6): Wiring with sheath protection, (7): Slider for restricting the motion to the sagittal plane

Let L_c , L_t , L_{tor} be the lengths of the calf, thigh and torso respectively (values are given in Fig. 3(b)) and $\theta = (\theta_{sf}, \theta_{sk}, \theta_{sh}, \theta_{nsh}, \theta_{nsk})^T$ be the angles of stance foot (foot of the stance leg), stance knee (knee of the stance leg), stance hip, non-stance (swing leg) hip and non-stance knee respectively. These variables form the configuration space of the robot, Q_R , and are shown in Fig. 1(b). Note that every time the swing foot hits the ground, the stance and non-stance nomenclatures are switched in the physical biped. Formally, we represent the bipedal robot as a hybrid system (see [3, 4] for a formal definition):

$$\mathcal{H}\mathcal{C}_R = (X_R, U_R, S_R, \Delta_R, f_R, g_R), \quad (1)$$

where $X_R \subset TQ_R$ is the domain given by the constraint $h_R(\theta) \geq 0$, where h_R is the height of the swing foot, $U_R \subset \mathbb{R}^4$ is the set of admissible controls, $S_R \subset X_R$ is the

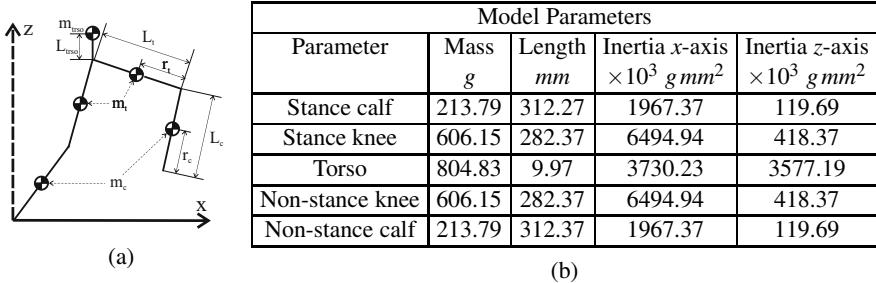


Fig. 3 (a) shows the notations used for masses and lengths for the links. (b) contains the masses, lengths and inertia of the 5 links of AMBER.

guard given by $h_R = 0$, Δ_R is the reset map which provides an instantaneous change in velocity at foot strike, and $\dot{x} = f_R(x) + g_R(x)u$, with $x = (\theta^T, \dot{\theta}^T)^T \in \mathbb{R}^{10}$ and u the torque input, is a control system obtained from the Lagrangian of the robot (which includes the mass and inertia of all links, the motors and the boom [18]). Since the robot is controlled by DC motors, we will also be interested in the control system which includes the DC motor models which results in the control system $\dot{x} = f_{Rv}(x) + g_{Rv}(x)V_{in}$ with voltage, V_{in} , being the control input.

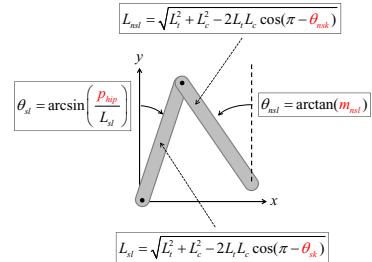
Human-Inspired Functions. By considering human walking data (as described in [3]), we discover that certain outputs (or virtual constraints), computed from the human joint data, display simple behavior; this core observation will be central to the design of human-inspired controllers. With the goal of picking outputs that elucidate the underlying structure of walking through a low-dimensional representation, or “virtual model,” we pick outputs that represent the human (and bipedal robot) as a compass-gait biped [7, 14] and the SLIP model [10]. In particular, the following collection of outputs yields such a representation (as illustrated on the right): the linearization of the x -position of the hip, p_{hip} , given by:

$$\delta p_{\text{hip}}(\theta) = L_c(-\theta_{sf}) + L_t(-\theta_{sk} - \theta_{sh}), \quad (2)$$

the linearization of the slope of the non-stance leg m_{nsi} , (the tangent of the angle between the z-axis and the line on the non-stance leg connecting the ankle and hip), given by:

$$\delta m_{nsi}(\theta) = -\theta_{sf} - \theta_{sk} - \theta_{sh} + \theta_{nsh} + \frac{L_c}{L_c + L_t} \theta_{nsk}, \quad (3)$$

the angle of the stance knee, θ_{sk} , the angle of the non-stance knee, θ_{nsk} , the angle of the torso from vertical, $\theta_{tor}(\theta) = \theta_{sf} + \theta_{sk} + \theta_{sh}$. It is important to note that



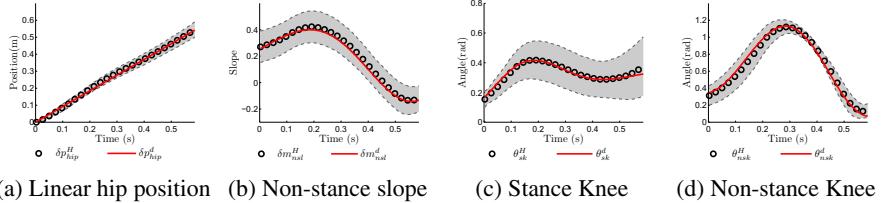


Fig. 4 The black circles indicate the mean of the human output data (see [5]). The grey shaded area indicates the standard deviation from the mean trajectory. The red solid lines are the fits of the canonical functions to the mean human data.

the linearized form of these outputs, rather than their original nonlinear form [3], is considered to allow for simpler implementation. Inspection of these outputs, as computed from the human data and shown in Fig. 4, reveals that they appear to display very simple behavior. In the case of the (linearized) position of the hip, it appears to essentially be a linear function of time:

$$\delta p_{\text{hip}}^d(t, v) = v_{\text{hip}} t, \quad (4)$$

The remaining outputs, (the non-stance slope δm_{nsl} , the stance knee θ_{sk} , the non-stance knee θ_{nsk} and the torso angle θ_{tor}) appear to act like a second order linear system. This motivated the introduction of the *canonical walking function* [3, 4]:

$$y_H(t, \alpha) = e^{-\alpha_4 t} (\alpha_1 \cos(\alpha_2 t) + \alpha_3 \sin(\alpha_2 t)) + \alpha_5. \quad (5)$$

which is simply the time solution to a linear mass-spring damper system, with $\alpha_4 = \zeta \omega_n$, where ζ is the damping ratio and ω_n is the natural frequency, $\alpha_2 = \omega_d$, where $\omega_d = \omega_n \sqrt{1 - \zeta^2}$ is the damped natural frequency, $\alpha_1 = c_0$ and $\alpha_3 = c_1$, where c_0, c_1 are determined by the initial conditions of the system and $\alpha_5 = g$, where g is the gravity related constant. Performing a least squares fit of the human output data with these functions results in near unity correlations (see [5]); implying that for the specific outputs chosen humans appear to act like linear mass-spring-damper systems. This is an important conclusion because it illustrates the simplicity in behavior that humans display when walking.

Human-Inspired Outputs. Having obtained outputs from the inspection of human data, the goal will be to construct a controller that drives the outputs of the robot to the outputs of the human, $y_a(\theta(t)) \rightarrow y_d(t, \alpha)$, with:

$$y_d(t, \alpha) = \begin{bmatrix} y_H(t, \alpha_{nsl}) \\ y_H(t, \alpha_{sk}) \\ y_H(t, \alpha_{nsk}) \\ y_H(t, \alpha_{tor}) \end{bmatrix}, \quad y_a(\theta) = \begin{bmatrix} \delta m_{nsl}(\theta) \\ \theta_{sk} \\ \theta_{nsk} \\ \theta_{tor}(\theta) \end{bmatrix}, \quad (6)$$

where $y_H(t, \alpha_i)$, $i \in \{nsl, sk, nsk, tor\}$ is the canonical walking function (5) but with parameters, α_i specific to the output being considered. By grouping these

parameters with the velocity of the hip, v_{hip} , that appears in (4) results in the vector of parameters $\alpha = (v_{\text{hip}}, \alpha_{nsl}, \alpha_{sk}, \alpha_{nsk}, \alpha_{tor}) \in \mathbb{R}^5$.

In order to remove the dependence of $y_d(t, \alpha)$, we introduce a parameterization of time based upon the fact that the (linearized) position of the hip is accurately described by a linear function of time. This motivates the following parameterization:

$$\tau(\theta) = \frac{\delta p_{\text{hip}}^R(\theta) - \delta p_{\text{hip}}^R(\theta^+)}{v_{\text{hip}}}, \quad (7)$$

where $p_{\text{hip}}^R(\theta^+)$ is the hip-position of the robot at the beginning of a step, where θ^+ is a point where the height of the non-stance foot is zero, i.e., $h_R(\theta^+) = 0$. Using this parameterization and (6), we define the following *human-inspired output*:

$$y_\alpha(\theta) = y_a(\theta) - y_d(\tau(\theta), \alpha). \quad (8)$$

Human-Inspired Control. Consider again the affine control system (f_R, g_R) associated with the hybrid model of AMBER (1). The human outputs were explicitly chosen so that the decoupling matrix, $A(\theta, \dot{\theta}) = L_{g_R} L_{f_R} y_\alpha(\theta, \dot{\theta})$ with L (Lie derivative) being nonsingular. Therefore, the human-inspired outputs are (vector) relative degree 2 and we can define the following torque controller:

$$u_{(\alpha, \varepsilon)}(\theta, \dot{\theta}) = -A^{-1}(\theta, \dot{\theta}) (L_{f_R}^2 y_\alpha(\theta, \dot{\theta}) + 2\varepsilon L_{f_R} y_\alpha(\theta, \dot{\theta}) + \varepsilon^2 y_\alpha(\theta)). \quad (9)$$

In other words, we can apply input/output linearization to obtain the linear system on the human-inspired outputs: $\ddot{y}_\alpha = -2\varepsilon\dot{y}_\alpha - \varepsilon^2 y_\alpha$. This system is exponentially stable, implying that for $\varepsilon > 0$ the control law $u_{(\alpha, \varepsilon)}$ drives $y_\alpha \rightarrow 0$. More generally, it renders the *zero dynamics surface*:

$$\mathbf{Z}_\alpha = \{(\theta, \dot{\theta}) \in TQ_R : y_\alpha(\theta) = \mathbf{0}, L_{f_R} y_\alpha(\theta, \dot{\theta}) = \mathbf{0}\} \quad (10)$$

invariant and exponentially stable for the *continuous dynamics*. Yet this property does not hold for the hybrid dynamics since discrete impacts in the system cause the state to be “thrown” off of the zero dynamics surface. Therefore, the goal is to achieve *hybrid zero dynamics*: $\Delta_R(S_R \cap \mathbf{Z}_\alpha) \subset \mathbf{Z}_\alpha$, i.e., render the zero dynamics surface invariant through impact. This will imply that the behavior of the robot will be characterizable by the “virtual model” that motivated the output functions under consideration, and will thus allow us to guarantee the existence of walking gaits.

Optimization Theorem. We now present the main theorem (originally introduced in [3, 4, 5]) that will be used to generate the control parameters and experimentally implemented on AMBER to obtain robotic walking. From the mean human walking data, we obtain discrete times, $t^H[k]$, and discrete values for the human output data, $y_i^H[k]$ and the canonical walking functions, $y_i^d(t, \alpha_i)$ for $i \in \text{Output} = \{\text{hip}, \text{nsl}, \text{sk}, \text{nsk}, \text{tor}\}$; for example, $y_{\text{nsl}}^H[k] = y_H(kT, \alpha_{\text{nsl}})$, where T is the discrete time interval and $k \in \mathbb{Z}$. We can now define the following human-data cost function:

$$\text{Cost}_{\text{HD}}(\alpha) = \sum_{k=1}^K \sum_{i \in \text{Output}} \left(y_i^H[k] - y_i^d(t^H[k], \alpha_i) \right)^2 \quad (11)$$

which is simply the sum of squared residuals. To determine the parameters for the human walking functions, we need only solve the optimization problem:

$$\alpha^* = \underset{\alpha \in \mathbb{R}^{21}}{\operatorname{argmin}} \text{Cost}_{\text{HD}}(\alpha) \quad (12)$$

which yields the least squares fit of the mean human output data with the canonical walking functions. While this provides an α^* that yields a good fit of the human data (see Fig. 4), these parameters will not result in robotic walking due to the differences between the robot and a human. Therefore, the goal is to determine these parameters which provide the best fit of the human data while simultaneously guaranteeing stable robotic walking for AMBER¹. This motivates the following theorem:

Theorem 1. *The parameters α^* solving the constrained optimization problem:*

$$\alpha^* = \underset{\alpha \in \mathbb{R}^{21}}{\operatorname{argmin}} \text{Cost}_{\text{HD}}(\alpha) \quad (13)$$

$$\text{s.t } y(\vartheta(\alpha)) = \mathbf{0} \quad (C1)$$

$$dy_\alpha(\Delta_\theta \vartheta(\alpha)) \Delta_{\dot{\theta}}(\vartheta(\alpha)) \dot{\vartheta}(\alpha) = \mathbf{0} \quad (C2)$$

$$dh_R(\vartheta(\alpha)) \dot{\vartheta}(\alpha) < 0 \quad (C3)$$

$$\mathcal{D}_Z(\vartheta(\alpha)) < 0 \quad (C4)$$

$$0 < \Delta_Z(\vartheta(\alpha)) < 1 \quad (C5)$$

yield hybrid zero dynamics: $\Delta_R(S_R \cap \mathbf{Z}_{\alpha^*}) \subset \mathbf{Z}_{\alpha^*}$. Moreover, there exists an $\hat{\varepsilon} > 0$ such that for all $\varepsilon > \hat{\varepsilon}$ the hybrid system $\mathcal{H}_R^{(\alpha^*, \varepsilon)}$, obtained by applying the control law (9) to the hybrid control system (1), has a stable periodic orbit with fixed point $(\theta^*, \dot{\theta}^*) \in S_R \cap \mathbf{Z}_{\alpha^*}$ that can be explicitly computed.

It is not possible to introduce all of the elements utilized in Theorem 1 due to space constraints but a detailed explanation can be found in [4]. Of particular importance is the point $(\vartheta(\alpha), \dot{\vartheta}(\alpha)) \in S_R \cap \mathbf{Z}_{\alpha}$ on the intersection of the zero dynamics surface and the guard that can be explicitly computed in terms of the parameters α

¹ It is important to note that [22] also presents an optimization problem that results in the least squares fit of human walking data subject to constraints that ensure HZD. Yet the theorem presented here is a substantial departure from the results in [22] in several important ways: [22] considers human joint angles, while we consider output functions, [22] fits high degree (9th order) polynomials to this data to create virtual constraints while we utilize the canonical walking function (which is nonlinear, and has far fewer parameters), [22] defines a configuration at the end of the step a priori and uses these to constrain the parameters of the outputs to ensure HZD while we define the point in terms of the parameters and allow it to change with the parameters as a result. All of these considerations require the use of different methods and theory and, fundamentally, changes the walking achieved.

(this point will later be used in additional constraints that will yield physically realizable walking). In other words, the configuration and velocities at the beginning and end of a step can change with the parameters allowing for a better translation of the outputs to robots which have different mass and length parameters from humans. In addition, $(\vartheta(\alpha), \dot{\vartheta}(\alpha))$ allows for the constraints in the optimization to be framed only in terms of the parameters, α . For these constraints, (C1) and (C2) (when coupled with the way $(\vartheta(\alpha), \dot{\vartheta}(\alpha))$ is computed from the outputs) ensure that the state of the robot is restricted to the zero dynamics surface even through the impacts. (C3) ensures that the non-stance foot intersects the guard transversally. (C4) and (C5) imply the existence and stability of a periodic orbit in the hybrid zero dynamics. In particular, $\mathcal{D}_Z(\vartheta(\alpha))$, which is a function of the energy contained in the zero dynamics, determines the existence of a step (which in turn determines the existence of the periodic orbit). $\Delta_Z(\vartheta(\alpha))$, gives the post impact velocity in the zero dynamics from pre-impact velocity, and therefore (C5) indicates the stability of the resulting periodic orbit. Finally, following from the results in [23], the existence and stability of a periodic orbit in the hybrid zero dynamics implies the stability of a periodic orbit in the full-order dynamics for sufficiently large ϵ , i.e., the end result is a stable walking gait.

Additional Constraints. The walking that we achieve using Theorem 1 should be physically realizable, which necessitates the additional constraints that ensure that the resulting control parameters will experimentally result in walking with AMBER:

(C6) Foot scuff prevention: The height of the swing foot at any point of time, must be such that it is greater than a quadratic polynomial, $h_R(\theta) > P(\theta)$, where $P(\theta) = ax_f(\theta)^2 + bx_f(\theta) + c$ with $x_f(\theta)$ being the horizontal position of the swing foot w.r.t. the stance foot and

$$a = -\frac{4h_{max}}{\text{SL}(\alpha)^2}, \quad b = \frac{4h_{max}\text{SL}(\alpha)}{\text{SL}(\alpha)^2}, \quad c = -\frac{4h_{max}x_f(\vartheta(\alpha))x_f(\Delta_R(\vartheta(\alpha)))}{\text{SL}(\alpha)^2},$$

where $\text{SL}(\alpha) = x_f(\vartheta(\alpha)) - x_f(\Delta_R(\vartheta(\alpha)))$ is the step length of the robot, computed from α through $\vartheta(\alpha)$. These constants, therefore, can be adjusted based on the required maximum stance foot height, h_{max} , and step length, $\text{SL}(\alpha)$.

(C7) Peak torque: The maximum torque delivered by the motors is limited. Therefore, the peak torque during a walking gait must be: $\max(u_{(\alpha,\epsilon)}(\theta, \dot{\theta})) < u_{max}$. Here $u_{(\alpha,\epsilon)}$ is dependent on the parameters α and ϵ , given in (9) and u_{max} is the maximum torque of the motors (for AMBER, $u_{max} = 2\text{Nm}$).

(C8) Hip-Velocity: The desired hip velocity of the biped must be within reasonable limits. Therefore, we introduce the constraint: $v_{min} < v_{hip} < v_{max}$.

For AMBER, $v_{min} = 0.1\text{m/s}$, $v_{max} = 0.6\text{m/s}$.

(C9) Angular velocities of joints: The maximum angular velocities with which the joints can turn are limited by the maximum angular velocities of the motors. The motors used in AMBER have a maximum angular velocity of 6.5rad/s .

3 Algorithmic Implementation and Experimental Results

The control law proposed in the previous section requires us to linearize the dynamics of AMBER through model inversion, which requires exact values of masses, inertias and dimensions of the robot. This is not only complex to implement but realizing the control law (9) on AMBER could potentially consume both time and resources, and achieving walking may still not be guaranteed due to a potentially inexact model. We, therefore, take a different approach by arguing that due to the “correct” choice of output functions—and specifically the human-inspired outputs—it is possible to obtain walking through simple controllers that are easy to implement and inherently more robust. Specifically, we present a proportional voltage controller on the human-inspired outputs, and demonstrate through simulation that robotic walking is obtained on AMBER. The simplicity of this controller implies that it can be efficiently implemented in software, and the details of this implementation are given. Finally, experimental results are presented showing that bipedal robotic walking is obtained with AMBER that is both efficient and robust.

Human-Inspired Voltage Control. Even if walking is obtained formally through input/output linearization, i.e., model inversion, the controllers are often implemented through PD control on the torque (see, for example, [17]). Since AMBER is not equipped with torque sensors, we sought an alternative method for feedback control implementation. Because AMBER is powered by DC motors, the natural input to consider is voltage, V_{in} , which indirectly affects the torques acting on the joints. Let V_{nsl} , V_{sk} , V_{nsk} and V_{tor} be the voltage input to the motors at the non-stance hip, stance knee, non-stance knee and stance hip, respectively. Define the following human-inspired proportional (P) voltage control law:

$$V_{in} = \begin{bmatrix} V_{nsl}(\theta) \\ V_{sk}(\theta) \\ V_{nsk}(\theta) \\ V_{tor}(\theta) \end{bmatrix} = -K_p y_\alpha(\theta), \quad (14)$$

where K_p is the constant matrix with its diagonal entries being the proportional gains for each of the motors and its non-diagonal entries being zero since the motors are controlled independently. This controller can be applied to the control system $\dot{x} = f_{R_v}(x) + g_{R_v}(x)V_{in}$ modeling the bipedal robot in conjunction with the motors. It can be seen that the control law (proportional control) solely depends on the generalized coordinates of robot (angles), θ , and not on the angular velocities. This marks a drastic change from the traditional ways of computing control. Evidently, and importantly, this avoids computation of angular velocities of the joints, which would have been computationally expensive and inaccurate.

It is important to note that the P voltage control law (14) is equivalent to a PD torque controller, where the derivative (D) constant is specified by the properties of the motor:

$$V_{in} = -K_p y_\alpha(\theta) = R_a i_a + K_\omega \omega \implies u(\theta, \dot{\theta}) = -K_\phi R_a^{-1} K_p y_\alpha(\theta) - K_\phi R_a^{-1} K_\omega \dot{\theta}$$

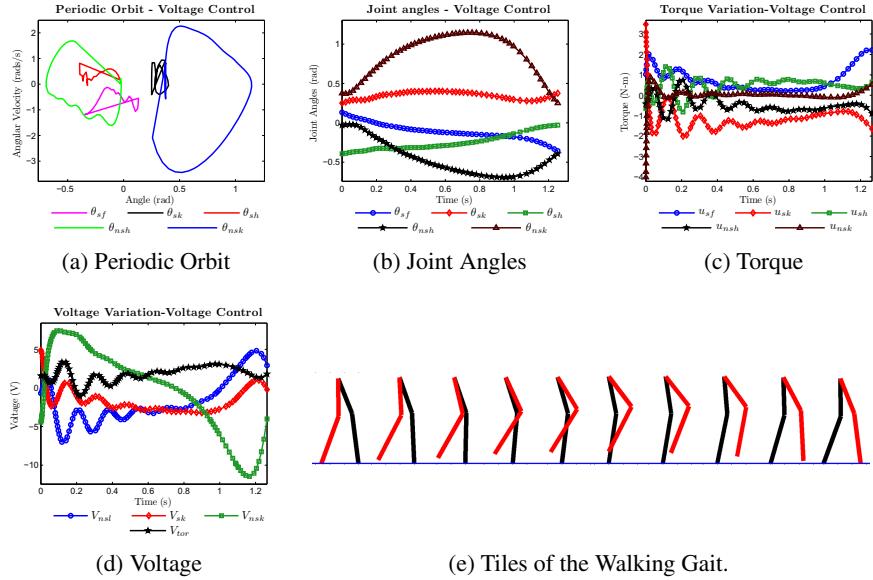
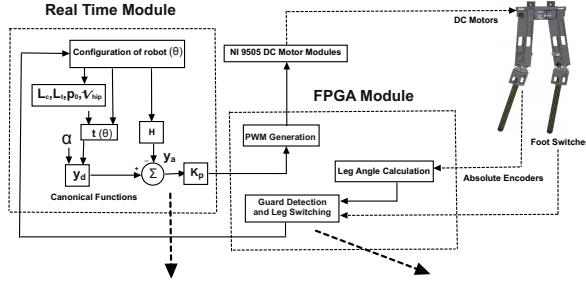


Fig. 5 Walking gait for AMBER obtained in simulation through P-voltage control

where K_φ is the torque constant matrix, and K_ω is the motor constant matrix. Hence, the control being applied is, in the end, related to the conventional torque PD control methods adopted in literature (see [9]).

Simulation Results. To obtain walking in simulation with AMBER through the formal methods discussed in Sect. 2, we begin with the hybrid model of the robot \mathcal{H}_R given in (1). Applying the human-inspired controller (9) and solving the optimization problem in Theorem 1 subject to the additional constraints (C6)-(C9) results in a hybrid system $\mathcal{H}_R^{(\alpha^*, \varepsilon)}$ that provably has a stable periodic orbit, i.e., a stable walking gait. The parameters α^* are then used in the P-voltage control, and the resulting closed loop system, which includes the mechanical and electrical models of AMBER is simulated. This results in a new periodic orbit (that is “near” the periodic orbit for the human-inspired torque controller). The resulting walking that is obtained through P-voltage control is shown in Fig. 5, along with the periodic orbit, joint angles, torques and voltages. The torque and voltage figures show oscillations. This is partly due to the application of a single order linear controller on a higher order nonlinear system.

Experimental Setup. AMBER’s experimental set up consists of three main segments: controller, actuators and sensors. Fig. 2 shows the locations of various sensors and actuators on AMBER. The Real Time (RT) processor and the FPGA in the cRIO form the controller, the DC motors powered by the H-bridge modules form the actuators, and the encoders connected to the joints along with contact switches at the feet form the sensors. The RT processor is connected via ethernet to a host PC for

**Algorithm 1** Real Time Module

```

Input: Amber Parameters: Calf Length( $L_c$ ), Thigh Length( $L_t$ );
Input: Optimization Parameters:  $\delta p_{\text{hip}}^R(\theta^+), v_{\text{hip}}, \alpha$ ;
Input: Proportional Gain ( $K_p$ );
Input:  $\theta_{sf}, \theta_{sk}, \theta_{sh}, \theta_{nsf}, \theta_{nsk}$  ;
Input:  $\bar{L} / R$  stance ; // Says which leg is supporting
Input: Encoder Not Working;
Input: Drive-Status;
Output: Enable Motors, Disable Motors ;
Output: PWM Duty Cycles for Voltages ;
1: Run Real Time (RT) VI;
2: Enable Motor ;
3: repeat
4:   Wait till all motors are Enabled
5: until ( Drive_Status == Enable )
6: while ( Stop-RT (or) Encoder_Error != 1 ) do
7:   if ( (Any Joint Angle Exceeds Limits)
        (or) (Encoder Not Working) ) then
8:     Encoder_Error ← 1;
9:   else
10:    Calculate  $\tau(\theta)$  ;
11:    PWM.Count1 ←  $K_p(y_H(\tau(\theta), \alpha_{nsf}) - \delta m_{nsf}(\theta(t)))$ ;
12:    PWM.Count2 ←  $K_p(y_H(\tau(\theta), \alpha_{sk}) - \theta_{sk}(t))$ ;
13:    PWM.Count3 ←  $K_p(y_H(\tau(\theta), \alpha_{nsk}) - \theta_{nsk}(t))$ ;
14:    PWM.Count4 ←  $K_p(y_H(\tau(\theta), \alpha_{tor}) - \theta_{tor}(\theta(t)))$ ;
15:   if ( $\bar{L} / R$  stance) then
16:      $V_{LHIP} \leftarrow$  PWM.Count1;
17:      $V_{LKNEE} \leftarrow$  PWM.Count3;
18:      $V_{RHIP} \leftarrow$  PWM.Count4;
19:      $V_{RKNEE} \leftarrow$  PWM.Count2;
20:   else
21:      $V_{LHIP} \leftarrow$  PWM.Count4;
22:      $V_{LKNEE} \leftarrow$  PWM.Count2;
23:      $V_{RHIP} \leftarrow$  PWM.Count1;
24:      $V_{RKNEE} \leftarrow$  PWM.Count3;
25:   end if
26: end while
28: Disable Motor Drives;
29: Report Errors and Stop the Real Time VI;

```

Algorithm 2 FPGA Module

```

Input: Enable Motors, Disable Motors;
Input: PWM Duty Cycles for Voltages;
Output:  $\theta_{sf}, \theta_{sk}, \theta_{sh}, \theta_{nsf}, \theta_{nsk}$  ;
Output:  $\bar{L} / R$  stance; // Says which leg is supporting
Output: Encoder Not Working; Drive-Status;
1: loop
2:   if ( Enable Motors (or) !Disable Motors ) then
3:     Enable all NI9505 DC Motors;
4:   end if
5:   if ( !Enable Motors (or) Disable Motors ) then
6:     Disable all NI9505 DC Motors;
7:   end if
8:   if ( PWM Duty Cycle > 75% ) then
9:     PWM Duty Cycle ← 75%;
10:   end if
11:   if ( Left Leg stance ) then
12:      $\bar{L} / R$  stance ← 0;
13:   else if ( Right Leg stance ) then
14:      $\bar{L} / R$  stance ← 1;
15:   end if
16:   if ( Signal low for 2 periods of encoder pulse ) then
17:     Encoder Not Working ← 1;
18:   else
19:     Encoder Not Working ← 0;
20:   end if
21:   if ( $\bar{L} / R$  stance ) then
22:      $\theta_{sh} \leftarrow$  LHIP_ANGLE ;
23:      $\theta_{nsk} \leftarrow$  LKNEE_ANGLE ;
24:      $\theta_{sh} \leftarrow$  RHIP_ANGLE ;
25:      $\theta_{sk} \leftarrow$  RKNEE_ANGLE ;
26:   else
27:      $\theta_{sh} \leftarrow$  LHIP_ANGLE ;
28:      $\theta_{sk} \leftarrow$  LKNEE_ANGLE ;
29:      $\theta_{sh} \leftarrow$  RHIP_ANGLE ;
30:      $\theta_{nsk} \leftarrow$  RKNEE_ANGLE ;
31:   end if
32:    $\theta_{sf} \leftarrow \theta_{tor} - \theta_{sk} - \theta_{sh}$  ;
33: end loop

```

Fig. 6 Controller overview with Pseudo Algorithms for RT and FPGA

data logging. The Real Time processor carries out floating point operations, while the FPGA interacts with I/O devices and provides hardware level interaction with the actuators and the sensors. AMBER walks on a treadmill, so the treadmill speed (resolution of 0.1 mph) is adjusted to be roughly equal to the desired average speed of walking (0.8 mph) and then fine-tuned by using an autotransformer. In this manner, we can bring the speed very close to the value predicted in simulation. The robot is then powered on and slowly lowered down to the treadmill; after a couple of steps,

the robot steadily falls into a limit cycle. The rest of this section is devoted to explaining the implementation of control law algorithms and the experimental results.

Implementation of Feedback Control Law. The schematic depicting the implementation of voltage based P-control on human-inspired outputs for AMBER is presented in Fig. 6. Contact switches are used to detect the guard and initiate the discrete dynamics for the robot. The variable, \bar{L}/R Stance, is used to identify which of the two legs is a stance leg. When the left leg is in stance phase, the watchdog in the controller checks every 5ms for right foot strike and vice-versa. Debounce logic is implemented, which discards any swing leg contact happening in less than 0.2s from previous foot strike. This eliminates accidental switchings when swing leg lifts off from the ground behind the stance leg.

The pseudo-algorithms used to implement the control law for Labview (RT and FPGA modules) are presented in Fig. 6. It is evident from the size of the code, that a complex task of human-like robotic walking was achieved using not more than 100 lines of pseudo code and minimal resources for hardware implementation (Fig. 7) with voltage-based P control on the human-inspired outputs. This highlights the effectiveness and simplicity of the algorithm used to implement the control law and we claim that, this potentially is one of the reasons why the walking obtained was robust.

Experimental Results. Implementing the algorithm in AMBER results in bipedal robotic walking (see [1] for a video of AMBER walking and responding to external disturbances). Walking tiles from the experiment and the simulation are shown in Fig. 8, and are in good agreement with each other. The similarity between the experimental and simulated behavior can be further seen by comparing the trajectories of the joint angles with the simulation, as shown in Fig. 9.

The human-inspired controller design has made the resulting walking efficient, robust and “human-like”. The specific cost of transport (electrical) for AMBER walking at 0.44 m/s is 1.88 using an average power of 27 W, which is very low

| Component | Total | Used (%) |
|-----------------|-------|----------|
| Slices | 7200 | 39 |
| Slice Registers | 28800 | 20.8 |
| Slice LUTs | 28800 | 28.1 |
| DSP48s | 48 | 20.8 |
| Block RAMs | 48 | 0 |

Fig. 7 FPGA Resource Used

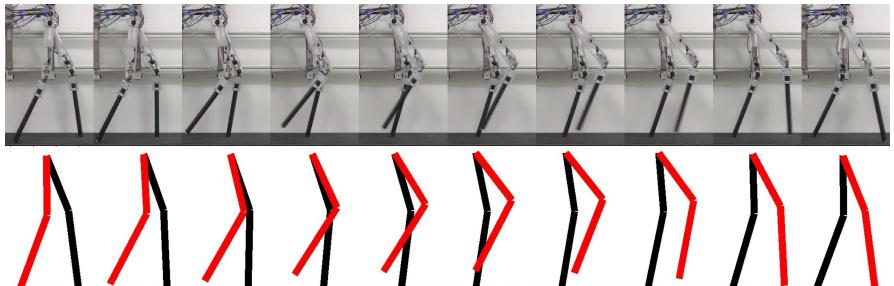


Fig. 8 Experimental walking tiles of AMBER (top) compared with the walking obtained from the simulation (bottom). See [1] for the video of AMBER walking.

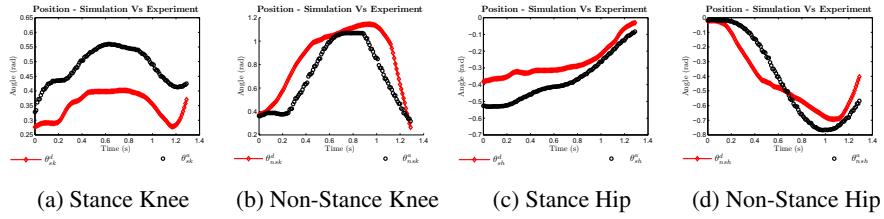


Fig. 9 Experimental vs. simulated angles over one step

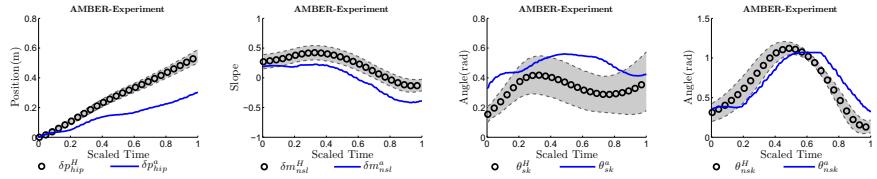


Fig. 10 Experimental robotic output data compared to human output data

Table 1 Power to Weight Ratio (W/kg)

| Power Plot | Robot | Compliance | Value | Reference |
|------------|---------------|------------|-------|----------------------------------|
| | AMBER | No | 13.2 | 44W, 3.33kg |
| | ERNIE | Yes | 53.7 | 1 kW ¹ , 18.6kg [23] |
| | RABBIT | No | 93.2 | 2.98 kW ² , 32kg [23] |
| | NAO H25 V3.3 | No | 35.4 | 177W ³ , 5kg |
| | DARWIN | No | 27.6 | 80W ⁴ , 2.9kg |
| | MABEL | Yes | 58.6 | 3.28 kW ⁵ , 56kg [9] |
| | Cornell Biped | Yes | 1.5 | 19W ⁶ , 12.7kg [7] |

¹ EC 45-136212 (250 W), so for 4 motors total power: 1 kW

² RS 420J performance curves indicate 1 HP motor, for 4 motors total power: 2.98 kW

³ Type1: RE-MAX 17 (4W), Type2: RE-MAX 24 (11W). Type1 Motors on Head-2, Type1 Motors on Arms -12, Type2 Motors on Legs- 11. Total power for 25 motors: 177W

⁴ Specifications of Dynamixel RX-28 at 12 V has values of RE-MAX 17 motor(4W) with 1:193 gear ratio, so for 20 motor modules total power: 80 W

⁵ QBO5600-X0X (843.892 W)- 2 ,QBO5601-X0X(798.605 W) - 2 , so for 4 motors total power: 3.28 KW

⁶ For two 9.5 Watt 6.4 oz MicroMotors, total power: 19 W

compared to commercial robots like Honda [7] and it also has the least installed power to weight ratio (W/kg) among robots with no compliance as indicated in Table 1. In addition, the walking achieved with AMBER is incredibly robust; with no changes to the controller, AMBER is able to successfully handle disturbances like push-pull, hitting, tripping and even rough terrains with ease (see video: [2]).

Finally, comparison between AMBER outputs with the human data, from which the controller was originally derived (see Fig. 10), demonstrate that the walking is remarkably “human-like” despite the vast differences between AMBER and humans.

4 Conclusions

This paper successfully translated formal methods in human-inspired control to efficient algorithmic implementation and finally experimentally realized walking on AMBER. Specifically, formal methods utilizing model inversion were presented that provably result in walking; these were translated to implementable control strategies through voltage-based P-control on the human-inspired outputs. The simplicity of the algorithmic implementation of this control law resulted in low computation overhead thereby enabling us to use a time step of 5ms for each calculation and minimal hardware resources (39% of the FPGA). With no feet actuation, the overall energy efficiency is enhanced, which enabled us to have the lowest installed W/kg when compared with the contemporary robots, as shown in Table 1. While it must be pointed out that some of the robots also carry support electronics and some are meant for running (MABEL) which has resulted in them requiring higher power; nevertheless, we claim that the proposed method of voltage-based P-control on human-inspired outputs can result in robust walking with a very good cost of transport. While achieving a walking gait that is very close to the natural human walking gait, the biped is also tolerant to changes in terrain (6cm), change of treadmill speeds (12.5%) and even force disturbances on all of the links of the robot. It is important to highlight that the proposed voltage-based control law is dependent only on the configuration variables as opposed to using speed and acceleration feedback, which constitute the inherent simplicity and advantage of indirectly affecting the torque produced at a joint through voltage at the motor level. The end result is robust walking both in simulation as well as in the experiment.

Acknowledgments. The authors would like thank Michael Zeagler and Jordan Lack for their support in machining and assembling of AMBER. We also thank National Instruments for their support for the project by providing the necessary hardware and software, and specifically Andy Chang for numerous discussions on implementation. This work is supported by NSF grants CNS-0953823 and CNS-1136104, NHARP award 00512-0184-2009, NASA grant NNX12AB58G.

References

1. Amber walking and undergoing robustness tests,
<http://youtu.be/SYXWoNU8QUE>
2. Robustness tests conducted on AMBER, <http://youtu.be/RgQ8atV1NW0>
3. Ames, A.D.: First Steps Toward Automatically Generating Bipedal Robotic Walking from Human Data. In: Kozłowski, K. (ed.) Robot Motion and Control 2011. LNCIS, vol. 422, pp. 89–116. Springer, Heidelberg (2012)

4. Ames, A.D.: First steps toward underactuated human-inspired bipedal robotic walking. In: 2012 IEEE Conference on Robotics and Automation, St. Paul, Minnesota (2012)
5. Ames, A.D., Cousineau, E.A., Powell, M.J.: Dynamically stable robotic walking with NAO via human-inspired hybrid zero dynamics. In: Hybrid Systems: Computation and Control, Beijing, China (April 2012)
6. Burg, T., Dawson, D., Hu, J., de Queiroz, M.: An adaptive partial state-feedback controller for RLED robot manipulators. *IEEE Transactions on Automatic Control* 41(7), 1024–1030 (1996)
7. Collins, S., Ruina, A., Tedrake, R., Wisse, M.: Efficient bipedal robots based on passive-dynamic walkers. *Science* 307, 1082–1085 (2005)
8. Geyer, H., Herr, H.: A muscle-reflex model that encodes principles of legged mechanics produces human walking dynamics and muscle activities. *IEEE Transactions on Neural Systems and Rehabilitation Engineering* 18(3), 263–273 (2010)
9. Grizzle, J.W., Hurst, J., Morris, B., Park, H., Sreenath, K.: MABEL, a new robotic bipedal walker and runner. In: American Control Conference, St. Louis, MO, pp. 2030–2036 (2009)
10. Holmes, P., Full, R.J., Koditschek, D., Guckenheimer, J.: The dynamics of legged locomotion: Models, analyses, and challenges. *SIAM Rev.* 48, 207–304 (2006)
11. Ijspeert, A.J.: Central pattern generators for locomotion control in animals and robots: a review. *Neural Networks* 21(4), 642–653 (2008)
12. Liu, C., Cheah, C.C., Slotine, J.E.: Adaptive jacobian tracking control of rigid-link electrically driven robots based on visual task-space information. *Automatica* 42(9), 1491–1501 (2006)
13. Manchester, I.R., Mettin, U., Iida, F., Tedrake, R.: Stable dynamic walking over uneven terrain. *The International Journal of Robotics Research* 30(3), 265–279 (2011)
14. McGeer, T.: Passive dynamic walking. *Intl. J. of Robotics Research* 9(2), 62–82 (1990)
15. Morimoto, J., Cheng, G., Atkeson, C., Zeglin, G.: A simple reinforcement learning algorithm for biped walking. In: Proceedings of the 2004 IEEE International Conference on Robotics & Automation, New Orleans, LA (May 2004)
16. Nielsen, J.B.: How we walk: Central control of muscle activity during human walking. *The Neuroscientist* 9(3), 195–204 (2003)
17. Park, H.-W., Sreenath, K., Hurst, J., Grizzle, J.W.: Identification of a bipedal robot with a compliant drivetrain: Parameter estimation for control design. *IEEE Control Systems Magazine* 31(2), 63–88 (2011)
18. Pasupuleti, M., Nadubettu Yadukumar, S., Ames, A.D.: Human-inspired underactuated bipedal robotic walking with amber on flat-ground, up-slope and uneven terrain. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Algarve, Portugal (2012)
19. Pouliquen, I., Grizzle, J.W.: The Spring Loaded Inverted Pendulum as the Hybrid Zero Dynamics of an Asymmetric Hopper. *Transaction on Automatic Control* 54(8), 1779–1793 (2009)
20. Raibert, M.H.: Legged robots. *Communications of the ACM* 29(6), 499–514 (1986)
21. Spong, M.W., Bullo, F.: Controlled symmetries and passive walking. *IEEE TAC* 50(7), 1025–1031 (2005)
22. Srinivasan, S., Raptis, I.A., Westervelt, E.R.: Low-dimensional sagittal plane model of normal human walking. *ASME J. of Biomechanical Eng.* 130(5) (2008)
23. Westervelt, E.R., Grizzle, J.W., Chevallereau, C., Choi, J.H., Morris, B.: Feedback Control of Dynamic Bipedal Robot Locomotion. CRC Press, Boca Raton (2007)

Direct Trajectory Optimization of Rigid Body Dynamical Systems through Contact

Michael Posa and Russ Tedrake

Abstract. Direct methods for trajectory optimization are widely used for planning locally optimal trajectories of robotic systems. Most state-of-the-art techniques treat the discontinuous dynamics of contact as discrete modes and restrict the search for a complete path to a specified sequence through these modes. Here we present a novel method for trajectory planning through contact that eliminates the requirement for an *a priori* mode ordering. Motivated by the formulation of multi-contact dynamics as a Linear Complementarity Problem (LCP) for forward simulation, the proposed algorithm leverages Sequential Quadratic Programming (SQP) to naturally resolve contact constraint forces while simultaneously optimizing a trajectory and satisfying nonlinear complementarity constraints. The method scales well to high dimensional systems with large numbers of possible modes. We demonstrate the approach using three increasingly complex systems: rotating a pinned object with a finger, planar walking with the Spring Flamingo robot, and high speed bipedal running on the FastRunner platform.

1 Introduction

Trajectory optimization is a powerful framework for planning locally optimal trajectories for linear or nonlinear dynamical systems. Given a control dynamical system, $\dot{x} = f(x, u)$, and an initial condition of the system $x(0)$, trajectory optimization aims to design a finite-time input trajectory, $u(t), \forall t \in [0, T]$, which minimizes some cost function over the resulting input and state trajectories. There are a number of popular methods for transcribing the trajectory optimization problem into a finitely parameterized nonlinear optimization problem (see [2]). Broadly speaking, these transcriptions fall into two categories: the *shooting* methods and the *direct* methods. In shooting methods, the nonlinear optimization searches over (a finite

Michael Posa · Russ Tedrake

Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, MA 02193
e-mail: {mposa, russt}@mit.edu

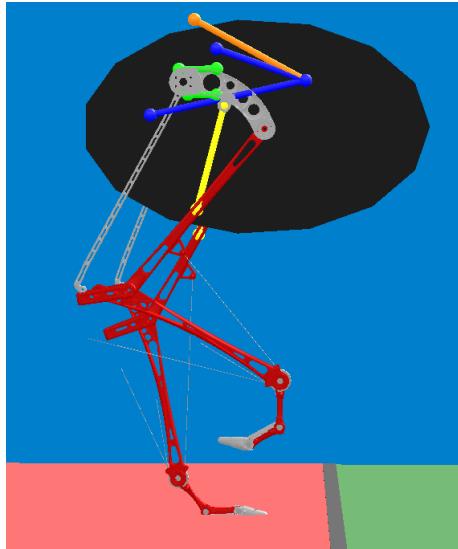
parameterization of $u(t)$, using a forward simulation from $x(0)$ to evaluate the cost of every candidate input trajectory. In direct methods, the nonlinear optimization simultaneously searches over parameterizations of $u(t)$ and $x(t)$; here no simulation is required and instead the dynamics are imposed as a set of optimization constraints, typically evaluated at a selection of *collocation* points[9]. Mixtures of shooting and direct methods are also possible, and fall under the umbrella of *multiple shooting*.

Direct methods typically enjoy a considerable numerical advantage over the shooting methods. Shooting methods are plagued by poorly conditioned gradients; for instance, a small change in the input tape at $t = 0$ will often have a dramatically larger effect on the cost than a small change near time T . Direct methods can also be initialized with an initial guess for the state trajectory, $x(t)$, which can often be easier to determine than an initial tape for $u(t)$. A reasonable initial tape is generally helpful in avoiding problems with local minima. The resulting optimization problems are also sparse, allowing efficient (locally optimal) solutions with large-scale sparse solvers such as SNOPT[8], and trivial parallel/distributed evaluation of the cost and constraints.

In this paper, we consider the problem of designing direct trajectory optimization methods for rigid-body systems with contact. This is an essential problem for robotics which arises in any tasks involving locomotion or manipulation. The collision events that correspond with making or breaking contact, however, greatly complicate the trajectory optimization problem as they result in large or impulsive forces and rapid changes in velocity. While it is possible to resolve contact through the use of continuous reaction forces like simulated springs and dampers, the resulting differential equations are typically stiff and require an extremely small time step, making trajectory optimization very inefficient. For numerical efficiency, a preferred method is to approximate contact as an *autonomous hybrid* dynamical system that undergoes discontinuous switching (see [20]). The discrete transitions are fully autonomous as we can directly control neither the switching times nor the switching surface.

There are a number of impressive success stories for trajectory optimization in these hybrid models, for instance the optimization of an impressive 3D running gait[14]. These results primarily use direct methods. But they are plagued with one major short-coming - the optimization is constrained to operate within an *a priori* specification of the ordering of hybrid modes. For a human running where motion capture can provide a good initial guess on the trajectory, this may be acceptable. It is much more difficult to imagine a mode specification for a multi-fingered hand manipulating a complex object that is frequently making and breaking contact with different links on the hand. Perhaps as a result, there is an apparent lack of planning solutions for robotic manipulation which plan *through* contact - most planners plan up to a pre-grasp then activate a separate grasping controller. Indeed, the multi-contact dynamics engines used to simulate grasping[10, 11] do not use hybrid models of the dynamics, because the permutations of different possible modes grows exponentially with the number of links and contact points, and because hybrid models can be plagued by infinitely-frequent collisions (e.g., when a bouncing ball comes to rest on a surface). Instead, simulation tools make use of time-stepping solutions that solve contact constraints using numerical solutions to a linear

Fig. 1 The bipedal FastRunner robot is designed to run at speeds of over 20 mph. Each leg has 5 degrees of freedom and multiple passive springs and tendons. The legs are driven at the hip to keep the leg mass as low as possible.



complementarity problem (LCP) which can be solved with well known methods like Lemke's Algorithm[17]. Significant additional work has been done to extend the original LCP methods to guarantee solutions to multi-body contact[1, 11].

In many cases, solutions to the LCP for forward simulation can be found via the minimization of a convex quadratic program (QP). In this paper we demonstrate that it is possible, indeed natural, to fold the complementarity constraints directly into nonlinear optimization for trajectory design, which we solve here using sequential quadratic programming (SQP). The key trick is resolving the contact forces, the mode-dependent component of the dynamics in the traditional formulation, as additional decision variables in the optimization. We demonstrate that this is an effective and numerically robust way to solve complex trajectories without the need for a mode schedule.

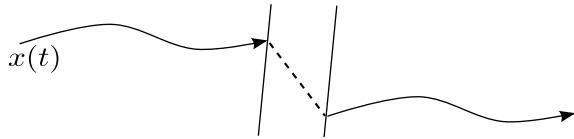
Specifically, this work was motivated by the challenge of optimizing trajectories for a new running robot called "FastRunner"[5]. FastRunner, illustrated in Figure 1 is a bipedal robot concept designed to run at speeds over 20 mph and up to 50 mph. Most notably, FastRunner has an exceptionally clever, but also exceptionally complex, leg design with four-bar linkages, springs, clutches, hard joint stops, tendons and flexible toes. The planar FastRunner the model has 13 degrees of freedom, 6 contact points, and 16 additional constraint forces, and was beyond the scope of our previously existing trajectory optimization tools.

1.1 The Pitfalls of Mode Schedules

For simple hybrid systems, including point foot models of walking robots, trajectories of the hybrid system can be described by a smooth dynamics up until a guard

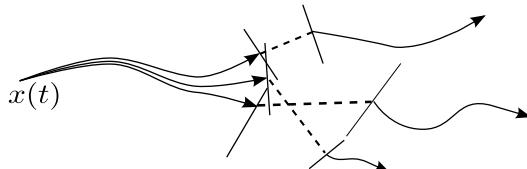
condition is met (e.g., the robot's foot hits the ground), then a discontinuous jump in state-space, corresponding here to an instantaneous loss of velocity as energy is dissipated into the ground—followed by another smooth dynamical system, as cartooned in Figure 2. For a fixed mode schedule, direct methods for hybrid trajectory optimization proceed by optimizing each segment independently, with additional constraints enforcing that the segments connect to each other through the hybrid events.

Fig. 2 Hybrid trajectories can be found by individually optimizing over the continuous dynamics of a specified mode sequence. The dashed line indicates the discontinuous jump from one mode to another.



However, when the dynamics get more complex, the geometric constraints imposed by the hybrid system become more daunting. The FastRunner model has a hybrid event every time that any of the ground contact points (on either foot) make or break contact with the ground. But the model also undergoes a hybrid transition every time that any one of the joints hits a joint-limit, and every time that any one of the ground reaction forces enters or leaves the friction cone (transitioning from rigid to a sliding contact). Indeed, the number of possible hybrid modes of the system grows exponentially with the number of constraints. The geometry of the hybrid guards becomes very complex, as cartooned in Figure 3. In these models, small changes to the input tape can result in a very different schedule of hybrid modes. Restricting the trajectory optimization search to the initial mode schedule can result in a very limited search and often in failure to find a feasible trajectory that satisfies all of the constraints.

Fig. 3 When the hybrid transition map of Figure 2 becomes increasingly complex, it is no longer trivial to specify the optimal mode sequence



Despite the obvious limitation of requiring this mode schedule, it has proven surprisingly difficult to remove this assumption in the direct methods. Slight variations from the original sequence are possible if the formulation allows the time duration of individual modes to vanish, as in [16]. For problems with fewer possible modes, outer optimization loops have been used to determine the hybrid mode schedule,

as in [19]. In some cases, the combinatorial problem of solving for a mode schedule have been addressed by combinatorial planners; a variant of the Rapidly-Exploring Random Tree (RRT) algorithm was used in [15] to produce bounding trajectories for a quadruped over terrain. Methods for optimal control which approximate the global optimal, such as brute force methods based on dynamic programming, have also been applied[3], but are so far limited to low dimensional problems.

1.2 Contact Dynamics as a Linear Complementarity Problem

In order to avoid the combinatorial explosion of hybrid models, simulation techniques in computer graphics and in grasping research use a different formulation of contact, summarized briefly here. The forward dynamics of a rigid-body (e.g., with a floating base) subject to contact constraints can be written as

$$\text{find } \ddot{q}, \lambda \quad (1)$$

$$\text{subject to } H(q)\ddot{q} + C(q, \dot{q}) + G(q) = B(q)u + J(q)^T\lambda, \quad (2)$$

$$\phi(q) \geq 0 \quad (3)$$

$$\lambda \geq 0 \quad (4)$$

$$\forall i, \phi_i(q)\lambda_i = 0 \quad (5)$$

where $q \in \mathbb{R}^n$ is the vector of generalized coordinates, $H(\cdot)$ is the inertial matrix, $C(\cdot, \cdot)$ represents the Coriolis terms, $G(\cdot)$ the gravitational forces, and $B(\cdot)$ is the input mapping. $\phi(q) \geq 0$ represents a non-penetration constraint where the equality holds if and only if the two bodies are in contact, λ represents the constraint forces, and $J(\cdot) = \frac{\partial \phi}{\partial q}$ represents the Jacobian projecting constraint forces into the generalized coordinates. Vector inequalities are to be interpreted as element-wise inequality constraints. Equation 5, often referred to as a complementarity constraint, ensures that the contact forces can be non-zero if and only if the bodies are in contact. Solving these equations for \ddot{q} requires solving a Nonlinear Complementarity Problem (NCP).

The solution to these dynamics are potentially complex, involving high impact forces occurring across very short time periods (e.g., at the moment of a new collision). In the limiting case of purely rigid bodies, the constraint forces, $\lambda(t)$, must be modeled with the Dirac δ functions, or as hybrid impulsive events. However, many of these complexities can be avoided by discretizing the system in time. [17] introduced a time-stepping method that only considers the integral of contact forces over a period and so does not differentiate between continuous and impulsive forces.

For the Stewart and Trinkle time-stepping method, the manipulator dynamics and constraint terms can be evaluated at the known q, \dot{q} , allowing the state at the next time step to be written as the solution to a linear set of equations subject to linear complementarity constraints, resulting in an LCP. For 3D models, the Coulomb

friction cone must be approximated by a series of linear constraints, but otherwise still fits into the LCP framework. It has been proven that solutions exist to this LCP and, under reasonable conditions, can be computed using Lemke's Algorithm or similar methods. Here, solving each LCP corresponds to simulating a single time step.

Despite the success in forward simulation, to our knowledge the LCP formulation has not been used in direct trajectory optimization. In related work, [18] explored the use of Stochastic Complementarity Problems in a shooting method. Their approach does not require an explicit mode schedule, but instead relies on the passive dynamics of the system to initiate contact. In cases where the simulated dynamics of the initial tape do not find the desired mode sequence, no gradient information will be available to guide the local search.

2 Approach

Contact constraints formulated using the complementarity conditions fit naturally into the direct formulation of trajectory optimization. Rather than solving the LCP for the contact forces λ at each step, we directly optimize over the space of feasible states, control inputs, constraint forces, and trajectory durations. Treating the contact forces as optimization parameters is similar to how direct methods treat the state evolution implicitly. The number of parameters and constraints increases, but the problem is often better conditioned and more tractable to state of the art solvers.

$$\underset{\{h, x_0, \dots, x_N, u_1, \dots, u_N, \lambda_1, \dots, \lambda_N\}}{\text{minimize}} \quad g_f(x_N) + h \sum_{k=1}^N g(x_{k-1}, u_k) \quad (6)$$

Here, $g(\cdot, \cdot)$ and $g_f(\cdot)$ are the integrated and final cost functions respectively. This optimization problem is subject to constraints imposed by the manipulator dynamics and by rigid body contacts. To integrate the dynamics, both forwards and backwards Euler methods are equally applicable. Unlike with forward simulation, our methods are already implicit and so backwards integration adds no computational cost. Given that we wish to use large time intervals, we chose backwards integration for added numerical stability. For ease of notation, we will write $H_k = H(q_k)$ and likewise for other matrix functions in the manipulator dynamics. Where h is the length of the time-steps and for $k = 1, \dots, N - 1$, the manipulator dynamics from (2) imply the constraints:

$$\begin{aligned} q_k - q_{k+1} + h\dot{q}_{k+1} &= 0 \\ H_{k+1}(\dot{q}_{k+1} - \dot{q}_k) + h(C_{k+1} + G_{k+1} - B_{k+1}u_{k+1} - J_{k+1}^T\lambda_{k+1}) &= 0 \end{aligned} \quad (7)$$

For the rest of this paper, we will, for simplicity, only consider the case where the contact dynamics are planar. However, since we use the unsimplified nonlinear constraints, these methods easily extend to 3D contacts. For a given contact point,

write the contact force in a reference frame aligned to the contact surface, $\lambda = [\lambda_x \lambda_z]^T$. We then have the same set of unilateral and bilateral contact constraints as before:

$$\phi(q_k) \geq 0 \quad (8)$$

$$\lambda_{k,z} \geq 0 \quad (9)$$

$$(\infty\lambda_{k,z})^2 - \lambda_{k,x}^2 \geq 0 \quad (10)$$

$$\phi(q_k)\lambda_{k,z} = 0 \quad (11)$$

$$\psi(q_k, \dot{q}_k)\lambda_{k,z} = 0 \quad (12)$$

where $\psi(q, \dot{q})$ is the relative velocity between two bodies that can make contact and Equation 12 represents an additional no-slip constraint.

Unlike with the task of pure simulation where these constraints and the dynamical constraints in (7) can be linearized about the initial state, here we must consider the higher order behavior and so we use the true formulation as a Nonlinear Complementarity Problem (NCP). Coulomb friction is described in (10), although any differentiable friction formulation could be used. We can construct an alternative formulation that allows for trajectories that violate the no-slip constraint by modeling sliding and kinetic friction. To do this, we introduce the slack variable $\gamma_k \geq 0$ that, when the lateral force is non-zero, represents the magnitude of the contact velocity. Additionally, substitute $\lambda_{k,x} = \lambda_{k,x}^+ - \lambda_{k,x}^-$ where $\lambda_{k,x}^+, \lambda_{k,x}^- \geq 0$. With these new contact parameters, replace (10) and (12) with:

$$\gamma_k + \psi(q_k, \dot{q}_k) \geq 0 \quad (13)$$

$$\gamma_k - \psi(q_k, \dot{q}_k) \geq 0 \quad (14)$$

$$\infty\lambda_{k,z} - \lambda_{k,x}^+ - \lambda_{k,x}^- \geq 0 \quad (15)$$

$$(\gamma_k + \psi(q_k, \dot{q}_k))\lambda_{k,x}^+ = 0 \quad (16)$$

$$(\gamma_k - \psi(q_k, \dot{q}_k))\lambda_{k,x}^- = 0 \quad (17)$$

$$(\infty\lambda_{k,z} - \lambda_{k,x}^+ - \lambda_{k,x}^-)\gamma_k = 0 \quad (18)$$

Simple position constraints, like joint limits, can be written in a similar manner. Here, λ is a torque or force acting directly on a joint. To express the requirement that $q \leq q_{max}$, write:

$$\phi(q_k) = q_{max} - q_k \geq 0 \quad (19)$$

$$\lambda_k \leq 0 \quad (20)$$

$$\phi(q_k)\lambda_k = 0 \quad (21)$$

It is important to note the relative indexing of the complementarity and dynamical constraints. Over the interval $[t_k, t_{k+1}]$, the contact impulse can be non-zero if and only if $\phi(q_{k+1}) = 0$; that is, the bodies must be in contact at the end of the given

interval. This allows the time-stepping integration scheme to approximate inelastic collisions where the interacting bodies stick together. This is not necessarily an appropriate approximation for bodies that may rapidly rebound off one another, since any compliance must be modeled through a linkage in one of the bodies and the time step must be appropriately small.

We also note here the role of the discrete time steps when resolving contacts. Our approach makes no effort to determine the exact time that contact between bodies is made or broken. The constraint forces over the time step directly before collision are precisely those required for the two bodies to be tangentially in contact. Additionally, since no force is permitted during the period when contact is being broken, there is the implicit requirement that take-off exactly coincide with one of the discrete time intervals. Traditional optimal control approaches allow the overall duration of the trajectory to change, but not the individual time steps. This results in an overly restrictive optimization problem that may exclude desirable trajectories. One feasible solution is to create decision variables that divide each time step h into two periods. This can alternatively be expressed as having individual time steps h_k with pairwise constraints:

$$h_{2j-1} + h_{2j} = h_{2k+1} + h_{2j+2}, \quad j = 1, \dots, \lfloor \frac{N-3}{2} \rfloor \quad (22)$$

In practice, these additional free parameters are useful in expanding the space of feasible solutions while still allowing for relatively large time steps. Unlike the tools designed for multi-body simulation, we still do not need to explicitly solve for the zero-crossings when two bodies make and break contact[1]. Additionally, we are generally interested in problems where a robot is interacting with a limited set of environmental surfaces like the ground or an object for manipulation. In these cases, we avoid the combinatorial complexity of the number of potential contacts that might necessitate such a design.

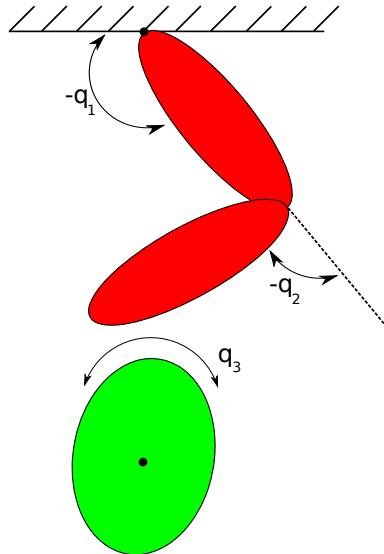
Since both state and constraint forces are solved implicitly, this program has a relatively large number of decision variables and constraints. However, as is typical in direct methods, this resulting NP is generally sparse and so is suitable for implementation with SNOPT[7].

3 Example Applications

3.1 Finger Contact

Recent research by Tassa and Todorov has explored the possibility of using stochastic complementarity with optimal control[18]. In this work, they use a Dynamic Programming based approach to find a trajectory for the sample problem of a two link manipulator that must spin an ellipse. This is a simple example with three

Fig. 4 The finger, shown in red, is fully actuated and makes contact with the third ellipse to drive it about its axis. Here, $\phi(q)$ is the shortest distance between the distal finger and the free ellipse.



degrees of freedom and only one contact point where the optimal mode schedule is immediately obvious. However, it provided an early test for our methods. We constrained the system to start from rest, $q_1 = 0, q_2 = 0, q_3 = 0$, and optimized for a quadratic cost on control input and velocity of the free ellipse:

$$g(x, u) = \sum_{k=1}^N \dot{q}_3^T Q \dot{q}_3 + u^T R u \quad (23)$$

The parameters for size and mass and for the cost function were chosen to directly parallel the previous work by Tassa and Todorov. As we hoped, our approach succeeded in quickly finding a locally optimal trajectory. For $N = 20$, convergence is reached in roughly 30 seconds. As we increased the overall duration of the trajectory, the optimization process found an increasing number of flicking motions where, after making contact, it drew the finger back up to make another pass. Additionally, Tassa and Todorov note that the effect of gravity was required to pull the manipulator into contact with the ellipse in order for the optimization process to discover the possibility of contact. Our approach does not have this limitation. If we eliminate gravity from the system, even given an initial trajectory that starts at rest with $u(t) = 0$, our methods successfully initiate contact between the manipulator and ellipse. Both of these results speak to the capability of our algorithm to actively identify a mode schedule that is not forced by the initial tape or the system's dynamics.

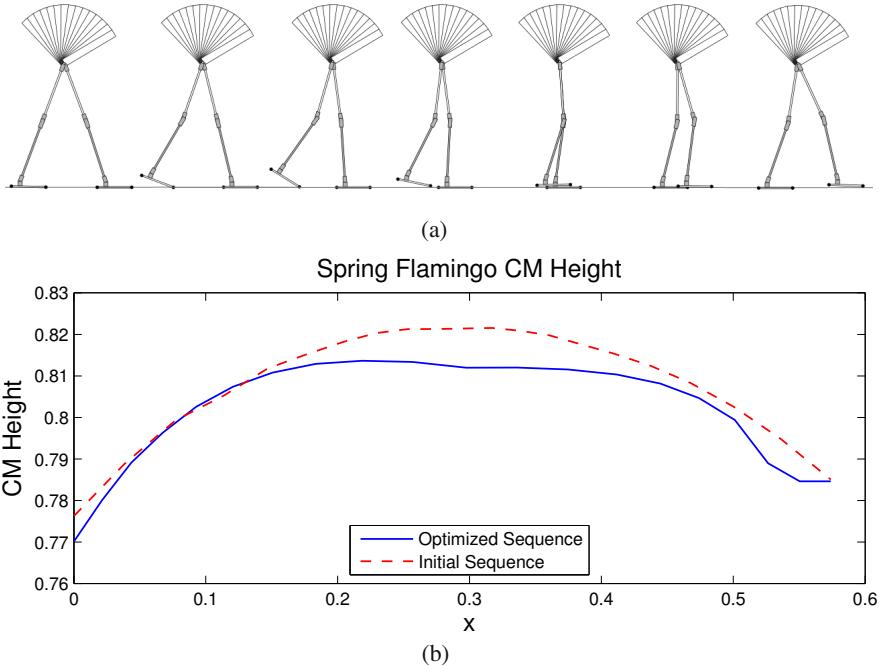


Fig. 5 (a) A walking gait for Spring Flamingo that minimizes mechanical cost of transport. To generate this trajectory, the height of the swing foot was not considered, so the solution is a minimalist trajectory with very little ground clearance. (b) The height of the center of mass over the optimized and initial trajectories is plotted over the sequence. The optimal trajectory minimizes unnecessary vertical motion of the robot.

3.2 Spring Flamingo Walking Gait

To analyze a more realistic system, we tested our methods on a planar simulation of the Spring Flamingo robot[13]. On Spring Flamingo, each leg has three actuated joints (hip, knee, and ankle) and there are contact points at the toe and heel of each foot. Many hybrid walking models use a constrained form of the dynamics, where a foot in contact with the ground is treated as a pin joint. Here, however, we deal with the full constrained dynamics where the body of the robot is modeled as a *floating base* parameterized by the variables (x, y, θ) which represent the planar position and pitch of the robot. Periodic constraints were used to generate a cyclic walking gait and the trajectory was optimized for mechanical cost of transport. Cost of transport is a good, unitless indication of the energy consumption required for locomotion. Where d is the total distance traveled, we write the cost as:

$$g(x, u) = \frac{1}{mgd} \sum_{k=1}^N \sum_i |\dot{q}_{k,i} u_{k,i}| \quad (24)$$

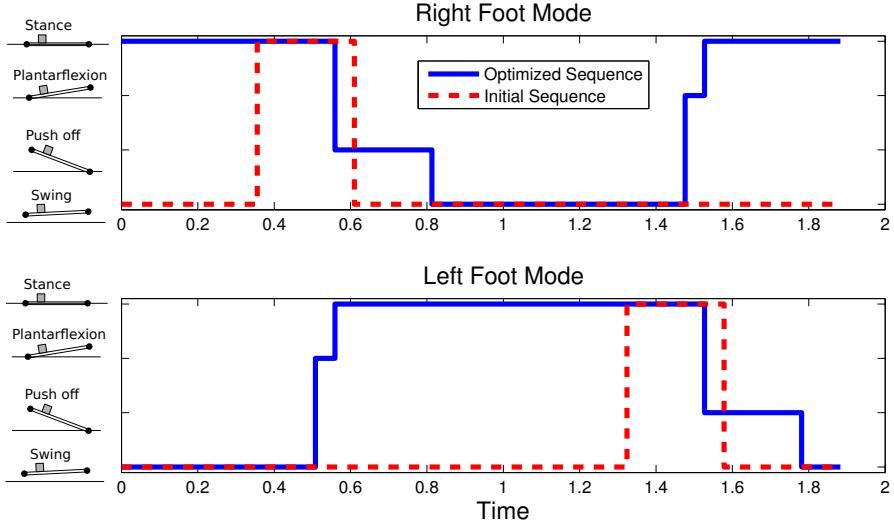


Fig. 6 The optimized mode sequence of the left and right feet is plotted against time and the mode transitions are labeled. The SQP was initialized with the significantly different sequence, demonstrating the ability of the algorithm to independently plan through contact discontinuities. Note that, in this case, the locally optimal trajectory has distinct heel strike and heel off events.

Since the solutions to the NP are local, our methods discovered a wide variety of feasible gaits that satisfied the general constraints dependent on the initial condition set. For instance, given the task of finding a periodic gait that travels a specific distance, hopping motions and gaits with relatively short or long strides are possible local solutions. In particular, the input λ tape implicitly identifies the mode sequence of the initial guess. However, the solution is not restricted to the given ordering. Figure 6 shows the initial and optimized mode sequences of a particular Spring Flamingo gait. Here, the initial tape leads the trajectory to a right-left walking gait but details such as independent heel strike and heel off were identified in the optimization process. To optimize for a cyclic gait, it is natural to write the periodicity constraint:

$$x_N = x_1 \quad (25)$$

Since we would prefer to search over the smaller space of a half gait, and we wish the robot to walk a minimum distance, we reformulate the periodic requirement to account for symmetry and add a unilateral constraint on stride length. Where q_l and q_r are the left and right joints, respectively, and d_{min} is the minimum stride length, we have:

$$\begin{bmatrix} y_N \\ \theta_N \\ \dot{x}_N \\ \dot{y}_N \\ \dot{\theta}_N \\ q_{l,N} \\ q_{r,N} \\ \dot{q}_{l,N} \\ \dot{q}_{r,N} \end{bmatrix} = \begin{bmatrix} y_1 \\ \theta_1 \\ \dot{x}_1 \\ \dot{y}_1 \\ \dot{\theta}_1 \\ q_{r,1} \\ q_{l,1} \\ \dot{q}_{r,1} \\ \dot{q}_{l,1} \end{bmatrix} \quad (26)$$

$$x_N \geq x_1 + d_{min} \quad (27)$$

With these linear constraints and given a nominal trajectory from Pratt's original work on the robot where the cost of transport was 0.18[12], our methods identified a periodic walking gait which reduced the cost to 0.04. This is a significant reduction in cost and is impressive in its own right, especially for a system with no passive elements to store and release energy. Figure 5(a) shows the optimized walking gait and the height of the center of mass (CM) throughout the trajectory, compared with that of the nominal gait. The optimal trajectory minimizes wasteful up and down motion of the CM. Note also that the foot swing height is very low to minimize any velocity at impact.

3.3 FastRunner Gait

The research behind this paper was motivated by the challenges posed by the FastRunner platform shown in Figure 1. For the previous examples, it is certainly possible to identify a desired mode sequence. This is a difficult task, however, for a system like FastRunner. A planar model of the robot has 13 degrees of freedom, including three articulated toe segments on each foot that can make or break contact with the ground. Additionally, there are a total of 16 unilateral joint limits, many of which are designed to be used while running at high speed. Scheduling the order of these contacts and joint limits is not practical.

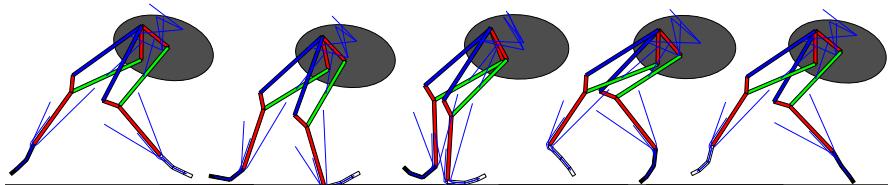


Fig. 7 A generated trajectory for the FastRunner robot running at over 20 mph. The solid elements show the leg linkages and the blue lines indicate springs and tendons. Only the hip joints of the robot are actuated.

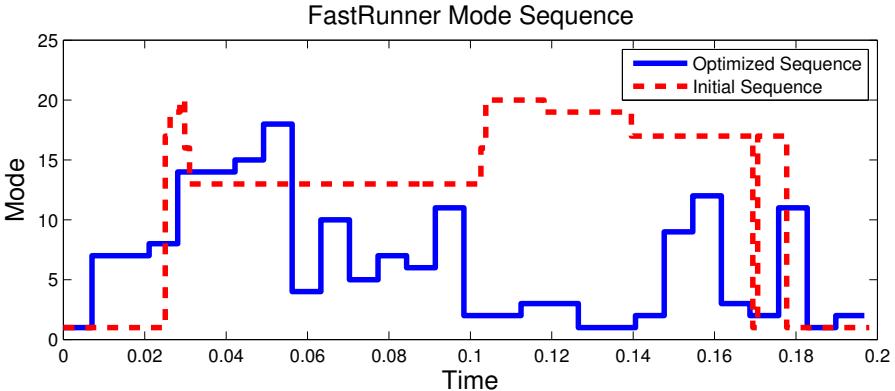


Fig. 8 Out of more than 4 million possible discrete modes, the sequence for one locally optimal cyclic trajectory is shown. This sequence, which passes through 15 different modes, is compressed and plotted above against the largely distinct mode sequence of the initial trajectory.

Figure 7 shows a motion sequence of an optimized periodic running gait, averaging over 20 mph. As with Spring Flamingo, constraints (26) and (27) restricted the search space and this trajectory was optimized for mechanical cost of transport. Both the leg linkages and passive elements like springs and tendons are shown in the figure. The complexity of the system and the stiffness of some of the springs posed additional problems for the optimization. In this case, additional linear constraints were useful in guiding the NP solver away from poorly conditioned or infeasible regions. This is typical for SQP methods, where the program can be difficult to solve if the local QP is a poor estimate of the true nonlinear program.

With 22 discrete variables, there are over 4 million possible discrete modes for the FastRunner robot where each mode has a unique system of continuous dynamical equations. One possible mode sequence discovered by the optimization process is illustrated in Figure 8. The individual mode transitions shown occur when the contact state of one of the toes changes or when a joint limit becomes active or inactive. While the states of individual discrete variables, such as toe contacts, may overlap between the initial and optimal trajectories, the aggregate discrete states show almost no agreement. This speaks to the combinatorial complexity of planning a mode schedule for a system like FastRunner. Despite the complexity of the system, by taking advantage of the NCP constraint formulation, our methods are now able to generate a locally optimal gait for FastRunner.

4 Discussion and Future Work

In our approach, we write a complex NP that in practice has been tractable for state of the art solvers. Since the problem is non-convex, we are limited to locally optimal solutions. As is typical for non-convex problems, applying linear constraints

on the decision variables to steer the solver away from singularities or other poorly conditioned regions can be critical to finding a desirable solution. In the examples above, this is typically done by eliminating obviously undesirable or infeasible regions of the joint space. However, we have also found that certain relaxations of the complementarity conditions can greatly improve the rate of convergence and reduce the likelihood of a poor, local solution. In particular, the simple relaxation $\phi(q)\lambda \leq \alpha$ works well in practice. This has the intuitive effect of smoothing out the complementarity conditions by allowing force to be applied at a distance during early phases of the algorithm. This additionally grows the feasible state space of the NP to include regions where the complementarity gradient is more useful to the solver. As α is driven to 0 over a few passes, the slack in the constraint helps allow the solver to converge to a better trajectory by improving the conditioning of the SQP. Many smoothing functions for NPs exist and have been used to directly solve these problems, such as the Fischer-Burmeister function [6] or the class of functions suggested in [4], and these functions may be applicable here as well.

As is mentioned above, throughout this paper we have solely dealt with the case of the robot interacting with a single object in the environment. While this is not a hard constraint, an increase in the number of interacting bodies will lead to a potentially combinatorial number of contacts and corresponding contact forces. Of course, this also implies $O(\exp(n^2))$ potential hybrid modes. We have also focused on purely inelastic collisions where the effective coefficient of restitution vanishes. The works in [1, 11] have developed LCP based simulation tools for multi-body contact and elastic collisions, and we believe our methods should extend to these areas as well.

Future work will also include extension of the methods described here to stabilize about the planned trajectory with a form of Model Predictive Control (MPC). We believe that, given a nominal trajectory, this approach will solve the problem of real-time local control as a convex optimization program and that the controller will be capable of planning a finite-horizon path with a different mode sequence than that of the nominal trajectory.

5 Conclusion

To control highly nonlinear robotic systems through real-world environments, it is critical that we be able to generate feasible, high quality trajectories. Current, state of the art techniques struggle when presented with complex systems where the hybrid sequence is difficult to intuit. Here, we have presented a method for trajectory optimization through the discontinuities of contact that does not rely on *a priori* specification of a mode schedule. Our approach combines traditional, direct local control approaches with an complementarity based contact model into a single Nonlinear Program. By writing the dynamics and constraints without reference to explicit hybrid modes, we are now able to easily plan through the discontinuous dynamics of contact. Additionally, unlike with other methods, we do not require arbitrary or hand-tuned parameters nor do we rely on the passive system dynamics to generate a

mode schedule. Once convergence is reached, the solution strictly satisfies all contact and dynamics constraints. We applied our method to three different systems, including the high dimensional FastRunner robot where we generated a high speed running gait. Future efforts will be focused on improving the convergence properties of this algorithm and extending it to real-time trajectory stabilization.

Acknowledgments. This work was supported by the DARPA Maximum Mobility and Manipulation program, BAA-10-65-M3-FP-024.

References

1. Anitescu, M., Potra, F.A.: Formulating dynamic multi-rigid-body contact problems with friction as solvable linear complementarity problems. *Nonlinear Dynamics* 14(3), 231–247 (1997)
2. Betts, J.T.: Practical Methods for Optimal Control Using Nonlinear Programming. SIAM Advances in Design and Control. Society for Industrial and Applied Mathematics (2001)
3. Byl, K., Tedrake, R.: Approximate optimal control of the compass gait on rough terrain. In: Proc. IEEE International Conference on Robotics and Automation (ICRA) (2008)
4. Chen, C., Mangasarian, O.L.: A class of smoothing functions for nonlinear and mixed complementarity problems. *Computational Optimization and Applications* 5(2), 97–138 (1996)
5. Cotton, S., Olaru, I., Bellman, M., van der Ven, T., Godowski, J., Pratt, J.: Fastrunner: A fast, efficient and robust bipedal robot. concept and planar simulation. In: Proceeding of the IEEE International Conference on Robotics and Automation (ICRA) (2012)
6. Fischer, A.: A special newton-type optimization method. *Optimization* 24(3-4), 269–284 (1992)
7. Gill, P.E., Murray, W., Saunders, M.A.: SNOPT: An SQP Algorithm for Large-Scale Constrained Optimization. *SIAM Review* 47(1), 99–131 (2005)
8. Gill, P.E., Murray, W., Saunders, M.A.: User's Guide for SNOPT Version 7: Software for Large -Scale Nonlinear Programming, February 12 (2006)
9. Hargraves, C.R., Paris, S.W.: Direct Trajectory Optimization Using Nonlinear Programming and Collocation. *J. Guidance* 10(4), 338–342 (1987)
10. Koenig, N., Howard, A.: Design and use paradigms for gazebo, an open-source multi-robot simulator. In: Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2004), vol. 3, pp. 2149–2154. IEEE (2004)
11. Miller, A.T., Christensen, H.I.: Implementation of multi-rigid-body dynamics within a robotic grasping simulator. In: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2003), vol. 2, pp. 2262–2268. IEEE (2003)
12. Pratt, J.: Exploiting Inherent Robustness and Natural Dynamics in the Control of Bipedal Walking Robots. PhD thesis, Computer Science Department, Massachusetts Institute of Technology (2000)
13. Pratt, J., Pratt, G.: Intuitive Control of a Planar Bipedal Walking Robot. In: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA) (1998)
14. Schultz, G., Mombaur, K.: Modeling and Optimal Control of Human-Like Running. *IEEE/ASME Transactions on Mechatronics* 15(5), 783–792 (2010)
15. Shkolnik, A., Levashov, M., Manchester, I.R., Tedrake, R.: Bounding on rough terrain with the littledog robot. *The International Journal of Robotics Research (IJRR)* 30(2), 192–215 (2011)

16. Srinivasan, M., Ruina, A.: Computer optimization of a minimal biped model discovers walking and running. *Nature* 439, 72–75 (2006)
17. Stewart, D.E., Trinkle, J.C.: An implicit time-stepping scheme for rigid body dynamics with inelastic collisions and coulomb friction. *International Journal for Numerical Methods in Engineering* 39(15), 2673–2691 (1996)
18. Tassa, Y., Todorov, E.: Stochastic complementarity for local control of discontinuous dynamics. In: *Proceedings of Robotics: Science and Systems (RSS)*. Citeseer (2010)
19. Wampler, K., Popovic, Z.: Optimal gait and form for animal locomotion. *ACM Transactions on Graphics (TOG)* 28(3), 60 (2009)
20. Westervelt, E.R., Grizzle, J.W., Chevallereau, C., Choi, J.H., Morris, B.: *Feedback Control of Dynamic Bipedal Robot Locomotion*. CRC Press, Boca Raton (2007)

Robust Online Motion Planning with Regions of Finite Time Invariance

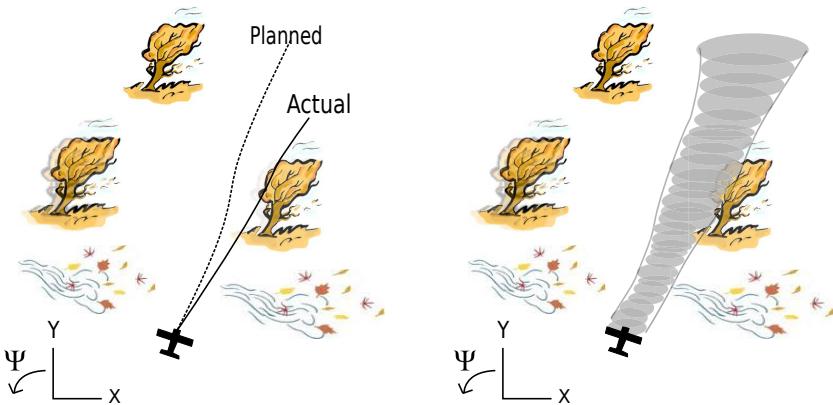
Anirudha Majumdar and Russ Tedrake

Abstract. In this paper we consider the problem of generating motion plans for a nonlinear dynamical system that are guaranteed to succeed despite uncertainty in the environment, parametric model uncertainty, disturbances, and/or errors in state estimation. Furthermore, we consider the case where these plans must be generated online, because constraints such as obstacles in the environment may not be known until they are perceived (with a noisy sensor) at runtime. Previous work on feedback motion planning for nonlinear systems was limited to offline planning due to the computational cost of safety verification. Here we take a *trajectory library* approach by designing controllers that stabilize the nominal trajectories in the library and pre-computing regions of finite time invariance (“funnels”) for the resulting closed loop system. We leverage sums-of-squares programming in order to efficiently compute funnels which take into account bounded disturbances and uncertainty. The resulting *funnel library* is then used to *sequentially compose* motion plans at runtime while ensuring the safety of the robot. A major advantage of the work presented here is that by explicitly taking into account the effect of uncertainty, the robot can evaluate motion plans based on how vulnerable they are to disturbances. We demonstrate our method on a simulation of a plane flying through a two dimensional forest of polygonal trees with parametric uncertainty and disturbances in the form of a bounded “cross-wind”.

1 Introduction

The ability to plan and execute dynamic motions under uncertainty is a critical skill with which we must endow our robots in order for them to perform useful tasks in the real world. Whether it is an unmanned aerial vehicle (UAV) flying at high speeds through a cluttered environment in the presence of wind gusts, a legged robot

Anirudha Majumdar · Russ Tedrake
Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge MA - 02139
e-mail: {anirudha, russt}@mit.edu



(a) A plane deviating from its nominal planned trajectory due to a heavy cross-wind. (b) The “funnel” of possible trajectories.

Fig. 1 Not accounting for uncertainty while planning motions can lead to disastrous consequences

traversing rough slippery terrain, or a micro-air vehicle with noisy on-board sensing, the inability to take into account disturbances, model uncertainty and state uncertainty can have disastrous consequences.

Motion planning has been the subject of significant research in the last few decades and has enjoyed a large degree of success in recent years. Planning algorithms like the Rapidly-exploring Randomized Tree (RRT) [12], RRT* [11], and related trajectory library approaches [13] [5] can handle large state space dimensions and complex differential constraints, and have been successfully demonstrated on a wide variety of hardware platforms [22] [21]. However, a significant failing is their inability to explicitly reason about uncertainty and feedback. Modeling errors, state uncertainty and disturbances can lead to failure if the system deviates from the planned nominal trajectories. A cartoon of the issue is sketched in Figure 1(a), where a UAV attempting to fly through a forest with a heavy cross-wind gets blown off its planned nominal trajectory and crashes into a tree.

More recently, planning algorithms which explicitly take into account feedback control have been proposed. LQR-Trees [24] and the minimum snap trajectory generation approach [16] operate by generating locally optimal trajectories through state space and stabilizing them locally. However, the former approach lacks the ability to handle scenarios in which the task and environment are unknown till run-time since it is too computationally intensive for online implementation. The latter technique has no mechanism for reasoning about uncertainty in the form of unmodeled disturbances, state errors and model uncertainty.

In this paper, we present a partial solution to these issues by combining trajectory libraries, feedback control, and sums-of-squares programming [18] in order to

perform robust motion planning in the face of uncertainty. In particular, in the offline computation stage, we design a finite library of motion primitives and augment them with feedback controllers that locally stabilize them. Then, using sums-of-squares programming, we compute robust regions of finite time invariance (“funnels”) around these trajectories that guarantee stability of the closed loop system in the presence of sensor noise, parametric model uncertainty, unmodeled bounded disturbances and changes in initial conditions. A cartoon of these regions of finite time invariance, or “funnels”, is shown in Figure 1(b). Finally, we provide a way of composing these robust motion plans online in order to operate in a provably safe manner.

One of the most important advantages that our approach affords us is the ability to choose between the motion primitives in our library in a way that takes into account the dynamic effects of uncertainty. Imagine a UAV flying through a forest that has to choose between two motion primitives: a highly dynamic roll maneuver that avoids the trees in front of the UAV by a large margin or a maneuver involves flying straight while avoiding the trees only by a small distance. A traditional approach that neglects the effects of disturbances and uncertainty may prefer the former maneuver since it avoid the trees by a large margin and is therefore “safer”. However, a more careful consideration of the two maneuvers leads to a different conclusion: the dynamic roll maneuver is far more susceptible to wind gusts and state uncertainty than the second one. Thus, it may be much more advantageous to execute the second motion primitive. Further, it may be possible that neither maneuver is guaranteed to succeed and it is safer to abort the mission and simply transition to a hover mode. Our approach allows robots to make these critical decisions, which are essential if robots are to move out of labs and operate in real-world environments.

2 Related Work

The motion planning aspect of our approach draws inspiration from the vast body of work that is focused on computing motion primitives in the form of trajectory libraries. For example, trajectory libraries have been used in diverse applications such as humanoid balance control [13], autonomous ground vehicle navigation [21] and grasping [4]. The Maneuver Automaton [5] attempts to capture the formal properties of trajectory libraries as a hybrid automaton, thus providing a nice unifying theoretical framework. Further theoretical investigations have focused on the offline generation of diverse but sparse trajectories that ensure the robot’s ability to perform the necessary tasks online in an efficient manner [6]. More recently, tools from sub-modular sequence optimization have been leveraged in the optimization of the sequence and content of trajectories evaluated online [4].

Robust motion planning has also been a very active area of research in the robotics community. Early work focused on the purely kinematic problem of planning paths through configuration space with “tubes” of specified radii around them such that all paths in the tube remained collision-free [9] [8]. Recent work has focused on reasoning more explicitly about the manner in which

uncertainty/ disturbances influence the dynamics of the robot, and is closer in spirit to the work presented here. In particular, [20] approaches the problem through dynamic programming on a model with disturbances by making use of the Maneuver Automaton framework mentioned earlier. However, the work does not take into account obstacles in the environment and does not provide or make use of any explicit guarantees on allowed deviations from the planned trajectories in the Maneuver Automaton. The authors of [27] attempt to address the robust motion planning problem through H^∞ control techniques. However, the results focused on a planar vehicle and the basic motion planning framework does not easily extend to three dimensional environments. Also, while regions of invariance are approximated analytically and numerically, they are used only to evaluate the performance of the controller and not while constructing motion plans. Another approach that is closely related to ours is Model Predictive Control with Tubes [15]. The idea is to solve the optimal control problem online with guaranteed “tubes” that the trajectories stay in. However, the method is limited to linear systems and convex constraints.

A critical component of the work presented here is the computation of regions of invariance for nonlinear systems via Lyapunov functions. This idea, along with the metaphor of a “funnel”, was introduced to the robotics community in [2], where funnels were *sequentially composed* in order to produce dynamic behaviors in a robot. In recent years, sums-of-squares programming has emerged as a way of checking the Lyapunov function conditions [18]. The technique relies on the ability to check positivity of multivariate polynomials by expressing them as a sum-of-squares. This can be written as a semi-definite optimization program and is amenable to efficient computational algorithms such as interior point methods [17]. Assuming polynomial dynamics of our system, one can check that a polynomial Lyapunov candidate, $V(x)$, satisfies $V(x) > 0$ and $\dot{V}(x) < 0$ in some region B_r . Importantly, the same idea can be used in computing regions of finite time invariance (“funnels”) around time-indexed trajectories of the system [25]. Further, robust regions of attraction can be computed using an approach that verifies stability/invariance of sets around fixed points under parametric uncertainty [26]. In this paper, we show how to combine these ideas to compute robust regions of finite-time invariance around *trajectories* that guarantee that the if the system starts off in the set of given initial conditions, it will remain in the computed “funnel” even if the model of the dynamics is uncertain and the system is subjected to bounded disturbances and state uncertainty.

3 Contributions

This paper makes two main contributions. First, we provide a way of generating regions of finite time invariance (“funnels”) for time-varying polynomial systems subjected to a general class of uncertainty (bounded uncertainty in parameters entering polynomially in the dynamics). This is an extension of the results presented in [25], which presents a method for computing “funnels” for systems with time-varying polynomial dynamics assuming no model uncertainty/disturbances. Second, we show how a library of such funnels can be precomputed offline and composed

together at runtime in a receding horizon manner while ensuring that the resulting closed loop system is “safe” (i.e. avoids obstacles and switches between the planned sequence of funnels). This can be viewed as an extension of the LQR-Trees algorithm [24] for feedback motion planning, which was limited to offline planning due to the relatively large computational cost of computing the funnels; our algorithm is suitable for real-time, online planning. We expect this framework to be useful in robotic tasks where the dynamics and perceptual system of the robot are difficult to model perfectly and for which the robot does not have access to the geometry of the environment till runtime.

4 Approach

A considerable amount of research effort in the motion planning community has focused on the design of trajectory libraries (Section 2). Further, a substantial but largely separate literature in control theory [19] [7] exists for the design of controllers that stabilize uncertain non-linear systems along nominal trajectories. Hence, in this section, we do not focus on the particulars of the generation of trajectory libraries and controllers. Rather, we concentrate on how these powerful methods from the fields of motion planning and control theory can be combined with sums-of-squares programming in order to perform robust online motion planning in the face of dynamic and state uncertainty. To this end, we assume that we are provided with a trajectory library consisting of a finite set of nominal feasible trajectories for the robot and a corresponding set of controllers that stabilize these trajectories. We discuss one particular method for obtaining trajectory libraries, controllers and Lyapunov functions in Section 5.

4.1 Robust Regions of Finite Time Invariance

The techniques for the computation of regions of finite time invariance presented in [25] can be extended to handle scenarios in which there is uncertainty in state and dynamics. This computation is similar to the one presented in [26], but instead of computing regions of attraction for asymptotically stable equilibrium points, we ask for certificates of finite time invariance around trajectories (“funnels”). Let $x_0(t) : [0, T] \mapsto \mathbb{R}^n$ be a trajectory in our trajectory library and $u(x, t)$ be the controller that stabilizes $x_0(t)$. Then, the closed loop dynamics of the system can be written as:

$$\dot{x} = f(x, t, w(t, x)) \quad (1)$$

where $w(t, x) \in W \subset \mathbb{R}^d$ is a free (but bounded) uncertainty term that can be used to model instantaneous disturbances, parametric model uncertainty, and state uncertainty.

We further assume that we have a time varying Lyapunov function, $V(x, t)$, that locally guarantees stability around the trajectory. An example of how such a candidate Lyapunov function may be obtained in practice is provided in Section 5.

Given a set of initial conditions $X_0 = \{x | V(x, t) \leq \rho(0)\}$, our goal is to compute a tight outer estimate of the set of states the system may evolve to under the closed loop dynamics and bounded uncertainty. In particular, we are concerned with finding a set of states $X_t = \{x | V(x, t) \leq \rho(t)\}$ for each time $t \in [0, T]$ such that:

$$x(0) \in X_0 \implies x(t) \in X_t, \forall t \in [0, T].$$

As described in [25], under mild technical conditions it is sufficient to find $\rho(t)$ such that:

$$V(x, t) = \rho(t) \implies \dot{V}(x, t, w) \leq \dot{\rho}(t), \forall w \in W. \quad (2)$$

As noted in [25], checking condition (2) on a finite set of time samples, t_i , results in large computational gains while still maintaining accuracy. Thus, assuming that the closed loop dynamics are polynomial in x and w for a given t , and that w belongs to a *bounded* semi-algebraic set $W = \{w | W_j(w) \geq 0\}$, we can write a sums-of-squares optimization program to compute $\rho(t_i)$ (and thus X_t):

$$\begin{aligned} & \underset{\rho, L_i, M_j}{\text{minimize}} \quad \sum_{t_i} \rho(t_i) \\ & \text{subject to} \quad \dot{\rho}(t_i) - \dot{V}(x, w) - L_i(x)[V(x) - \rho(t_i)] - \sum_j M_j(w)W_j(w) \geq 0, \forall t_i \\ & \quad M_j > 0, \quad \rho(t_i) \geq 0 \end{aligned} \quad (3)$$

It is easy to see that (3) is a sufficient condition for (2) since when $W_j(w) \geq 0$ and $V(x) = \rho(t_i)$, we have that $\dot{\rho}(t_i) - \dot{V}(x, w) \geq 0$. This optimization program is bilinear in the decision variables and is amenable to an alternating search over ρ and Lagrange multipliers (as in [25]). Note that the objective in our sums-of-squares program, $\sum_{t_i} \rho(t_i)$, helps us find a *tight* conservative estimate of the set of states the closed loop system may evolve to under the given uncertain dynamics. Further, one can very easily augment this optimization program to handle actuator saturations in a manner similar to the one described in [24].

4.2 Funnel Libraries

The tools from Section 4.1 can be used to create libraries of funnels offline. Given a trajectory library, \mathcal{T} , consisting of finitely many trajectories $x_i(t)$, locally stabilizing controllers $u_i(x, t)$, and associated candidate Lyapunov functions $V_i(x, t)$, we can compute a robust funnel for each trajectory in \mathcal{T} . However, there is an important issue that needs to be addressed when designing libraries of funnels and has an analogy in the traditional trajectory library approach. In particular, trajectories in a traditional trajectory library need to be designed in a way that allows them to be sequenced together. More formally, let \mathcal{P} denote the projection operator that projects a state, x , onto the subspace formed by the non-cyclic dimensions of the system (i.e. the dimensions with respect to which the Lagrangian of the system is

not invariant). Then, for two trajectories $x_i(t)$ and $x_j(t)$ to be executed one after another, we must have

$$\mathcal{P}(x_i(T_i)) = \mathcal{P}(x_j(0)).$$

Note that the cyclic coordinates do not pose a problem since one can simply “shift” trajectories around in these dimensions. This issue is discussed thoroughly in [5] and is addressed by having a *trim trajectory* of the system that other trajectories (*maneuvers*) start from and end at (of course, one may also have more than one *trim trajectory*).

In the case of *funnel* libraries, however, it is neither necessary nor sufficient for the nominal trajectories to line up in the non-cyclic coordinates. It is the *interface* between funnels that is important. Let $x_i(t)$ and $x_j(t)$ be two nominal trajectories in our library and $F_i(t) = \{x | V_i(x, t) \leq \rho_i(t)\}$ and $F_j(t) = \{x | V_j(x, t) \leq \rho_j(t)\}$ be the corresponding funnels. Further, we write $x = [x_c, x_{nc}]$, where x_c represent the cyclic dimensions and x_{nc} the non-cyclic ones. Following Burridge and Koditschek [2], we say that two funnels are *sequentially composable* if:

$$\begin{aligned} \mathcal{P}(F_i(T_i)) &\subset \mathcal{P}(F_j(0)) \\ \iff \forall x = [x_c, x_{nc}] \in F_i(T_i), \quad \exists x_{0,c} \text{ s.t. } [x_{0,c}, x_{nc}] \in F_j(0) \end{aligned} \tag{4}$$

While (4) is a necessary and sufficient condition for two funnels to be executed one after another, the dependence of $x_{0,c}$ on x makes searching for $x_{0,c}$ a non-convex problem in general. Thus, we set $x_{0,c}$ to be the cyclic coordinates of $x_j(0)$, resulting in a stronger sufficient condition that can be checked via a sums-of-squares program:

$$\forall x = [x_c, x_{nc}] \in F_i(T_i), \quad [x_{0,c}, x_{nc}] \in F_j(0) \tag{5}$$

Intuitively, (5) corresponds to “shifting” the inlet of funnel F_j along the cyclic dimensions so it lines up with x_c . Assuming $F_i(T_i)$ and $F_j(0)$ are semi-algebraic sets, a simple sums-of-squares feasibility program can be used to check this condition:

$$\begin{aligned} \rho_j(0) - V_j(\mathcal{P}([x_j(0), x_{nc}], 0)) + L(x)[V_i([x_c, x_{nc}], T_i) - \rho_i(T_i)] &\geq 0 \\ L(x) &\geq 0 \end{aligned} \tag{6}$$

where $L(x)$ is a non-negative Lagrange multiplier. Note that not all pairs of funnels in the library will be *sequentially composable* in general. Thus, as we discuss in Section 4.3, we must be careful to ensure *sequential composability* when planning sequences of funnels online.

4.3 Online Planning with Funnels

Having computed libraries of funnels in the offline pre-computation stage, we can proceed to use these primitives to perform robust motion planning online. The robot’s task specification may be in terms of a goal region that must be reached (as in the case of a manipulator arm grasping an object), or in terms of a nominal

direction the robot should move in while avoiding obstacles (as in the case of a UAV flying through a forest or a legged robot walking over rough terrain). For the sake of concreteness, we adopt the latter task specification although one can easily adapt the contents of this section to the former specification. We further assume that the robot is provided with polytopic regions in configuration space that obstacles are guaranteed to lie in and that the robot's sensors only provide this information up to a finite (but receding) spatial horizon. Our task is to sequentially compose funnels from our library in a way that avoids obstacles while moving forwards in the nominal direction. The finite horizon of the robot's sensors along with the computational power at our disposal determines how long the sequence of planned funnels can be at any given time.

The most important computation that needs to be performed at runtime is to check whether a given funnel intersects an obstacle. For the important case in which our Lyapunov functions are quadratic in x , this computation is a Quadratic Program (QP) and can be solved very efficiently (as evidenced by the success of Model Predictive Control [3]). Let the funnel be given by the sets $X_{t_i} = \{x | V(x, t_i) \leq \rho(t_i)\}$ for $i = 1, \dots, N$, and a particular obstacle defined by half-plane constraints $A_j x \geq 0$ for $j = 1, \dots, M$. Note that A_j will typically be sparse since it will contain zeros in places corresponding to non-configuration space variables (like velocities). Then, for $i = 1, \dots, N$, we solve the following QP:

$$\begin{aligned} & \underset{x}{\text{minimize}} && V(x, t_i) \\ & \text{subject to} && A_j x \geq 0, \forall j \end{aligned} \tag{7}$$

Denoting the solution of (7) for a given t_i as $V^*(x^*, t_i)$, the funnel does not intersect the obstacle if and only if $V^*(x^*, t_i) \geq \rho(t_i), \forall t_i$. Multiple obstacles are handled by simply solving (7) for each obstacle. An important point that should be noted is that we do not require the obstacle regions to be convex. It is only required that they are represented as unions of convex polytopic sets.

For higher order polynomial Lyapunov functions, the following sums-of-squares conditions need to be checked for all t_i :

$$\begin{aligned} & V(x, t_i) - \rho(t_i) - L_i(x) \sum_j A_j x \geq 0 \\ & L_i(x) \geq 0 \end{aligned} \tag{8}$$

However, these provide only sufficient conditions for non-collision. Thus, if the conditions in (8) are met, one is guaranteed that there is no intersection with the obstacle. The converse is not true in general. Further, depending on the state space dimension of the robot, this optimization problem may be computationally expensive to solve online. Hence, for tasks in which online execution speed is crucial, one may need to restrict oneself to quadratic Lyapunov functions. However, since most popular control design techniques like the Linear Quadratic Regulator, H^∞ and LMI methods ([19] [7]) provide quadratic Lyapunov function candidates, the loss is not substantial.

Algorithm 1 provides a sketch of the online planning loop. At every control cycle, the robot updates its state in the world along with the obstacle positions. It then checks to see if the sequence of funnels it is currently executing may lead to a collision with an obstacle (which should only be the case if the sensors report new obstacles). If so, it replans a sequence of funnels that can be executed from its current state and don't lead to any collisions. The $\text{ReplanFunnels}(x, \mathcal{O})$ subroutine assumes that funnel sequences that are *sequentially composable* in the sense of Section 4.2 have been ordered by preference during the pre-computation stage. For example, sequences may be ordered by the amount they make the robot progress in its nominal direction for a navigation task. $\text{ReplanFunnels}(x, \mathcal{O})$ goes through funnel sequences and checks two things. First, it checks that its current state is contained in the first funnel in the sequence (after appropriately shifting the funnel in the cyclic dimensions). Second, it checks that the sequence leads to no collisions with obstacles. The algorithm returns the first sequence of funnels that satisfies both criteria. Finally, the online planing loop computes which funnel of the current plan it is in and applies the corresponding control input $u_i(x, t.\text{internal})$.

Of course, several variations on Algorithm 1 are possible. In practice, it may not be necessary to consider re-planning at the frequency of the control loop. Instead, longer sections of the plan may be executed before re-planning. Also, instead of choosing the most “preferred” collision-free sequence of funnels, one could choose the sequence that maximizes the minimum over t_i of $\frac{V^*(x^*, t_i)}{\rho(t_i)}$. As before, $V^*(x^*, t_i)$ is the solution of the QP (7). Since the 1-sublevel set of $\frac{V(x, t_i)}{\rho(t_i)}$ corresponds to the funnel, maximizing this is a reasonable choice for choosing sequences of funnels.

Algorithm 1. Online Planning

```

1: Initialize current planned funnel sequence,  $\mathcal{P} = \{F_1, F_2, \dots, F_n\}$ 
2: for  $t = 0, \dots$  do
3:    $\mathcal{O} \leftarrow$  Obstacles in sensor horizon
4:    $x \leftarrow$  Current state of robot
5:   Collision  $\leftarrow$  Check if  $\mathcal{P}$  collides with  $\mathcal{O}$  by solving QPs (7)
6:   if Collision then
7:      $\mathcal{P} \leftarrow \text{ReplanFunnels}(x, \mathcal{O})$ 
8:   end if
9:    $F.\text{current} \leftarrow F_i \in \mathcal{P}$  such that  $x \in F_i$ 
10:   $t.\text{internal} \leftarrow$  Internal time of  $F.\text{current}$ 
11:  Apply control  $u_i(x, t.\text{internal})$ 
12: end for

```

5 Example

We demonstrate our approach on a model of an aircraft flying in two dimensions through a forest of polygonal trees. A pictorial depiction of the model is provided in

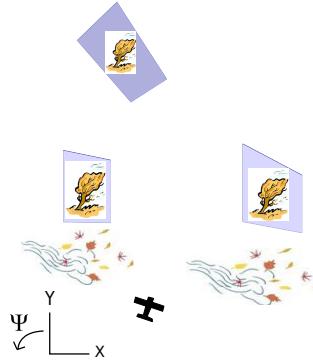


Fig. 2 Visualization of the system showing the coordinate system, polygonal obstacles, and “cross-wind”

Figure 2. The aircraft is constrained to move at a fixed forward speed and can control the second derivative of its yaw angle. We introduce uncertainty into the model by assuming that the speed of the plane is uncertain and time-varying and that there is a time-varying “cross-wind” whose magnitude is instantaneously bounded. The full non-linear dynamics of the system are then given by:

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ \psi \\ \dot{\psi} \end{bmatrix}, \quad \dot{\mathbf{x}} = \begin{bmatrix} -v(t) \cos \psi \\ v(t) \sin \psi \\ \dot{\psi} \\ u \end{bmatrix} + \begin{bmatrix} w(t) \\ 0 \\ 0 \\ 0 \end{bmatrix}. \quad (9)$$

with the speed of the plane $v(t) \in [9.5, 10.5]m/s$ and cross-wind $w(t) \in [-0.3, 0.3m/s]$.

The plane’s trajectory library, \mathcal{T} , consists of 11 trajectories and is shown in Figure 3(a). The trajectories $x_i(t) : [0, T_i] \mapsto \mathbb{R}^4$ and the corresponding nominal open-loop control inputs $u_i(t)$ were obtained via the direct collocation trajectory optimization method [1] by constraining $x_i(0)$ and $x_i(T_i)$ and locally minimizing a cost of the form:

$$J = \int_0^{T_i} [1 + u_0(t)^T R(t) u_0(t)] dt.$$

Here, R is a positive-definite cost matrix. For each $x_i(t)$ in \mathcal{T} we design a H^∞ controller that locally stabilizes the system. This is done by first computing a time-varying linearization of the dynamics and disturbances about the trajectory:

$$\dot{\bar{x}} \approx A_i(t)\bar{x}(t) + B_i(t)\bar{u}(t) + D_i(t)w(t)$$

where $\bar{x} = x - x_i(t)$ and $\bar{u} = u - u_i(t)$. The optimal linear control law, $\bar{u}^*(x, t) = -R^{-1}B_i(t)^T S_i(t)\bar{x}$ is obtained by solving the Generalized Riccati Differential Equation:

$$-\dot{S}_i(t) = Q + S_i(t)A_i(t) + A_i(t)^T S_i(t) - S_i(t)[B_i(t)R^{-1}B_i(t)^T - \frac{1}{\gamma^2}D_i(t)D_i(t)^T]S_i(t)$$

with final value conditions $S_i(T) = S_{T,i}$. The quadratic functional:

$$V_i(x, t) = (x - x_i(t))^T S_i(t)(x - x_i(t))$$

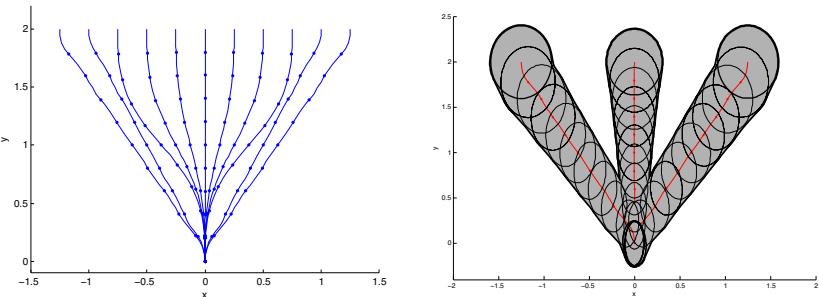
is our locally valid Lyapunov candidate. Then, as described in Section 4.1, we use $V_i(x, t)$ to compute robust regions of finite-time invariance (“funnels”). A few implementation details are worth noting here. Similar to [24], we perform the verification on the time-varying nonlinear system by taking third-order Taylor-approximations of the dynamics about the nominal trajectories. Secondly, the set of possible initial conditions for a given funnel, $X_0 = \{x|V(x, t) \leq \rho(0)\}$ needs to be specified. Since the Riccati equation does not allow us to fix $S(0)$ directly, we can only affect the shape of the initial ellipsoidal region by scaling $\rho(0)$. In order to choose a physically meaningful set of initial conditions we specify a desired ellipsoidal set $X_{0,des} = \{x|V_{des}(x) \leq 1\}$, and then choose the smallest $\rho(0)$ such that $X_{0,des} \subset X_0$. The following sums-of-squares program can be solved to obtain such a $\rho(0)$:

$$\begin{aligned} & \underset{\rho(0), \tau}{\text{minimize}} \quad \rho(0) \\ & \text{subject to} \quad \rho(0) - V(x, 0) + \tau(V_{des}(x) - 1) \geq 0 \\ & \quad \tau \geq 0 \end{aligned} \tag{10}$$

Three of the funnels in our library are shown in Figure 3(b). Note that the funnels have been projected down from the original four dimensional state space to the x-y plane for the sake of visualization.

Figure 5 demonstrates the use of the online planning algorithm in Section 4.3. The plane plans two funnels in advance while nominally attempting to fly in the y-direction and avoiding obstacles. The projection of the full sequence of funnels executed by the plane is shown in the figure. Figures 4(a) and 4(b) show the plane flying through the same forest with identical initial conditions. The only difference is that the cross-wind term is biased in different directions. In Figure 4(a), the cross-wind is primarily blowing towards the right, while in Figure 4(b), the cross-wind is biased towards the left. Of course, the plane is not directly aware of this difference, but ends up following different paths around the obstacles as it is buffeted around by the wind.

Finally, we demonstrate the utility of explicitly taking into account uncertainty in Figure 5. There are two obstacles in front of the plane. The two options available to the plane are to fly straight in between the obstacles or to bank right and attempt to go around them. If the motion planner didn’t take uncertainty into account and



(a) Trajectory library consisting of 11 locally optimal trajectories.
(b) Three funnels in our funnel library projected onto the x-y plane.

Fig. 3 Trajectory and funnel libraries for the plane

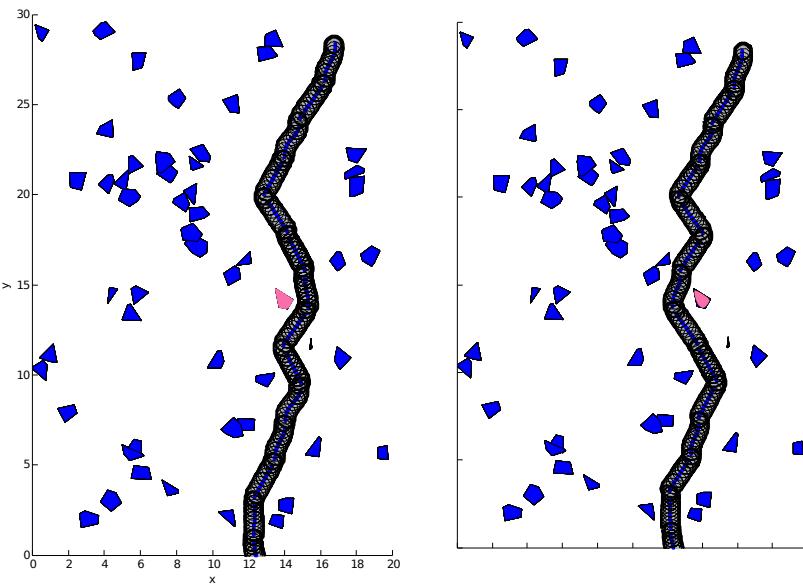


Fig. 4 Robust online planning through a forest of polygonal obstacles. The two subfigures show the plane flying through the same forest, but with the cross-wind biased in different directions. The eventual path through the forest is different, as the plane goes right around the marked obstacle in (a) and left in (b).

chose the one that maximized the average distance to the obstacles, it would choose the trajectory that banks right and goes around the obstacles. However, taking the funnels into account leads to a different decision: going straight in between the obstacles is safer even though the distance to the obstacles is smaller.

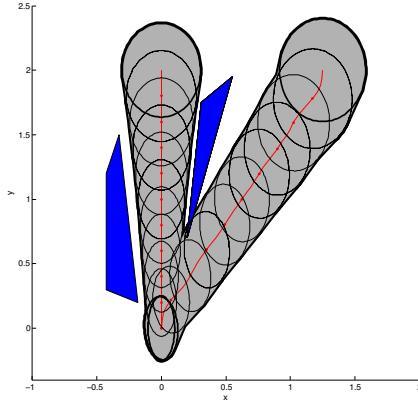


Fig. 5 This figure shows the utility of explicitly taking uncertainty into account while planning. The intuitively more “risky” strategy of flying in between two closely spaced obstacles is guaranteed to be safe, while the path that avoids going in between obstacles is less robust to uncertainty and could lead to a collision.

6 Discussion and Future Work

6.1 Stochastic Verification

Throughout this paper, we have assumed that all disturbances and uncertainty are bounded with probability one. In practice, this assumption may either not be fully valid or could lead to over-conservative performance. In such situations, it is more natural to provide guarantees of finite time invariance of a probabilistic nature. Recently, results from classical super-martingale theory have been combined with sums-of-squares programming in order to compute such probabilistic certificates of finite time invariance [23]. i.e. provide upper bounds on the probability that a stochastic nonlinear system will leave a given region of state space. The results presented in [23] can be directly combined with the approach presented in this work to perform robust online planning on stochastic systems and will be the subject of future work.

6.2 Continuously Parameterized Families of Funnels

As discussed in Section 4.2, we are currently partially exploiting invariances in the dynamics by shifting trajectories (and corresponding funnels) that we want to execute next in the cyclic coordinates so they line up with the cyclic coordinates of the robot’s current state. In our example from Section 5, this simply corresponds to translating and rotating funnels so the beginning of the next trajectory lines up with the current state’s x,y and yaw. However, we could further exploit invariances in the dynamics by shifting funnels around locally to ensure that they don’t

intersect an obstacle while still maintaining the current state inside the funnel. One can then think of the nominal trajectories and funnels being *continuously parameterized* by shifts in the cyclic coordinates. Interestingly, it is also possible to use sums-of-squares programming to compute conservative funnels for cases in which one shifts the nominal trajectory in the *non-cyclic* coordinates [14]. Thus, one could potentially significantly improve the richness of the funnel library by pre-computing continuously parameterized funnel libraries instead of just a finite family. However, choosing the right “shift” to apply at runtime is generally a non-convex problem (since the free-space of the robot’s environment is non-convex) and thus one can only hope to find “shifts” that are locally optimal.

6.3 Sequence Optimization for Large Funnel Libraries

For extremely large funnel libraries, it may be computationally difficult to search all the funnels while planning online. This is a problem that traditional trajectory libraries face too [4]. Advances in submodular sequence optimization were leveraged in [4] to address this issue. The approach involves limiting the set of trajectories that are evaluated online and optimizing the sequence in which trajectories are evaluated. Guarantees are provided on the sub-optimality of the resulting strategy. This technique could be adapted to work in our framework too and will be addressed in future work.

6.4 Designing and Evaluating Trajectory Libraries

As discussed in Section 2, the design of trajectory libraries has been the subject of a large body of work in motion planning. One of the most exciting recent results is presented in [10] in which percolation theory is used to provide mathematical bounds on the speed at which a robot can safely fly through a forest. Assuming that the probability distribution that gave rise to the forest is ergodic and under a few technical conditions on the dynamics of the robot, it is shown that there exists a critical speed beyond which the robot cannot fly forever without collisions. If the robot flies below this speed, there exists an infinite path through the forest. We believe that the techniques used in [10] can be used to *evaluate and compare* funnel libraries too. In the setting where the distribution of the obstacle generating process is known and is ergodic, we can compute the probability that there exists a sequence of funnels that can be chained together to traverse the forest without colliding with a tree. Intuitively, the idea is to consider all possible funnel sequences through the forest and then computing the probability that there exists a particular sequence with no tree lying inside any funnel in that sequence.

This may provide an important extension to the approach presented in this paper since it allows us to directly deal with *uncertainty in the nature of the environment* too. It also provides us with a meaningful way to evaluate whether a given funnel library is “good enough” or whether more primitives are required in order to have a high chance of success.

7 Conclusion

In this paper, we have presented an approach for motion planning in *a priori* unknown environments with dynamic uncertainty in the form of bounded parametric model uncertainty, disturbances, and state errors. The method augments the traditional trajectory library approach by constructing stabilizing controllers around the nominal trajectories in a library and computing robust regions of finite time invariance (“funnels”) for the resulting closed loop controllers via sums-of-squares programming. The pre-computed set of funnels is then used to plan online by *sequentially composing* them together in a manner that ensures obstacles are avoided. By explicitly taking into account uncertainty and disturbances while making motion plans, we can evaluate trajectory sequences based on how susceptible they are to disturbances. We have demonstrated our approach on a simulation of a plane flying in two dimensions through a forest of polygonal obstacles. Future work will focus on extending our results to scenarios in which a stochastic description of uncertainty is more appropriate and using the tools from this paper to evaluate different trajectory libraries in order to determine whether they can be successfully employed to perform the task at hand.

Acknowledgements. This work was partially supported by ONR MURI grant N00014-09-1-1051 and the MIT Intelligence Initiative.

References

1. Betts, J.T.: Practical Methods for Optimal Control Using Nonlinear Programming. SIAM Advances in Design and Control. Society for Industrial and Applied Mathematics (2001)
2. Burridge, R.R., Rizzi, A.A., Koditschek, D.E.: Sequential composition of dynamically dexterous robot behaviors. International Journal of Robotics Research 18(6), 534–555 (1999)
3. Camacho, E.F., Bordons, C.: Model Predictive Control, 2nd edn. Springer (2004)
4. Dey, D., Liu, T.Y., Sofman, B., Bagnell, D.: Efficient optimization of control libraries. Technical report, Technical Report, CMU-RI-TR-11-20 (2011)
5. Frazzoli, E., Dahleh, M., Feron, E.: Maneuver-based motion planning for nonlinear systems with symmetries. IEEE Transactions on Robotics 21(6), 1077–1091 (2005)
6. Green, C., Kelly, A.: Toward optimal sampling in the space of paths. In: 13th International Symposium of Robotics Research (November 2007)
7. Green, M., Limebeer, D.: Linear Robust Control. Prentice Hall (1995)
8. Hu, T.C., Kahng, A.B., Robins, G.: Optimal robust path planning in general environments. IEEE Transactions on Robotics and Automation 9(6), 775–784 (1993)
9. Jacobs, P., Canny, J.: Robust motion planning for mobile robots. In: Proceedings of the 1990 IEEE International Conference on Robotics and Automation, pp. 2–7. IEEE (1990)
10. Karaman, Frazzoli: High-speed flight through an ergodic forest. In: IEEE Conference on Robotics and Automation (submitted 2012)
11. Karaman, S., Frazzoli, E.: Sampling-based algorithms for optimal motion planning. Int. Journal of Robotics Research 30, 846–894 (2011)
12. Kuffner, J., LaValle, S.: RRT-connect: An efficient approach to single-query path planning. In: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), pp. 995–1001 (2000)

13. Liu, C., Atkeson, C.G.: Standing balance control using a trajectory library. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2009), pp. 3031–3036. IEEE (2009)
14. Majumdar, A., Tobenkin, M., Tedrake, R.: Algebraic verification for parameterized motion planning libraries. In: Proceedings of the 2012 American Control Conference (ACC) (2012)
15. Mayne, D.Q., Seron, M.M., Rakovic, S.V.: Robust model predictive control of constrained linear systems with bounded disturbances. *Automatica* 41(2), 219–224 (2005)
16. Mellinger, D., Kumar, V.: Minimum snap trajectory generation and control for quadrotors. In: 2011 IEEE International Conference on Robotics and Automation (ICRA), pp. 2520–2525. IEEE (2011)
17. Nesterov, Y., Nemirovskii, A.: Interior-Point Polynomial Algorithms in Convex Programming. Studies in Applied Mathematics. Society for Industrial Mathematics (1995)
18. Parrilo, P.A.: Structured Semidefinite Programs and Semialgebraic Geometry Methods in Robustness and Optimization. PhD thesis, California Institute of Technology (May 18, 2000)
19. Petersen, I.R., Ugrinovskii, V.A., Savkin, A.V.: Robust control design using H-8 methods. Communications and control engineering series. Springer (2000)
20. Schouwenaars, T., Mettler, B., Feron, E., How, J.P.: Robust motion planning using a maneuver automation with built-in uncertainties. In: Proceedings of the 2003 American Control Conference, vol. 3, pp. 2211–2216. IEEE (2003)
21. Sermanet, P., Scoffier, M., Crudele, C., Muller, U., LeCun, Y.: Learning maneuver dictionaries for ground robot planning. In: Proc. 39th International Symposium on Robotics (ISR 2008) (2008)
22. Shkolnik, A.: Sample-Based Motion Planning in High-Dimensional and Differentially-Constrained Systems. PhD thesis, MIT (February 2010)
23. Steinhardt, J., Tedrake, R.: Finite-time regional verification of stochastic nonlinear systems. In: Proceedings of Robotics: Science and Systems (RSS 2011), January 17 (2011)
24. Tedrake, R., Manchester, I.R., Tobenkin, M.M., Roberts, J.W.: LQR-Trees: Feedback motion planning via sums of squares verification. *International Journal of Robotics Research* 29, 1038–1052 (2010)
25. Tobenkin, M.M., Manchester, I.R., Tedrake, R.: Invariant funnels around trajectories using sum-of-squares programming. In: Proceedings of the 18th IFAC World Congress (2011), Extended Version Available Online: arXiv:1010.3013 [math.DS]
26. Topcu, U., Packard, A.K., Seiler, P., Balas, G.J.: Robust region-of-attraction estimation. *IEEE Transactions on Automatic Control* 55(1), 137–142 (2010)
27. Toussaint, G.J.: Robust control and motion planning for nonlinear underactuated systems using H-infinity Techniques. PhD thesis, University of Illinois at Urbana-Champaign, Urbana, IL (2000)

Towards Consistent Vision-Aided Inertial Navigation

Joel A. Hesch, Dimitrios G. Kottas, Sean L. Bowman, and Stergios I. Roumeliotis

Abstract. In this paper, we study estimator inconsistency in Vision-aided Inertial Navigation Systems (VINS) from a standpoint of system observability. We postulate that a leading cause of inconsistency is the gain of spurious information along unobservable directions, resulting in smaller uncertainties, larger estimation errors, and possibly even divergence. We develop an Observability-Constrained VINS (OC-VINS), which explicitly enforces the unobservable directions of the system, hence preventing spurious information gain and reducing inconsistency. Our analysis, along with the proposed method for reducing inconsistency, are extensively validated with simulation trials and real-world experiments.

1 Introduction

A Vision-aided Inertial Navigation System (VINS) fuses data from a camera and an Inertial Measurement Unit (IMU) to track the six-degrees-of-freedom (d.o.f.) position and orientation (pose) of a sensing platform. This sensor pair is ideal since it combines complementary sensing capabilities [5]. For example, an IMU can accurately track dynamic motions over short time durations, while visual data can be used to estimate the pose displacement (up to scale) between two time-separated views. Within the robotics community, VINS has gained popularity as a method to address GPS-denied navigation for several reasons. First, contrary to approaches which utilize wheel odometry, VINS uses inertial sensing that can track general 3D motions of a vehicle. Hence, it is applicable to a variety of platforms such as aerial vehicles, legged robots, and even humans, which are not constrained to move along planar trajectories. Second, unlike laser-scanner-based methods that rely on the existence of structural planes [10] or height invariance in semi-structured environments [30], using vision as an exteroceptive sensor enables VINS methods to

Joel A. Hesch · Dimitrios G. Kottas · Sean L. Bowman · Stergios I. Roumeliotis
Dept. of Comp. Sci. and Eng., University of Minnesota, Minneapolis, MN 55455, USA
e-mail: {joel,dkottas,bowman,stergiros}@cs.umn.edu

work in unstructured areas such as collapsed buildings or outdoors. Furthermore, both cameras and IMUs are light-weight and have low power-consumption requirements, which has lead to recent advances in onboard estimation for Micro Aerial Vehicles (MAVs) (e.g., [36, 37]).

Numerous VINS approaches have been presented in the literature, including methods based on the Extended Kalman Filter (EKF) [3, 17, 26], the Unscented Kalman Filter (UKF) [7], and Batch-least Squares (BLS) [32]. Non-parametric estimators, such as the Particle Filter (PF), have also been applied to visual odometry (e.g., [6, 33]). However, these have focused on the simplified problem of estimating the 2D robot pose since the number of particles required is exponential in the size of the state vector. Existing work has addressed a variety of issues in VINS, such as reducing its computational cost [26, 37], dealing with delayed measurements [36], increasing the accuracy of feature initialization and estimation [15], and improving the robustness to estimator initialization errors [21].

A fundamental issue that has not yet been fully addressed in the literature is how estimator inconsistency affects VINS. As defined in [1], a state estimator is consistent if the estimation errors are zero-mean and have covariance smaller than or equal to the one calculated by the filter. We analyze the structure of the true and estimated systems, and postulate that a main source of inconsistency is spurious information gained along unobservable directions of the system. Furthermore, we propose a simple, yet powerful, estimator modification that explicitly prohibits this incorrect information gain. We validate our method with Monte-Carlo simulations to show that it has increased consistency and lower errors compared to standard VINS. In addition, we demonstrate the performance of our approach experimentally to show its viability for improving VINS consistency.

The rest of this paper is organized as follows: We begin with an overview of the related work (Sect. 2). In Sect. 3, we describe the system and measurement models, followed by our analysis of VINS inconsistency in Sect. 4. The proposed estimator modification is presented in Sect. 4.1, and subsequently validated both in simulations and experimentally (Sects. 5 and 6). Finally, we provide our concluding remarks and outline our future research directions in Sect. 7.

2 Related Work

Until recently, little attention was paid within the robotics community to the effects that observability properties can have on nonlinear estimator consistency. The work by Huang et al. [11, 12, 13] was the first to identify this connection for several 2D localization problems (i.e., simultaneous localization and mapping, cooperative localization). The authors showed that, for these problems, a mismatch exists between the number of unobservable directions of the true nonlinear system and the linearized system used for estimation purposes. In particular, the estimated (linearized) system has one-fewer unobservable direction than the true system, allowing the estimator to surreptitiously gain spurious information along the direction corresponding to

global orientation. This increases the estimation errors while reducing the estimator uncertainty, and leads to inconsistency.

Several authors have studied the observability properties of VINS under a variety of scenarios. For the task of IMU-camera extrinsic calibration, Mirzaei and Roumeliotis [24], as well as Kelly and Sukhatme [16], have analyzed the system observability using Lie derivatives [8] to determine when the IMU-camera transformation is observable. Jones and Soatto [15] studied VINS observability by examining the indistinguishable trajectories of the system [14] under different sensor configurations (i.e., inertial only, vision only, vision and inertial). Finally, Martinelli [22] utilized the concept of continuous symmetries to show that the IMU biases, 3D velocity, and absolute roll and pitch angles, are observable for VINS.

VINS inconsistency was recently addressed by Li and Mourikis [18]. Specifically, they studied the link between the VINS observability properties and estimator inconsistency for the bias-free case, and leveraged the First-Estimates Jacobian (FEJ) methodology of [11] to mitigate inconsistency in Visual-Inertial Odometry (VIO). In contrast to their work, our approach has the advantage that any linearization method can be employed (e.g., computing Jacobians analytically, numerically, or using sample points) by the estimator. Additionally, we show that our approach is flexible enough to be applied in a variety of VINS problems such as VIO or Simultaneous Localization and Mapping (SLAM).

Specifically, we leverage the key result of the existing VINS observability analysis, i.e., that the VINS model has four unobservable degrees of freedom, corresponding to three-d.o.f. global translations and one-d.o.f. global rotation about the gravity vector. Due to linearization errors, the number of unobservable directions is reduced in a standard EKF-based VINS approach, allowing the estimator to gain spurious information and leading to inconsistency. What we present is a significant, nontrivial extension of our previous work on mitigating inconsistency in 2D robot localization problems [12]. This is due in part to the higher-dimensional state of the 3D VINS system as compared to 2D localization (15 elements vs. 3), as well as more complex motion and measurement models. Furthermore, the proposed solution for reducing estimator inconsistency is general, and can be directly applied in a variety of linearized estimation frameworks such as the EKF and UKF.

3 VINS Estimator Description

We begin with an overview of the propagation and measurement models which govern the VINS system. We adopt the EKF as our framework for fusing the camera and IMU measurements to estimate the state of the system including the pose, velocity, and IMU biases, as well as the 3D positions of visual landmarks observed by the camera. We operate in a previously unknown environment and utilize two types of visual features in our VINS framework. The first are opportunistic features (OFs) that can be accurately and efficiently tracked across short image sequences (e.g., using KLT [20]), but are not visually distinctive. OFs are efficiently used to estimate the motion of the camera, but they are not included in the state vector.

The second are Distinguishable Features (DFs), which are typically much fewer in number, and can be reliably redetected when revisiting an area (e.g., SIFT keys [19]). The 3D coordinates of the DFs are estimated to construct a map of the area.

3.1 System State and Propagation Model

The EKF estimates the IMU pose and linear velocity together with the time-varying IMU biases and a map of visual features. The filter state is the $(16 + 3N) \times 1$ vector:

$$\mathbf{x} = [{}^I\bar{q}_G^T \ \mathbf{b}_g^T \ {}^G\mathbf{v}_I^T \ \mathbf{b}_a^T \ {}^G\mathbf{p}_I^T \ | \ {}^G\mathbf{f}_1^T \dots {}^G\mathbf{f}_N^T]^T = [\mathbf{x}_s^T \ | \ \mathbf{x}_m^T]^T, \quad (1)$$

where $\mathbf{x}_s(t)$ is the 16×1 sensor platform state, and $\mathbf{x}_m(t)$ is the $3N \times 1$ state of the map. The first component of the sensor platform state is ${}^I\bar{q}_G(t)$ which is the unit quaternion representing the orientation of the *global frame* $\{G\}$ in the IMU frame, $\{I\}$, at time t . The frame $\{I\}$ is attached to the IMU, while $\{G\}$ is a local-vertical reference frame whose origin coincides with the initial IMU position. The sensor platform state also includes the position and velocity of $\{I\}$ in $\{G\}$, denoted by the 3×1 vectors ${}^G\mathbf{p}_I(t)$ and ${}^G\mathbf{v}_I(t)$, respectively. The remaining components are the biases, $\mathbf{b}_g(t)$ and $\mathbf{b}_a(t)$, affecting the gyroscope and accelerometer measurements, which are modeled as random-walk processes driven by the zero-mean, white Gaussian noise $\mathbf{n}_{wg}(t)$ and $\mathbf{n}_{wa}(t)$, respectively.

The map, \mathbf{x}_m , comprises N DFs, ${}^G\mathbf{f}_i$, $i = 1, \dots, N$, and grows as new DFs are observed [9]. However, we do not store OFs in the map. Instead, all OFs are processed and marginalized on-the-fly using the MSC-KF approach [25] (see Sect. 3.2). With the state of the system now defined, we turn our attention to the continuous-time kinematic model which governs the time evolution of the system state.

3.1.1 Continuous-Time Model

The system model describing the time evolution of the state is (see [4, 34]):

$${}^I\dot{\bar{q}}_G(t) = \frac{1}{2}\Omega(\omega(t)){}^I\bar{q}_G(t) \ , \ {}^G\dot{\mathbf{p}}_I(t) = {}^G\mathbf{v}_I(t) \ , \ {}^G\dot{\mathbf{v}}_I(t) = {}^G\mathbf{a}_I(t) \quad (2)$$

$$\dot{\mathbf{b}}_g(t) = \mathbf{n}_{wg}(t) \ , \ \dot{\mathbf{b}}_a(t) = \mathbf{n}_{wa}(t) \ , \ {}^G\dot{\mathbf{f}}_i(t) = \mathbf{0}_{3 \times 1}, \ i = 1, \dots, N. \quad (3)$$

In these expressions, $\omega(t) = [\omega_1(t) \ \omega_2(t) \ \omega_3(t)]^T$ is the rotational velocity of the IMU, expressed in $\{I\}$, ${}^G\mathbf{a}_I(t)$ is the body acceleration expressed in $\{G\}$, and

$$\Omega(\omega) = \begin{bmatrix} -[\omega \times] & \omega \\ -\omega^T & 0 \end{bmatrix}, \quad [\omega \times] \triangleq \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix}.$$

The gyroscope and accelerometer measurements, ω_m and \mathbf{a}_m , are modeled as

$$\omega_m(t) = \omega(t) + \mathbf{b}_g(t) + \mathbf{n}_g(t) \quad (4)$$

$$\mathbf{a}_m(t) = \mathbf{C}({}^I\bar{q}_G(t))({}^G\mathbf{a}_I(t) - {}^G\mathbf{g}) + \mathbf{b}_a(t) + \mathbf{n}_a(t), \quad (5)$$

where \mathbf{n}_g and \mathbf{n}_a are zero-mean, white Gaussian noise processes, and ${}^G\mathbf{g}$ is the gravitational acceleration. The matrix $\mathbf{C}(\bar{q})$ is the rotation matrix corresponding to \bar{q} . The DFs belong to the static scene, thus, their time derivatives are zero [see (3)].

Linearizing at the current estimates and applying the expectation operator on both sides of (2)-(3), we obtain the state estimate propagation model

$${}^I\dot{\hat{q}}_G(t) = \frac{1}{2}\Omega(\hat{\omega}(t))'{}^I\hat{q}_G(t) , \quad {}^G\dot{\hat{\mathbf{p}}}_I(t) = {}^G\hat{\mathbf{v}}_I(t) , \quad {}^G\dot{\hat{\mathbf{v}}}_I(t) = \mathbf{C}^T({}^I\hat{q}_G(t))\hat{\mathbf{a}}(t) + {}^G\mathbf{g} \quad (6)$$

$$\hat{\mathbf{b}}_g(t) = \mathbf{0}_{3 \times 1} , \quad \hat{\mathbf{b}}_a(t) = \mathbf{0}_{3 \times 1} , \quad {}^G\dot{\hat{\mathbf{f}}}_I(t) = \mathbf{0}_{3 \times 1} , \quad i = 1, \dots, N, \quad (7)$$

where $\hat{\mathbf{a}}(t) = \mathbf{a}_m(t) - \hat{\mathbf{b}}_a(t)$, and $\hat{\omega}(t) = \omega_m(t) - \hat{\mathbf{b}}_g(t)$. The $(15 + 3N) \times 1$ error-state vector is defined as

$$\tilde{\mathbf{x}} = \left[{}^I\delta\theta_G^T \quad \tilde{\mathbf{b}}_g^T \quad {}^G\tilde{\mathbf{v}}_I^T \quad \tilde{\mathbf{b}}_a^T \quad {}^G\tilde{\mathbf{p}}_I^T \mid {}^G\tilde{\mathbf{f}}_1^T \dots {}^G\tilde{\mathbf{f}}_N^T \right]^T = \left[\tilde{\mathbf{x}}_s^T \mid \tilde{\mathbf{x}}_m^T \right]^T , \quad (8)$$

where $\tilde{\mathbf{x}}_s(t)$ is the 15×1 error state corresponding to the sensing platform, and $\tilde{\mathbf{x}}_m(t)$ is the $3N \times 1$ error state of the map. For the IMU position, velocity, biases, and the map, an additive error model is utilized (i.e., $\tilde{x} = x - \hat{x}$ is the error in the estimate \hat{x} of a quantity x). However, for the quaternion we employ a multiplicative error model [34]. Specifically, the error between the quaternion \bar{q} and its estimate \hat{q} is the 3×1 angle-error vector, $\delta\theta$, implicitly defined by the error quaternion

$$\delta\bar{q} = \bar{q} \otimes \hat{q}^{-1} \simeq \left[\frac{1}{2}\delta\theta^T \quad 1 \right]^T , \quad (9)$$

where $\delta\bar{q}$ describes the small rotation that causes the true and estimated attitude to coincide. This allows us to represent the attitude uncertainty by the 3×3 covariance matrix $\mathbb{E}[\delta\theta\delta\theta^T]$, which is a minimal representation.

The linearized continuous-time error-state equation is

$$\dot{\tilde{\mathbf{x}}} = \begin{bmatrix} \mathbf{F}_{s,c} & \mathbf{0}_{15 \times 3N} \\ \mathbf{0}_{3N \times 15} & \mathbf{0}_{3N} \end{bmatrix} \tilde{\mathbf{x}} + \begin{bmatrix} \mathbf{G}_{s,c} \\ \mathbf{0}_{3N \times 12} \end{bmatrix} \mathbf{n} = \mathbf{F}_c \tilde{\mathbf{x}} + \mathbf{G}_c \mathbf{n} , \quad (10)$$

where $\mathbf{0}_{3N}$ denotes the $3N \times 3N$ matrix of zeros, $\mathbf{n} = [\mathbf{n}_g^T \quad \mathbf{n}_{wg}^T \quad \mathbf{n}_a^T \quad \mathbf{n}_{wa}^T]^T$ is the system noise, $\mathbf{F}_{s,c}$ is the continuous-time error-state transition matrix corresponding to the sensor platform state, and $\mathbf{G}_{s,c}$ is the continuous time input noise matrix, i.e.,

$$\mathbf{F}_{s,c} = \begin{bmatrix} -[\hat{\omega} \times] & -\mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ -\mathbf{C}^T({}^I\hat{q}_G) [\hat{\mathbf{a}} \times] & \mathbf{0}_3 & \mathbf{0}_3 & -\mathbf{C}^T({}^I\hat{q}_G) \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{I}_3 & \mathbf{0}_3 \end{bmatrix} , \quad \mathbf{G}_{s,c} = \begin{bmatrix} -\mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & -\mathbf{C}^T({}^I\hat{q}_G) \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{I}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 \end{bmatrix} \quad (11)$$

where $\mathbf{0}_3$ is the 3×3 matrix of zeros. The system noise is modelled as a zero-mean white Gaussian process with autocorrelation $\mathbb{E}[\mathbf{n}(t)\mathbf{n}^T(\tau)] = \mathbf{Q}_c\delta(t-\tau)$ which depends on the IMU noise characteristics and is computed off-line [34].

3.1.2 Discrete-Time Implementation

The IMU signals ω_m and \mathbf{a}_m are sampled at a constant rate $1/\delta t$, where $\delta t \triangleq t_{k+1} - t_k$. Every time a new IMU measurement is received, the state estimate is propagated using 4th-order Runge-Kutta numerical integration of (6)–(7). In order to derive the covariance propagation equation, we evaluate the discrete-time state transition matrix, Φ_k , and the discrete-time system noise covariance matrix, $\mathbf{Q}_{d,k}$, as

$$\Phi_k = \Phi(t_{k+1}, t_k) = \exp\left(\int_{t_k}^{t_{k+1}} \mathbf{F}_c(\tau) d\tau\right), \quad \mathbf{Q}_{d,k} = \int_{t_k}^{t_{k+1}} \Phi(t_{k+1}, \tau) \mathbf{G}_c \mathbf{Q}_c \mathbf{G}_c^T \Phi^T(t_{k+1}, \tau) d\tau.$$

The propagated covariance is then computed as $\mathbf{P}_{k+1|k} = \Phi_k \mathbf{P}_{k|k} \Phi_k^T + \mathbf{Q}_{d,k}$.

3.2 Measurement Update Model

As the camera-IMU platform moves, the camera observes both opportunistic and distinguishable visual features. These measurements are exploited to concurrently estimate the motion of the sensing platform and the map of DFs. We distinguish three types of filter updates: (i) DF updates of features already in the map, (ii) initialization of DFs not yet in the map, and (iii) OF updates. We first describe the feature measurement model, and subsequently detail how it is employed in each case.

To simplify the discussion, we consider the observation of a single point \mathbf{f}_i . The camera measures, \mathbf{z}_i , which is the perspective projection of the 3D point, $'\mathbf{f}_i$, expressed in the current IMU frame $\{I\}$, onto the image plane¹, i.e.,

$$\mathbf{z}_i = \frac{1}{z} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \eta_i, \quad \text{where} \quad [x \ y \ z]^T = {}' \mathbf{f}_i = \mathbf{C}({}' \bar{\mathbf{q}}_G) ({}^G \mathbf{f}_i - {}^G \mathbf{p}_I). \quad (12)$$

The measurement noise, η_i , is modeled as zero mean, white Gaussian with covariance \mathbf{R}_i . The linearized error model is $\tilde{\mathbf{z}}_i = \mathbf{z}_i - \hat{\mathbf{z}}_i \simeq \mathbf{H}_i \tilde{\mathbf{x}} + \eta_i$, where $\hat{\mathbf{z}}$ is the expected measurement computed by evaluating (12) at the current state estimate, and the measurement Jacobian, \mathbf{H}_i , is

$$\mathbf{H}_i = \mathbf{H}_{cam} [\mathbf{H}_{\theta_G} \ \mathbf{0}_{3 \times 9} \ \mathbf{H}_{\mathbf{p}_I} \mid \mathbf{0}_3 \cdots \mathbf{H}_{\mathbf{f}_i} \cdots \mathbf{0}_3] \quad (13)$$

$$\mathbf{H}_{cam} = \frac{1}{z^2} \begin{bmatrix} z & 0 & -x \\ 0 & z & -y \end{bmatrix}, \quad \mathbf{H}_{\theta_G} = [\mathbf{C}({}' \bar{\mathbf{q}}_G) ({}^G \mathbf{f}_i - {}^G \mathbf{p}_I) \times], \quad \mathbf{H}_{\mathbf{p}_I} = -\mathbf{C}({}' \bar{\mathbf{q}}_G), \quad \mathbf{H}_{\mathbf{f}_i} = \mathbf{C}({}' \bar{\mathbf{q}}_G)$$

Here, \mathbf{H}_{cam} , is the Jacobian of the perspective projection with respect to $'\mathbf{f}_i$, while \mathbf{H}_{θ_G} , $\mathbf{H}_{\mathbf{p}_I}$, and $\mathbf{H}_{\mathbf{f}_i}$, are the Jacobians of $'\mathbf{f}_i$ with respect to $'\bar{\mathbf{q}}_G$, ${}^G \mathbf{p}_I$, and ${}^G \mathbf{f}_i$, respectively.

This measurement model is utilized in each of the three update methods. For DFs that are already in the map, we directly apply the measurement model (12)–(13) to

¹ Without loss of generality, we express the image measurement in normalized pixel coordinates, and consider the camera frame to be coincident with the IMU. In practice, we perform both intrinsic and extrinsic camera/IMU calibration off-line [2, 24].

update the filter. We compute the Kalman gain, $\mathbf{K} = \mathbf{P}_{k+1|k} \mathbf{H}_i^T (\mathbf{H}_i \mathbf{P}_{k+1|k} \mathbf{H}_i^T + \mathbf{R}_i)^{-1}$, and the measurement residual $\mathbf{r}_i = \mathbf{z}_i - \hat{\mathbf{z}}_i$. Employing these quantities, we compute the EKF state and covariance update as

$$\hat{\mathbf{x}}_{k+1|k+1} = \hat{\mathbf{x}}_{k+1|k} + \mathbf{K} \mathbf{r}_i , \quad \mathbf{P}_{k+1|k+1} = \mathbf{P}_{k+1|k} - \mathbf{P}_{k+1|k} \mathbf{H}_i^T (\mathbf{H}_i \mathbf{P}_{k+1|k} \mathbf{H}_i^T + \mathbf{R}_i)^{-1} \mathbf{H}_i \mathbf{P}_{k+1|k} \quad (14)$$

For previously unobserved DFs, we compute an initial estimate, along with covariance and cross-correlations by solving a bundle-adjustment over a short time window [35]. Finally, for OFs, we employ the MSC-KF approach [25] to impose a pose update constraining all the views from which the feature was seen. To accomplish this, we utilize stochastic cloning [29] over a window of m camera poses.

4 Observability-Constrained VINS

Using the VINS system model presented above, we hereafter describe how the system observability properties influence estimator consistency. When using a linearized estimator, such as the EKF, errors in linearization while evaluating the system and measurement Jacobians change the directions in which information is acquired by the estimator. If this information lies along unobservable directions, it leads to larger errors, smaller uncertainties, and inconsistency. We first analyze this issue, and subsequently, present an Observability-Constrained VINS (OC-VINS) that explicitly adheres to the observability properties of VINS.

The Observability Gramian [23] is defined as a function of the linearized measurement model, \mathbf{H} , and the discrete-time state transition matrix, Φ , which are in turn functions of the linearization point, \mathbf{x} , i.e.,

$$\mathbf{M}(\mathbf{x}) = \begin{bmatrix} \mathbf{H}_1 \\ \mathbf{H}_2 \Phi_{2,1} \\ \vdots \\ \mathbf{H}_k \Phi_{k,1} \end{bmatrix} \quad (15)$$

where $\Phi_{k,1} = \Phi_{k-1} \cdots \Phi_1$ is the state transition matrix from time step 1 to k, with $\Phi_1 = \mathbf{I}_{15+3 \times N}$. To simplify the discussion, we consider a single landmark in the state vector, and write the first block row as

$$\mathbf{H}_1 = \mathbf{H}_{cam,1} \mathbf{C}({}^l\bar{q}_{G,1}) \left[[{}^G\mathbf{f} - {}^G\mathbf{p}_{l,1} \times] \mathbf{C}({}^l\bar{q}_{G,1})^T \mathbf{0}_3 \mathbf{0}_3 \mathbf{0}_3 -\mathbf{I}_3 \mathbf{I}_3 \right],$$

where ${}^l\bar{q}_{G,1}$, denotes the rotation of $\{G\}$ with respect to frame $\{l\}$ at time step 1, and for the purposes of the observability analysis, all the quantities appearing in the previous expression are the true ones. As shown in [9], the k -th block row, for $k > 1$, is of the form:

$$\mathbf{H}_k \Phi_{k,1} = \mathbf{H}_{cam,k} \mathbf{C}({}^l\bar{q}_{G,k}) \left[[{}^G\mathbf{f} - {}^G\mathbf{p}_{l,1} - {}^G\mathbf{v}_{l,1} \delta_{k-1} + \frac{1}{2} {}^G\mathbf{g} \delta_{k-1}^2] \times] \mathbf{C}({}^l\bar{q}_{G,1})^T \mathbf{D}_k -\mathbf{I} \delta_{k-1} \mathbf{E}_k -\mathbf{I}_3 \mathbf{I}_3 \right],$$

where $\delta_{k-1} = (k-1)\delta t$, and \mathbf{D}_k and \mathbf{E}_k are both time-varying matrices. It is straightforward to verify that the right nullspace of $\mathbf{M}(\mathbf{x})$ spans four directions, i.e.,

$$\mathbf{M}(\mathbf{x})\mathbf{N}_1 = \mathbf{0}, \quad \mathbf{N}_1 = \begin{bmatrix} \mathbf{0}_3 & \mathbf{C}({}^l\bar{q}_{G,1}) {}^G\mathbf{g} \\ \mathbf{0}_3 & \mathbf{0}_{3 \times 1} \\ \mathbf{0}_3 & -[{}^G\mathbf{v}_{l,1} \times] {}^G\mathbf{g} \\ \mathbf{0}_3 & \mathbf{0}_{3 \times 1} \\ \mathbf{I}_3 & -[{}^G\mathbf{p}_{l,1} \times] {}^G\mathbf{g} \\ \mathbf{I}_3 & -[{}^G\mathbf{f} \times] {}^G\mathbf{g} \end{bmatrix} = [\mathbf{N}_{t,1} \mid \mathbf{N}_{r,1}] \quad (16)$$

where $\mathbf{N}_{t,1}$ corresponds to global translations and $\mathbf{N}_{r,1}$ corresponds to global rotations about the gravity vector.

Ideally, any estimator we employ should correspond to a system with an unobservable subspace that matches these directions, both in number and structure. However, when linearizing about the estimated state $\hat{\mathbf{x}}$, $\mathbf{M}(\hat{\mathbf{x}})$ gains rank due to errors in the state estimates across time [9]. To address this problem and ensure that (16) is satisfied for every block row of \mathbf{M} when the state estimates are used for computing \mathbf{H}_ℓ , and $\Phi_{\ell,1}$, $\ell = 1, \dots, k$, we must ensure that $\mathbf{H}_\ell \Phi_{\ell,1} \mathbf{N}_1 = \mathbf{0}$, $\ell = 1, \dots, k$.

One way to enforce this is by requiring that at each time step

$$\mathbf{N}_{\ell+1} = \Phi_\ell \mathbf{N}_\ell, \quad \mathbf{H}_\ell \mathbf{N}_\ell = \mathbf{0}, \quad \ell = 1, \dots, k \quad (17)$$

where \mathbf{N}_ℓ , $\ell \geq 1$ is computed analytically (see (18) and [9]). This can be accomplished by appropriately modifying Φ_ℓ and \mathbf{H}_ℓ following the process described in the next section.

4.1 OC-VINS: Algorithm Description

Hereafter, we present our OC-VINS algorithm which enforces the observability constraints dictated by the VINS system structure. Rather than changing the linearization points explicitly (e.g., as in [11]), we maintain the nullspace, \mathbf{N}_k , at each time step, and use it to enforce the unobservable directions. The 15×4 nullspace block, \mathbf{N}_k^R , corresponding to the robot state is analytically defined as [9]:

$$\mathbf{N}_1^R = \begin{bmatrix} \mathbf{0}_3 & \mathbf{C}({}^l\hat{q}_{G,1|1}) {}^G\mathbf{g} \\ \mathbf{0}_3 & \mathbf{0}_{3 \times 1} \\ \mathbf{0}_3 & -[{}^G\hat{\mathbf{v}}_{l,1|1} \times] {}^G\mathbf{g} \\ \mathbf{0}_3 & \mathbf{0}_{3 \times 1} \\ \mathbf{I}_3 & -[{}^G\hat{\mathbf{p}}_{l,1|1} \times] {}^G\mathbf{g} \end{bmatrix}, \quad \mathbf{N}_k^R = \begin{bmatrix} \mathbf{0}_3 & \mathbf{C}({}^l\hat{q}_{G,k|k-1}) {}^G\mathbf{g} \\ \mathbf{0}_3 & \mathbf{0}_{3 \times 1} \\ \mathbf{0}_3 & -[{}^G\hat{\mathbf{v}}_{l,k|k-1} \times] {}^G\mathbf{g} \\ \mathbf{0}_3 & \mathbf{0}_{3 \times 1} \\ \mathbf{I}_3 & -[{}^G\hat{\mathbf{p}}_{l,k|k-1} \times] {}^G\mathbf{g} \end{bmatrix} = [\mathbf{N}_{t,k}^R \mid \mathbf{N}_{r,k}^R]. \quad (18)$$

The 3×4 nullspace block, \mathbf{N}_ℓ^f , corresponding to the feature state, is a function of the feature estimate at time t_ℓ when it was initialized, i.e.,

$$\mathbf{N}_k^f = [\mathbf{I}_3 - [{}^G\hat{\mathbf{f}}_{\ell|\ell} \times] {}^G\mathbf{g}] \quad (19)$$

4.1.1 Modification of the State Transition Matrix Φ

During the propagation step, we must ensure that $\mathbf{N}_{k+1}^R = \Phi_k^R \mathbf{N}_k^R$, where Φ_k^R is the first 15×15 sub-block of Φ_k corresponding to the robot state. We note that the

constraint on $\mathbf{N}_{r,k}^R$ is automatically satisfied by the structure of Φ_k^R [see (20) and [9]], so we focus on $\mathbf{N}_{r,k}^R$. We rewrite this equation element-wise as

$$\mathbf{N}_{r,k+1}^R = \Phi_k^R \mathbf{N}_{r,k}^R \rightarrow \begin{bmatrix} \mathbf{C} \left({}^I \hat{\mathbf{q}}_{G,k+1|k} \right)^G \mathbf{g} \\ \mathbf{0}_{3 \times 1} \\ -[{}^G \hat{\mathbf{v}}_{I,k+1|k} \times] {}^G \mathbf{g} \\ \mathbf{0}_{3 \times 1} \\ -[{}^G \hat{\mathbf{p}}_{I,k+1|k} \times] {}^G \mathbf{g} \end{bmatrix} = \begin{bmatrix} \Phi_{11} & \Phi_{12} & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ \Phi_{31} & \Phi_{32} & \mathbf{I}_3 & \Phi_{34} & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{I}_3 & \mathbf{0}_3 \\ \Phi_{51} & \Phi_{52} & \delta t \mathbf{I}_3 & \Phi_{54} & \mathbf{I}_3 \end{bmatrix} \begin{bmatrix} \mathbf{C} \left({}^I \hat{\mathbf{q}}_{G,k|k-1} \right)^G \mathbf{g} \\ \mathbf{0}_{3 \times 1} \\ -[{}^G \hat{\mathbf{v}}_{I,k|k-1} \times] {}^G \mathbf{g} \\ \mathbf{0}_{3 \times 1} \\ -[{}^G \hat{\mathbf{p}}_{I,k|k-1} \times] {}^G \mathbf{g} \end{bmatrix}. \quad (20)$$

From the first block row we have that

$$\mathbf{C} \left({}^I \hat{\mathbf{q}}_{G,k+1|k} \right)^G \mathbf{g} = \Phi_{11} \mathbf{C} \left({}^I \hat{\mathbf{q}}_{G,k|k-1} \right)^G \mathbf{g} \Rightarrow \Phi_{11} = \mathbf{C} \left({}^{I,k+1|k} \hat{\mathbf{q}}_{I,k|k-1} \right). \quad (21)$$

The requirements for the third and fifth block rows are

$$\Phi_{31} \mathbf{C} \left({}^I \hat{\mathbf{q}}_{G,k|k-1} \right)^G \mathbf{g} = [{}^G \hat{\mathbf{v}}_{I,k|k-1} \times] {}^G \mathbf{g} - [{}^G \hat{\mathbf{v}}_{I,k+1|k} \times] {}^G \mathbf{g} \quad (22)$$

$$\Phi_{51} \mathbf{C} \left({}^I \hat{\mathbf{q}}_{G,k|k-1} \right)^G \mathbf{g} = \delta t [{}^G \hat{\mathbf{v}}_{I,k|k-1} \times] {}^G \mathbf{g} + [{}^G \hat{\mathbf{p}}_{I,k|k-1} \times] {}^G \mathbf{g} - [{}^G \hat{\mathbf{p}}_{I,k+1|k} \times] {}^G \mathbf{g} \quad (23)$$

both of which are in the form $\mathbf{A}\mathbf{u} = \mathbf{w}$, where \mathbf{u} and \mathbf{w} are nullspace elements that are fixed, and we seek to find a perturbed \mathbf{A}^* , for $\mathbf{A} = \Phi_{31}$ and $\mathbf{A} = \Phi_{51}$ that fulfills the constraint. To compute the minimum perturbation, \mathbf{A}^* , of \mathbf{A} , we formulate the following minimization problem

$$\min_{\mathbf{A}^*} \|\mathbf{A}^* - \mathbf{A}\|_{\mathcal{F}}^2, \quad \text{s.t. } \mathbf{A}^* \mathbf{u} = \mathbf{w} \quad (24)$$

where $\|\cdot\|_{\mathcal{F}}$ denotes the Frobenius matrix norm. After employing the method of Lagrange multipliers, and solving the corresponding KKT optimality conditions, the optimal \mathbf{A}^* that fulfills (24) is $\mathbf{A}^* = \mathbf{A} - (\mathbf{A}\mathbf{u} - \mathbf{w})(\mathbf{u}^T \mathbf{u})^{-1} \mathbf{u}^T$.

We compute the modified Φ_{11} from (21), and Φ_{31} and Φ_{51} from (24) and construct the constrained discrete-time state transition matrix. We then proceed with covariance propagation (see Sect. 3.1).

4.1.2 Modification of H

During each update step, we seek to satisfy $\mathbf{H}_k \mathbf{N}_k = \mathbf{0}$. Based on (13), we can write this relationship *per feature* as

$$\mathbf{H}_{cam} \begin{bmatrix} \mathbf{H}_{\theta_G} & \mathbf{0}_{3 \times 9} & \mathbf{H}_{\mathbf{p}_I} & | & \mathbf{H}_{\mathbf{f}} \end{bmatrix} \begin{bmatrix} \mathbf{0}_3 & \mathbf{C} \left({}^I \hat{\mathbf{q}}_{G,k|k-1} \right)^G \mathbf{g} \\ \mathbf{0}_3 & \mathbf{0}_{3 \times 1} \\ \mathbf{0}_3 & -[{}^G \hat{\mathbf{v}}_{I,k|k-1} \times] {}^G \mathbf{g} \\ \mathbf{0}_3 & \mathbf{0}_{3 \times 1} \\ \mathbf{I}_3 & -[{}^G \hat{\mathbf{p}}_{I,k|k-1} \times] {}^G \mathbf{g} \\ \mathbf{I}_3 & -[{}^G \hat{\mathbf{f}}_{\ell|\ell} \times] {}^G \mathbf{g} \end{bmatrix} = \mathbf{0}. \quad (25)$$

The first block column of (25) requires that $\mathbf{H}_f = -\mathbf{H}_{p_I}$. Hence, we rewrite the second block column of (25) as

$$\mathbf{H}_{cam} [\mathbf{H}_{\theta_G} \ \mathbf{H}_{\mathbf{p}_I}] \begin{bmatrix} \mathbf{C} \left({}^I \hat{\mathbf{q}}_{G,k|k-1} \right) {}^G \mathbf{g} \\ \left([{}^G \hat{\mathbf{f}}_{\ell|\ell} \times] - [{}^G \hat{\mathbf{p}}_{I,k|k-1} \times] \right) {}^G \mathbf{g} \end{bmatrix} = \mathbf{0}. \quad (26)$$

This is a constraint of the form $\mathbf{A}\mathbf{u} = \mathbf{0}$, where \mathbf{u} is a fixed quantity determined by elements in the nullspace, and \mathbf{A} comprises elements of the measurement Jacobian. We compute the optimal \mathbf{A}^* that satisfies this relationship using the solution to (24). After computing the optimal \mathbf{A}^* , we recover the Jacobian as

$$\mathbf{H}_{cam} \mathbf{H}_{\theta_G} = \mathbf{A}'_{1:2,1:3}, \quad \mathbf{H}_{cam} \mathbf{H}_{\mathbf{p}_I} = \mathbf{A}'_{1:2,4:6}, \quad \mathbf{H}_{cam} \mathbf{H}_f = -\mathbf{A}'_{1:2,4:6} \quad (27)$$

where the subscripts (i:j, m:n) denote the submatrix spanning rows i to j, and columns m to n. After computing the modified measurement Jacobian, we proceed with the filter update as described in Sect. 3.2.

5 Simulations

We conducted Monte-Carlo simulations to evaluate the impact of the proposed Observability-Constrained VINS (OC-VINS) method on estimator consistency. We compared its performance to the standard VINS (Std-VINS), as well as the ideal VINS that linearizes about the true state². Specifically, we computed the Root Mean Squared Error (RMSE) and Normalized Estimation Error Squared (NEES) over 100 trials in which the camera-IMU platform traversed a circular trajectory of radius 5 m at an average velocity of 60 cm/s.³ The camera observed visual features distributed on the interior wall of a circumscribing cylinder with radius 6 m and height 2 m (see Fig. 1a). The effect of inconsistency during a single run is depicted in Fig. 1b. The error and corresponding 3σ bounds of uncertainty are plotted for the rotation about the gravity vector. It is clear that the Std-VINS gains spurious information, hence reducing its 3σ bounds of uncertainty, while the Ideal-VINS and the OC-VINS do not. The Std-VINS becomes inconsistent on this run as the orientation errors fall outside of the uncertainty bounds, while both the Ideal-VINS and the OC-VINS remain consistent. Figure 2 displays the RMSE and NEES, in which we observe that the OC-VINS obtains orientation accuracy and consistency levels similar to the ideal, while significantly outperforming Std-VINS. Similarly, the OC-VINS obtains better positioning accuracy compared to Std-VINS.

² Since the ideal VINS has access to the true state, it is not realizable in practice, but we include it here as a baseline comparison.

³ The camera had 45 deg field of view, with $\sigma_{px} = 1px$, while the IMU was modeled with MEMS quality sensors.

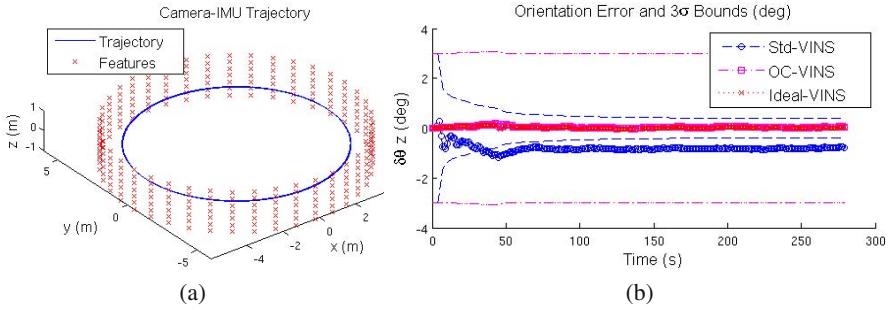


Fig. 1 (a) Camera-IMU trajectory and 3D features. (b) Error and 3σ bounds for the rotation about the gravity vector, plotted for a single run.

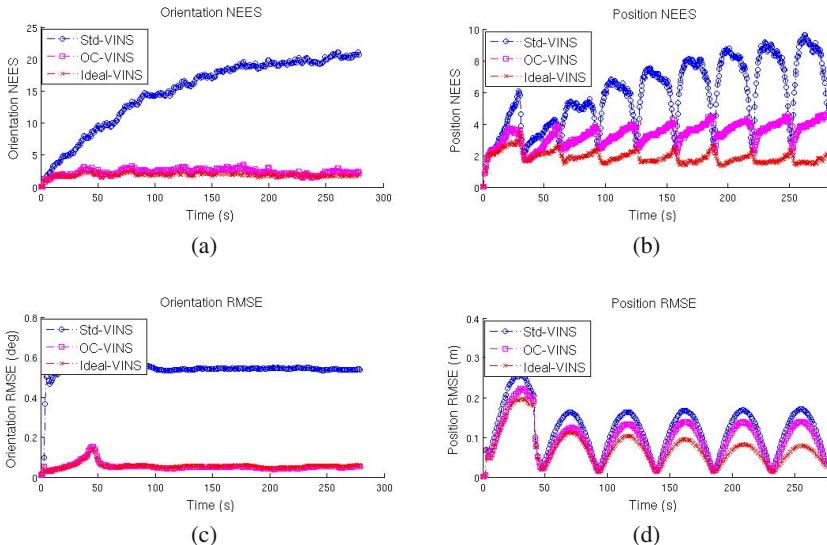


Fig. 2 The RMSE and NEES errors for position and orientation plotted for all three filters, averaged per time step over 100 Monte Carlo trials

6 Experimental Results

6.1 Implementation Remarks

The image processing is separated into two components: one for extracting and tracking short-term OFs, and one for extracting DFs to use in SLAM.

OFs are extracted from images using the Shi-Tomasi corner detector [31]. After acquiring image k , it is inserted into a sliding window buffer of m images, $\{k-m+1, k-m+2, \dots, k\}$. We then extract features from the first image in the

window and track them pairwise through the window using the KLT tracking algorithm [20]. To remove outliers from the resulting tracks, we use a two-point algorithm to find the essential matrix between successive frames. Given the filter's estimated rotation between image i and j , ${}^i\hat{q}_j$, we estimate the essential matrix from only two feature correspondences. This approach is more robust than the traditional five-point algorithm [27] because it provides two solutions for the essential matrix rather than up to ten, and as it requires only two data points, it reaches a consensus with fewer hypotheses when used in a RANSAC framework.

The DFs are extracted using SIFT descriptors [19]. To identify global features observed from several different images, we first utilize a vocabulary tree (VT) structure for image matching [28]. Specifically, for an image taken at time k , the VT is used to select which image(s) taken at times $1, 2, \dots, k-1$ correspond to the same physical scene. Among those images that the VT reports as matching, the SIFT descriptors from each are compared to those from image k to create tentative feature correspondences. The epipolar constraint is then enforced using RANSAC and Nister's five-point algorithm [27] to eliminate outliers. It is important to note that the images used to construct the VT (off-line) are not taken along our experimental trajectory, but rather are randomly selected from a set of representative images.

At every time step, the robot poses corresponding to the last m images are kept in the state vector, as described in [29]. After the processing of a new image is completed, all the OFs that first appeared at the oldest augmented robot pose, are processed following the MSC-KF approach, as discussed earlier. The frequency at which new DFs are initialized into the map is a scalable option which can be tuned according to the available computational resources, while DF updates occur at any reobservation of initialized features.

6.2 Experimental Evaluation

The experimental evaluation was performed with an Ascending Technologies Pelican quadrotor equipped with a PointGrey Chameleon camera, a Navchip IMU and a VersaLogic Core 2 Duo single board computer. For the purposes of this experiment, the onboard computing platform was used only for measurement logging and the quadrotor platform was simply carried along the trajectory. Note that the computational and sensing equipment does not exceed the weight or power limits of the quadrotor and it is still capable of flight. The platform traveled a total distance of 172.5 meters over two floors at the Walter Library at University of Minnesota. IMU signals were sampled at a frequency of 100 Hz while camera images were acquired at 7.5 Hz. The trajectory traversed in our experiment consisted of three passes over a loop that spanned two floors, so three loop-closure events occurred. The quadrotor was returned to its starting location at the end of the trajectory, to provide a quantitative characterization of the achieved accuracy.

Opportunistic features were tracked using a window of $m = 10$ images. Every m camera frames, up to 30 features from all available DFs are initialized and the state vector is augmented with their 3D coordinates. The process of initializing DFs [9] is

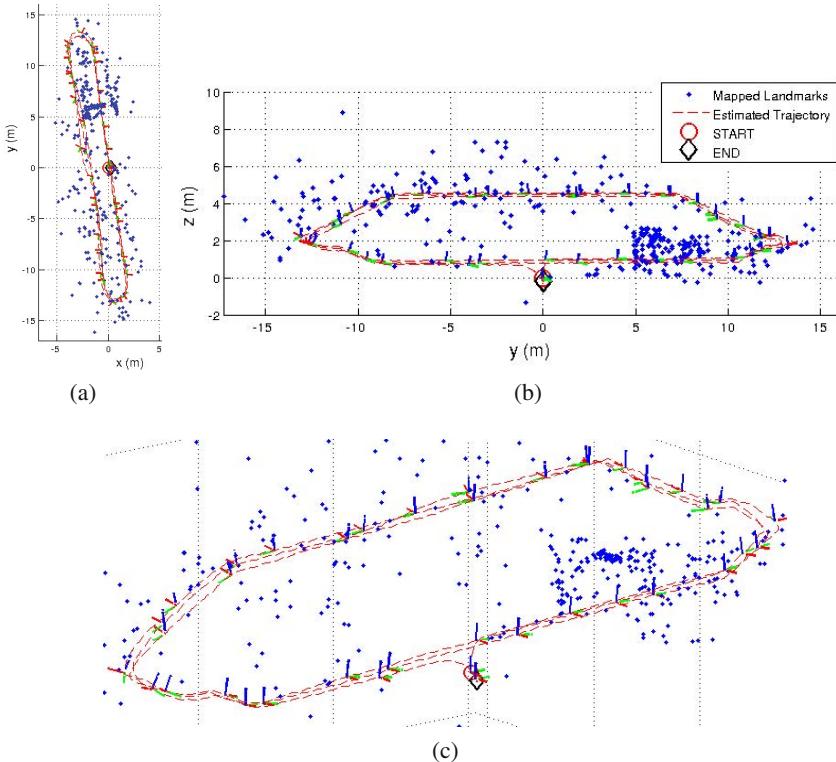


Fig. 3 The estimated 3D trajectory over the three traversals of the two floors of the building, along with the estimated positions of the mapped landmarks. (a) projection on the x and y axes, (b) projection on the y and z axes, (c) 3D view of the overall trajectory and estimated features.

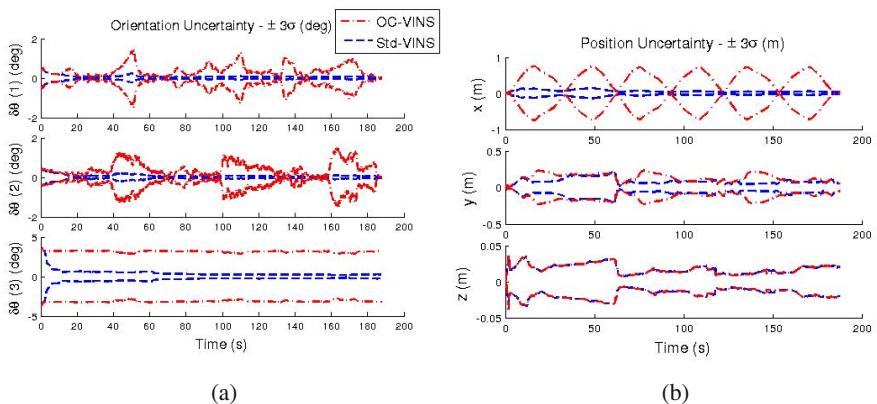


Fig. 4 Comparison of the estimated 3σ error bounds for attitude and position between Std-VINS and OC-VINS

continued until the occurrence of the first loop closure; from that point, no new DFs are considered and the filter relies upon the re-observation of previously initialized DFs and the processing of OFs.

For both the Std-VINS and the OC-VINS, the final position error was approximately 34 cm, which is less than 0.2% of the total distance traveled (see Fig. 3). However, the estimated covariances from the Std-VINS are smaller than those from the OC-VINS (see Fig. 4). Furthermore, uncertainty estimates from the Std-VINS decreased in directions that are unobservable (i.e., rotations about the gravity vector); this violates the observability properties of the system and demonstrates that spurious information is injected to the filter.

Figure 4(a) highlights the difference in estimated yaw uncertainty between the OC-VINS and the Std-VINS. In contrast to the OC-VINS, the Std-VINS covariance rapidly decreases, violating the observability properties of the system. Similarly, large differences can be seen in the covariance estimates for the x and y position estimates [see Fig. 4(b)]. The Std-VINS estimates a much smaller uncertainty than the OC-VINS, supporting the claim that Std-VINS tends to be inconsistent.

7 Conclusion and Future Work

In this paper, we analyzed the inconsistency of VINS from the standpoint of observability. Specifically, we showed that a standard VINS filtering approach leads to spurious information gain since it does not adhere to the unobservable directions of the true system. Furthermore, we introduced an observability-constrained VINS approach to mitigate estimator inconsistency by enforcing the unobservable directions explicitly. We presented both simulation and experimental results to support our claims and validate the proposed estimator.

In our future work, we are interested in analyzing additional sources of estimator inconsistency in VINS such as the existence of multiple local minima.

Acknowledgment. This work was supported by the University of Minnesota (UMN) through the Digital Technology Center (DTC) and the Air Force Office of Scientific Research (FA9550-10-1-0567). J. A. Hesch was supported by the UMN Doctoral Dissertation Fellowship.

References

1. Bar-Shalom, Y., Li, X.R., Kirubarajan, T.: *Estimation with Applications to Tracking and Navigation*. John Wiley & Sons, New York (2001)
2. Bouguet, J.-Y.: Camera calibration toolbox for matlab (2006)
3. Bryson, M., Sukkarieh, S.: Observability analysis and active control for airborne SLAM. *IEEE Trans. on Aerospace and Electronic Systems* 44(1), 261–280 (2008)
4. Chatfield, A.: Fundamentals of high accuracy inertial navigation. AIAA (American Institute of Aeronautics & Astronautics) (1997)
5. Corke, P., Lobo, J., Dias, J.: An introduction to inertial and visual sensing. *Int. Journal of Robotics Research* 26(6), 519–535 (2007)

6. Durrie, J., Gerritsen, T., Frew, E.W., Pledgie, S.: Vision-aided inertial navigation on an uncertain map using a particle filter. In: Proc. of the IEEE Int. Conf. on Robotics and Automation, Kobe, Japan, May 12-17, pp. 4189–4194 (2009)
7. Ebcin, S., Veth, M.: Tightly-coupled image-aided inertial navigation using the unscented Kalman filter. Technical report, Air Force Institute of Technology, Dayton, OH (2007)
8. Hermann, R., Krener, A.: Nonlinear controllability and observability. *IEEE Trans. on Automatic Control* 22(5), 728–740 (1977)
9. Hesch, J.A., Kottas, D.G., Bowman, S.L., Roumeliotis, S.I.: Observability-constrained vision-aided inertial navigation. Technical Report 2012-001, University of Minnesota, Dept. of Comp. Sci. & Eng., MARS Lab (February 2012)
10. Hesch, J.A., Mirzaei, F.M., Mariottini, G.L., Roumeliotis, S.I.: A Laser-aided Inertial Navigation System (L-INS) for human localization in unknown indoor environments. In: Proc. of the IEEE Int. Conf. on Robotics and Automation, Anchorage, AK, May 3-8, pp. 5376–5382 (2010)
11. Huang, G.P., Mourikis, A.I., Roumeliotis, S.I.: A first-estimates Jacobian EKF for improving SLAM consistency. In: Proc. of the Int. Symposium on Experimental Robotics, Athens, Greece, July 14-17, pp. 373–382 (2008)
12. Huang, G.P., Mourikis, A.I., Roumeliotis, S.I.: Observability-based rules for designing consistent EKF SLAM estimators. *Int. Journal of Robotics Research* 29(5), 502–528 (2010)
13. Huang, G.P., Trawny, N., Mourikis, A.I., Roumeliotis, S.I.: Observability-based consistent EKF estimators for multi-robot cooperative localization. *Autonomous Robots* 30(1), 99–122 (2011)
14. Isidori, A.: *Nonlinear Control Systems*. Springer (1989)
15. Jones, E.S., Soatto, S.: Visual-inertial navigation, mapping and localization: A scalable real-time causal approach. *Int. Journal of Robotics Research* 30(4), 407–430 (2011)
16. Kelly, J., Sukhatme, G.S.: Visual-inertial sensor fusion: Localization, mapping and sensor-to-sensor self-calibration. *Int. Journal of Robotics Research* 30(1), 56–79 (2011)
17. Kim, J., Sukkarieh, S.: Real-time implementation of airborne inertial-SLAM. *Robotics and Autonomous Systems* 55(1), 62–71 (2007)
18. Li, M., Mourikis, A.I.: Improving the accuracy of EKF-based visual-inertial odometry. In: Proc. of the IEEE Int. Conf. on Robotics and Automation, Minneapolis, MN, May 14-18, pp. 828–835 (2012)
19. Lowe, D.G.: Distinctive image features from scale-invariant keypoints. *Int. Journal of Computer Vision* 60(2), 91–110 (2004)
20. Lucas, B., Kanade, T.: An iterative image registration technique with an application to stereo vision. In: Proc. of the Int. Joint Conf. on Artificial Intelligence, Vancouver, B.C., Canada, August 24-28, pp. 674–679 (1981)
21. Lupton, T., Sukkarieh, S.: Visual-inertial-aided navigation for high-dynamic motion in built environments without initial conditions. *IEEE Trans. on Robotics* 28(1), 61–76 (2012)
22. Martinelli, A.: Vision and IMU data fusion: Closed-form solutions for attitude, speed, absolute scale, and bias determination. *IEEE Trans. on Robotics* 28(1), 44–60 (2012)
23. Maybeck, P.S.: *Stochastic models, estimation, and control*, vol. I. Academic Press, New York (1979)
24. Mirzaei, F.M., Roumeliotis, S.I.: A Kalman filter-based algorithm for IMU-camera calibration: Observability analysis and performance evaluation. *IEEE Trans. on Robotics* 24(5), 1143–1156 (2008)
25. Mourikis, A.I., Roumeliotis, S.I.: A multi-state constraint Kalman filter for vision-aided inertial navigation. In: Proc. of the IEEE Int. Conf. on Robotics and Automation, Rome, Italy, April 10-14, pp. 3565–3572 (2007)

26. Mourikis, A.I., Roumeliotis, S.I.: A dual-layer estimator architecture for long-term localization. In: Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition Workshops, Anchorage, AK, pp. 1–8 (June 2008)
27. Nistér, D.: An efficient solution to the five-point relative pose problem. In: Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition, Madison, WI, June 16-22, pp. 195–202 (2003)
28. Nistér, D., Stewénius, H.: Scalable recognition with a vocabulary tree. In: Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition, New York, NY, June 17-22, pp. 2161–2168 (2006)
29. Roumeliotis, S.I., Burdick, J.W.: Stochastic cloning: A generalized framework for processing relative state measurements. In: Proc. of the IEEE Int. Conf. on Robotics and Automation, Washington D.C., May 11-15, pp. 1788–1795 (2002)
30. Shen, S., Michael, N., Kumar, V.: Autonomous multi-floor indoor navigation with a computationally constrained MAV. In: Proc. of the IEEE Int. Conf. on Robotics and Automation, Shanghai, China, May 9-13, pp. 20–25 (2011)
31. Shi, J., Tomasi, C.: Good features to track. In: Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition, Washington, DC, June 27-July 2, pp. 593–600 (1994)
32. Strelow, D.W.: Motion estimation from image and inertial measurements. PhD thesis, Carnegie Mellon University, Pittsburgh, PA (November 2004)
33. Teddy Yap, J., Li, M., Mourikis, A.I., Shelton, C.R.: A particle filter for monocular vision-aided odometry. In: Proc. of the IEEE Int. Conf. on Robotics and Automation, Shanghai, China, May 9-13, pp. 5663–5669 (2011)
34. Trawny, N., Roumeliotis, S.I.: Indirect Kalman filter for 3D attitude estimation. Technical Report 2005-002, University of Minnesota, Dept. of Comp. Sci. & Eng., MARS Lab (March 2005)
35. Triggs, B., McLauchlan, P.F., Hartley, R.I., Fitzgibbon, A.W.: Bundle Adjustment – A Modern Synthesis. In: Triggs, B., Zisserman, A., Szeliski, R. (eds.) ICCV-WS 1999. LNCS, vol. 1883, pp. 298–372. Springer, Heidelberg (2000)
36. Weiss, S., Achtelik, M.W., Chli, M., Siegwart, R.: Versatile distributed pose estimation and sensor self-calibration for an autonomous MAV. In: Proc. of the IEEE Int. Conf. on Robotics and Automation, St. Paul, MN, May 14-18 (2012)
37. Williams, B., Hudson, N., Tweddle, B., Brockers, R., Matthies, L.: Feature and pose constrained visual aided inertial navigation for computationally constrained aerial vehicles. In: Proc. of the IEEE Int. Conf. on Robotics and Automation, Shanghai, China, May 9-13, pp. 431–438 (2011)

Learning to Segment and Track in RGBD

Alex Teichman and Sebastian Thrun

Abstract. We consider the problem of segmenting and tracking deformable objects in color video with depth (RGBD) data available from commodity sensors such as the Kinect. We frame this problem with very few assumptions - no prior object model, no stationary sensor, no prior 3D map - thus making a solution potentially useful for a large number of applications, including semi-supervised learning, 3D model capture, and object recognition. Our approach makes use of a rich feature set, including local image appearance, depth discontinuities, optical flow, and surface normals to inform the segmentation decision in a conditional random field model. In contrast to previous work, the proposed method *learns* how to best make use of these features from ground-truth segmented sequences. We provide qualitative and quantitative analyses which demonstrate substantial improvement over the state of the art.

1 Introduction

The availability of commodity depth sensors such as the Kinect opens the door for a number of new approaches to important problems in robot perception. In this paper, we consider the task of propagating an object segmentation mask through time. We assume a single initial segmentation is given and do not allow additional segmentation hints. In this work the initial segmentation is provided by human labeling, but this input could easily be provided by some automatic method, depending on the application. We do not assume the presence of a pre-trained object model (*i.e.* as the Kinect models human joint angles), as that would preclude the system from segmenting and tracking arbitrary objects of interest. As such, this task falls into the category of *model-free* segmentation and tracking, *i.e.* no prior class model

Alex Teichman · Sebastian Thrun
Stanford University, Stanford, CA 94305, USA
e-mail: teichman@cs.stanford.edu, thrun@stanford.edu

is assumed. Similarly, we do not assume the sensor is stationary or that a pre-built static environment map is available. There are several reasons a complete solution to this task would be useful in robotics, computer vision, and graphics.

First, model-free segmentation and tracking opens the door for a simple and effective method of semi-supervised learning in which a large number of unlabeled tracks of objects are used in conjunction with a small number of hand-labeled tracks to learn an accurate classifier. This method, known as tracking-based semi-supervised learning, was demonstrated in the autonomous driving context using laser range finders to learn to accurately recognize pedestrians, bicyclists, and cars versus other distractor objects in the environment using very few hand-labeled training examples [23]. However, model-free segmentation and tracking was more or less freely available because objects on streets generally avoid collision and thus remain depth-segmentable using simple algorithms. To extend tracking-based semi-supervised learning to the more general case in which objects are not easily depth-segmentable, more advanced model-free segmentation and tracking methods such as that of this paper are necessary.

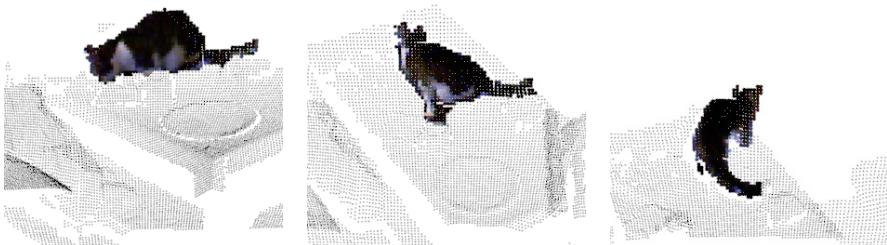


Fig. 1 Given an initial seed labeling (first frame; cat on a coffee table), the goal is to produce a detailed segmentation of the object over time (subsequent frames) without assuming a rigid object or a pre-trained object model. Foreground points are shown in bold.

Second, a new approach to object recognition with compelling benefits would be made possible. Currently, most object recognition methods fall into semantic segmentation or sliding window categories. In this new approach, objects are segmented and tracked over time and the track as a whole is classified online as new segmented frames stream in. This enables the classifier to use multiple views over time and to use track descriptors such as average object speed or maximum angular velocity. Some examples of this track-based approach exist, but only when special assumptions about the environment can be made, such as in [22].

Third, training data for object recognition tasks could easily be acquired with such a system. A few sparse labels provided by a user can be used to segment the remainder of the object track, providing new training examples with different object poses, view angles, lighting conditions, and so on, with little additional human effort. This same method could be used for 3D model-capture in unstructured environments, with all of the segmentation masks of a sequence being synthesized into one coherent object seen from all sides.

2 Previous Work

While there is much previous work on tracking in general, our problem's online nature, output of detailed segmentation masks rather than bounding boxes, and lack of simplifying assumptions restrict the directly-related previous work substantially. The most similar and recent work to ours is HoughTrack [7], which uses Hough Forests and GrabCut to segment consecutive frames. We provide a quantitative comparison to HoughTrack in Section 4.1. Older previous work on this problem include [3] and their multi-object extension [4], which use a generative model and level sets; [11], which tracks local object patches using Basin Hopping Monte Carlo sampling; and [15], which uses a conditional random field model and loopy belief propagation. All these works limit themselves to only a small number of features and do not use learning methods to find the best general-purpose tracker. Additionally, none of these works consider depth information.

We now briefly review previous work on related problems.

Model-Based Tracking - For some tasks, it is appropriate to model a specific object that is to be tracked, either with an explicit 3D model as in [14] or with pre-trained statistical models that are specific to the particular object class. Examples of the latter approach include the human pose tracking of the Kinect [18] and model-based car tracking in laser data for autonomous driving [13].

Interactive Segmentation - Human-assisted object segmentation has been the subject of much work including interactive graph cuts [5], GrabCut [16], and Video SnapCut [2]. These methods are largely intended for graphics or assisted-labeling applications as they require a human in the loop.

Bounding Box Tracking - Discriminative tracking [8, 19, 10] addresses the online, model-free tracking task, but where the goal is to track arbitrary objects in a bounding box rather than provide a detailed segmentation mask.

Rigid Object Tracking - Rigid object tracking using depth information, such as the open source method in PCL [17], addresses a similar but simpler problem, as it is not designed to work with deformable objects.

Offline Methods - The work of [6] takes as input segmentations for the first and last frames in a sequence, rather than just the first frame. While [24] takes as input only the segmentation of the first frame, they construct a CRF on the entire video sequence, with CRF labels corresponding to object flows. Here we are interested in a method that can update as new data streams in, thus making it applicable for robotics or other online vision tasks.

Background Subtraction - Finally, background subtraction approaches can greatly simplify the segmentation task. For example, [1] assumes a stationary camera to produce fine-grained segmentation masks for multiple objects. With depth sensors, background subtraction methods can also operate while the sensor is in motion: Google's autonomous car project uses pre-built 3D static environment maps to subtract away all 3D points except the moving obstacles [26]. This enables simple depth segmentation to be effective as long as moving objects do not touch, but assumes a high-precision localization system as well as a pre-learned static environment map.

3 Approach

There are a large number of possible cues that could inform a segmentation and tracking algorithm. Optical flow, image appearance, 3D structure, depth discontinuities, color discontinuities, etc., all provide potentially useful information. For example:

- An optical flow vector implies that the label of the source pixel at time $t - 1$ is likely to propagate to that of the destination pixel at time t .
- Pixels with similar color and texture to previous foreground examples are likely to be foreground.
- The shape of the object in the previous frame is likely to be similar to the shape of the object in the next frame.
- Points nearby in 3D space are likely to share the same label.
- Nearby points with similar colors are likely to share the same label.

Previous work generally focuses on a few particular features; here, we advocate the use of a large number of features. The above intuitions can be readily encoded by node and edge potentials in a conditional random field model. However, this additional complexity introduces a new problem: how much importance should each feature be assigned? While using a small number of features permits their weights to be selected by hand or tested with cross validation, this quickly becomes impractical as the number of features increases.

The margin-maximizing approach of structural SVMs [21, 25], adapted to use graph cuts for vision in [20], provides a solution to learning in these scenarios. Intuitively, this method entails running MAP inference, and then adding constraints to an optimization problem which assert that the margin between the ground truth labeling and the generated (and generally incorrect) labeling should be as large as possible. The application of this approach will be discussed in detail in Section 3.2.

3.1 Conditional Random Fields and Inference

A conditional random field is an undirected graphical model that can make use of rich feature sets and produce locally-consistent predictions. It aims to spend modeling power on only the distribution of the target variables given the observed variables. In particular, the conditional random field takes the form

$$\mathbb{P}(y|x) = \frac{1}{Z(x)} \exp(-E(y,x)), \quad (1)$$

where Z is the normalizer or partition function, $y \in \{-1, +1\}^n$ is the segmentation for an image with n pixels, and x is a set of features defined for the RGBD frame, to be detailed later. The energy function E contains the features that encode various

intuitions about what labels individual pixels should take and which pixels should share the same labels. In particular, the energy function is defined as

$$E(y, x) = \sum_{i \in \Phi_v} w_i \sum_{j \in v_i} \phi_j^{(i)}(y, x) + \sum_{i \in \Phi_e} w_i \sum_{(j, k) \in \mathcal{N}_i} \phi_{jk}^{(i)}(y, x). \quad (2)$$

Here, Φ_v is the set of node potential indices (*i.e.* one for each type of node potential such as local image appearance or 3D structure alignment), v_i is the set of all node indices for this potential type (normally one per pixel), and $\phi_j^{(i)}$ is the node potential of type i at pixel j . Similarly, Φ_e is the set of edge potential indices, \mathcal{N}_i is the neighborhood system for edge potential type i (normally pairs of neighboring pixels), and $\phi_{jk}^{(i)}$ is the edge potential between pixels j and k for edge potentials of type i . Thus, the weights apply at the feature-type level, *i.e.* w_i describes how important the i th feature is.

The goal of MAP inference in this model is to choose the most likely segmentation y given the features x by solving

$$\underset{y}{\text{maximize}} \quad \mathbb{P}(y|x) = \underset{y}{\text{minimize}} \quad E(y, x). \quad (3)$$

During inference, the weights w remain fixed. The solution to this problem can be efficiently computed for $y \in \{-1, +1\}^n$ and submodular energy function E using graph cuts [5].

3.2 Learning

Given a dataset of (y_m, x_m) for $m = 1 \dots M$, the goal of CRF learning is to choose the weights w that will result in the lowest test error. While it would be desirable to learn the weights directly using the maximum likelihood approach

$$\underset{w}{\text{maximize}} \quad \prod_m \mathbb{P}(y_m|x_m), \quad (4)$$

this is often intractable because of the presence of the partition function $Z(x) = \sum_y \exp(-E(y, x))$ in the gradient. This function sums over all 2^n segmentations of an image with n pixels.

Fortunately, there is an alternative approach known as the structural support vector machine [21, 25, 20], which we now briefly review. Solving the margin maximizing optimization problem

$$\begin{aligned} & \underset{w, \xi}{\text{minimize}} \quad \frac{1}{2} \|w\|^2 + \frac{C}{M} \sum_{m=1}^M \xi_m \\ & \text{subject to} \quad \xi \geq 0 \\ & \quad E(y_m, x_m) - E(y_m, x_m) \geq \Delta(y_m, y) - \xi_m \quad \forall m, \forall y \in \mathcal{Y} \end{aligned} \quad (5)$$

would result in a good solution. Here, C is a constant, Δ is a loss function, ξ_m is a slack variable for training example m , and \mathcal{Y} is the set of all possible labelings of an image. This problem, too, is intractable because it has exponentially many constraints. However, one can iteratively build a small, greedy approximation to the exponential set of constraints such that the resulting weights w are good.

Further, one can transform (5), known as the n -slack formulation, into the equivalent problem

$$\begin{aligned} & \underset{w, \xi}{\text{minimize}} && \frac{1}{2} \|w\|^2 + C\xi \\ & \text{subject to} && \xi \geq 0 \\ & && \frac{1}{M} \sum_{m=1}^M E(\hat{y}_m, x_m) - E(y_m, x_m) \geq \frac{1}{M} \sum_{m=1}^M \Delta(y_m, \hat{y}_m) - \xi \\ & && \forall (\hat{y}_1, \dots, \hat{y}_M) \in \mathcal{Y}^M, \end{aligned} \tag{6}$$

as described in [9]. As before, while the number of constraints is exponential, one can build a small, greedy approximation that produces good results. Known as the 1-slack formulation, this problem is equivalent and can be solved much more efficiently than (5), often by one or two orders of magnitude.

The goal is to learn the weights that will best segment an entire sequence given a single seed frame. However, the segmenter operates on a single frame at a time, so our training dataset must be in the form of (y, x) pairs. As will be discussed in Section 3.3, some of the features are stateful, *i.e.* they depend on previous segmentations. This presents a minor complication. At training time, we do not have access to a good choice of weights w , but the features depend on previous segmentations which in turn depend on the weights. To resolve this, we adopt the simple strategy of generating features assuming that previous segmentations were equal to ground truth. This dataset generation process is specified in Algorithm 1.

Algorithm 1. Training set generation

```

 $\mathbb{S} = \{S : S = ((y_0, d_0), (y_1, d_1), \dots)\}$  is a set of sequences,
    where  $y_i$  is a ground truth segmentation and  $d_i$  is an RGBD frame
 $\mathcal{D} = \emptyset$ 
for  $S \in \mathbb{S}$  do
    Initialize stateful components, e.g. the patch classifier that learns its model online
    for  $(y_i, d_i) \in S$  do
        Update stateful components using  $y_{i-1}$  as the previous segmentation
        Generate features  $x$ 
         $\mathcal{D} := \mathcal{D} \cup \{(y_i, x)\}$ 
    end for
end for
return  $\mathcal{D}$ 

```

Algorithm 2. Structural SVM for learning to segment and track

\mathcal{D} is a set of training examples (y, x) , formed as described in Algorithm 1.
 C and ε are constants, chosen by cross validation.

$\mathcal{W} \leftarrow \emptyset$
repeat

Update the parameters w to maximize the margin.

$$\underset{w, \xi}{\text{minimize}} \quad \frac{1}{2} \|w\|^2 + C\xi$$

$$\text{subject to} \quad w \geq 0, \quad \xi \geq 0$$

$$\frac{1}{M} \sum_{m=1}^M E(\hat{y}_m, x_m) - E(y_m, x_m) \geq \frac{1}{M} \sum_{m=1}^M \Delta(y_m, \hat{y}_m) - \xi$$

$$\forall (\hat{y}_1, \dots, \hat{y}_M) \in \mathcal{W}$$

for $(y_m, x_m) \in \mathcal{D}$ **do**

Find the MAP assignment using graph cuts.

$$\hat{y}_m \leftarrow \operatorname{argmin}_y E(y, x_m)$$

end for

$$\mathcal{W} \leftarrow \mathcal{W} \cup \{(\hat{y}_1, \dots, \hat{y}_M)\}$$

until $\frac{1}{M} \sum_{m=1}^M \Delta(y_m, \hat{y}_m) - E(\hat{y}_m, x_m) + E(y_m, x_m) \leq \xi + \varepsilon$

The structural SVM solver is detailed in Algorithm 2. Because the graph cuts solver assumes a submodular energy function, non-negative edge weights w_i for all $i \in \Phi_{\mathcal{E}}$ are required. As our node potentials are generally designed to produce values with the desired sign, we constrain them in Algorithm 2 to have non-negative weights as well. The term $\Delta(y_m, y)$ is 0-1 loss, but a margin rescaling approach could easily be obtained by changing Δ to Hamming loss and making a small modification to the graph cuts solver during learning (see [20]).

3.3 Energy Function Terms

We now review the particular choice of energy function (2) that we use in our implementation. The particular choice of features can be modified or added to without substantially changing the underlying framework. See Figure 2 for visualizations. All node potentials are designed to be constrained to $[-1, +1]$, and all edge potentials to $[-1, 0]$. While not strictly necessary, this aids in interpreting the weights learned by the structural SVM.

3.3.1 Node Potentials

Node potentials capture several aspects of shape, appearance, and motion. All take the form

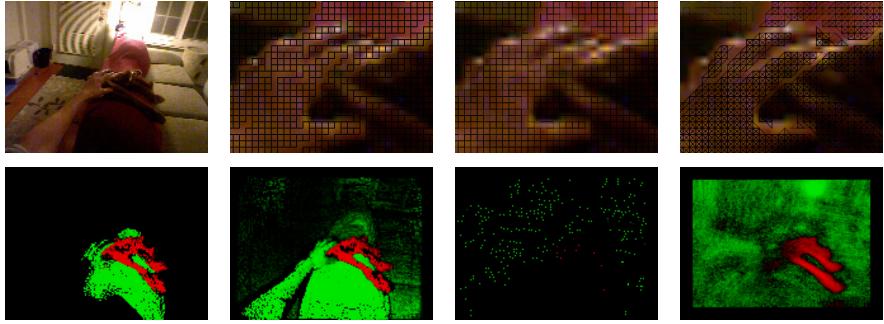


Fig. 2 Visualizations of selected edge and node potentials in the CRF. Edge potentials are zoomed in to show fine structure. Strong edge potentials are darker, while weaker edge potentials fade to original image color. Node potentials expressing a preference for foreground are shown in red, for background in green, and for neither in black. Top row: original image, canny edge potentials, color distance, depth edge potentials. Bottom row: ICP, frame alignment bilateral filter, optical flow, and patch classifier node potentials. Best viewed in color.

$$\phi_j(y, x) = \begin{cases} a_j & \text{if } y_j = 1 \\ b_j & \text{if } y_j = -1, \end{cases} \quad (7)$$

where a_j and b_j are functions of the data x , and $y_j = 1$ indicates that pixel j has been assigned to the foreground.

Seed Labels - The first frame of every sequence provides a foreground/background segmentation. Seed labels can also be used for algorithm-assisted human labeling, used here only in construction of the ground truth dataset. Each pixel in the image can be labeled by the user as foreground or background, or left unlabeled. Concretely, let $s_j \in \{-1, 0, +1\}$ be the seed labels for each pixel. Then, the potential is defined by

$$\phi_j(y, x) = \begin{cases} -1 & \text{if } y_j = s_j \\ 0 & \text{otherwise.} \end{cases}$$

Optical Flow - Pixels in the current frame with optical flow vectors from the previous frame are likely to have the label of the originating pixel. If no optical flow vector terminates at pixel j in the current frame, let $f_j = 0$. Otherwise, let f_j be equal to the label of the originating pixel from the previous frame. Then, the potential is defined by

$$\phi_j(y, x) = \begin{cases} -1 & \text{if } y_j = f_j \\ 0 & \text{otherwise.} \end{cases}$$

Frame Alignment Bilateral Filter - Optical flow provides correspondences from one frame to the next; after aligning these frames, a bilateral filter given 3D position

and RGB values is used to blur the labels from the previous frame into the current frame, respecting color and depth boundaries.

Specifically, alignment between the two frames is found using a RANSAC method, with 3D correspondences given by optical flow vectors. Flow vectors that originate or terminate on a depth edge are rejected, as their 3D location is often unstable. After alignment, we find the set of neighboring points N from the previous frame within a given radius of the query point, then compute the value

$$z_j = \sum_{k \in N} y'_k \exp \left(-\frac{\|c_j - c_k\|_2}{\sigma_c} - \frac{\|p_j - p_k\|_2}{\sigma_d} \right),$$

where y'_k is the $\{-1, +1\}$ label of point k in the previous segmentation, c_j is the RGB value of pixel j , and p_j is the 3D location of pixel j . The two σ s are bandwidth parameters, chosen by hand. The value z_j is essentially that computed by a bilateral filter, but without the normalization. This lack of normalization prevents a point with just a few distant neighbors from being assigned an energy just as strong as a point with many close neighbors. Referring to (7), the final energy assignment is made by setting $b_j = 0$ and $a_j = 1 - 2/(1 + e^{-z_j})$.

Patch Classifiers - Two different parameterizations of a random fern patch classifier similar to [12] are trained online and used to make pixel-wise predictions based on local color and intensity information.

A random fern classifier is a semi-naive Bayes method in which random sets of bit features are chosen (“ferns”), and each possible assignment to the bit features in the fern defines a bin. Each of these bins maintains a probability estimate by simple counting. The final probability estimate is arrived at by probabilistically combining the outputs of many different randomly generated ferns. See [12] for details. We use 50 randomly generated ferns, each of which have 12 randomly generated features.

Each bit feature in a fern is generated by very simple functions of the data contained in the image patch. For example, one of the bit features we use is defined by

$$z = \begin{cases} 1 & \text{if } I_{p_0} < I_{p_1} \\ 0 & \text{otherwise,} \end{cases}$$

where p_0 and p_1 are two pre-determined points in the image patch, and I_p is the intensity of the camera image at point p . Other bit features include color comparisons and Haar wavelet comparisons, efficiently computed using the integral image.

At each new frame, the classifier is updated using the graph cuts segmentation output from the previous frame as training data, then is used to compute a probability of foreground estimate for every pixel in the current frame. The value a_j is set to the probability of background, and b_j to probability of foreground.

Distance from Previous Foreground - Points very far away from the object’s position in the previous frame are unlikely to be foreground.

After alignment of the current frame to the previous frame using the same optical-flow-driven method as the bilateral term, we compute the distance d from the point j

in the current frame to the closest foreground point in the previous frame. The final potentials are set to $a_j = 0$ and $b_j = \exp(-d/\sigma) - 1$.

ICP - The iterative closest point algorithm is used to fit the foreground object's 3D points (and nearby background points) from the previous frame to the current frame. If the alignment has a sufficient number of inliers, points in the current frame are given node potentials similar to those of the bilateral node potential, but where the ICP-fit points are used instead of the entire previous frame.

Prior Term - To express a prior on background, we add a term for which $a_j = 0$ and $b_j = -1$.

3.3.2 Edge Potentials

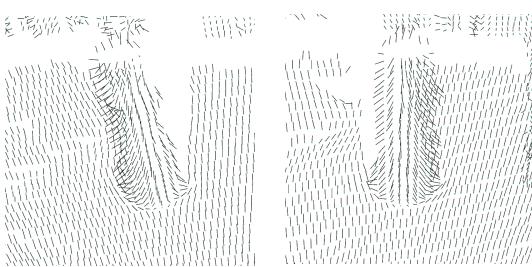


Fig. 3 While image appearance and depth information may sometimes provide little information about an object boundary, a change in surface normals can be informative. Surface normals are shown as small dark lines, seen from two angles to aid in depth perception.

Edge potentials capture the intuition that the foreground/background boundary is likely to be at an image gradient, color change, depth discontinuity, or surface normal change. All take the form

$$\phi_{jk}(y, x) = \begin{cases} a_{jk} & \text{if } y_j = y_k \\ 0 & \text{otherwise,} \end{cases}$$

where a_{jk} is a function of the data x . All edges are between neighboring points in the image.

Canny Edges - All neighboring pixels are connected by an edge potential except those cut by canny edges. Concretely,

$$a_{jk} = \begin{cases} -1 & \text{if neither pixel lies on a Canny edge} \\ 0 & \text{otherwise.} \end{cases}$$

Color Distance - Edge weights are assigned based on Euclidean distance between neighboring RGB values; $a_{jk} = -\exp(-||c_j - c_k||/\sigma)$, where c_j and c_k are the RGB values of the two pixels and σ is a bandwidth parameter, chosen by hand.

3D Distance - Points nearby in 3D are likely to share the same label, especially if they lie in the same plane. Out-of-plane distance changes are penalized more heavily than in-plane distance changes. Specifically,

$$a_{jk} = -\exp\left(-\frac{|(p_j - p_k)^T n_k|}{\sigma_n} - \frac{\|p_j - p_k\|_2}{\sigma_d}\right),$$

where p_j and p_k are the neighboring 3D points, n_k is the surface normal at point p_k , and the σ s are bandwidth parameters chosen by hand.

Surface Normals - Neighboring points that share the same surface normal are likely to have the same label. See Figure 3. We use $a_{jk} = -\exp(-\theta/\sigma)$, where θ is the angle between the two normals and σ is a bandwidth parameter chosen by hand.

Edge Potential Products - Several combinations of the above are also provided, taking the form $a_{jk} = -|\prod_i a_{jk}^{(i)}|$, where $a_{jk}^{(i)}$ is the value from one of the above edge potentials. Intuitively, this encodes an edge potential that is strong (*i.e.* favors label equality) if all of the component edge potentials are strong.

4 Experiments

We provide a quantitative analysis to demonstrate improvement over the state of the art and a qualitative discussion the strengths and weaknesses of the current implementation. All experiments use 160x120 RGBD data. The structural SVM of Algorithm 2 was implemented with standard interior point solver techniques. The graph cuts solver of [5] was used for inference.

4.1 Quantitative Analysis

As there is no work we are aware of which does segmentation and tracking of non-rigid objects using learning or depth data, we compare against HoughTrack, discussed in Section 2. We used the implementation that accompanies [7]. To ensure a fair comparison, we modified the HoughTrack implementation to be initialized with a segmentation mask rather than a bounding box.

No RGBD segmentation and tracking dataset currently exists, so we generated one containing 28 fully-labeled sequences with a total of about 4000 frames. Objects include, for example, a sheet of paper, cat, jacket, mug, laptop, and hat. In general, the dataset includes rigid and non-rigid objects, and textured and non-textured objects. Ground truth was generated by hand labeling, assisted with an interactive version of the segmenter, similar to the interactive graph cuts work of [5].

The dataset was split into a training set of about 500 frames over 10 sequences and a testing set of about 3500 frames over 18 sequences. Training of our method was run once on the training set and the resulting segmenter was used to produce results for all testing sequences. HoughTrack has no training stage, and thus did not use the training set. Testing results were produced for both by providing an

initial segmentation for the first frame of each sequence, and all further frames were segmented without additional input.

Individual frames are evaluated with two different metrics. Hamming loss, or total number of pixels wrong, is simple and direct but ignores the size of the object; a Hamming loss of twenty could be negligible or a significant fraction of the object in question. To correct for this, we also report *normalized accuracy*, which is 1 if all pixels in the frame are correctly labeled and 0 if the number of incorrectly labeled pixels equals or exceeds the number of pixels in the foreground object. More precisely, normalized accuracy is $1 - \min(1, \text{num_wrong}/\text{num_fg})$. Sequence results are reported as the average of these values over all frames.

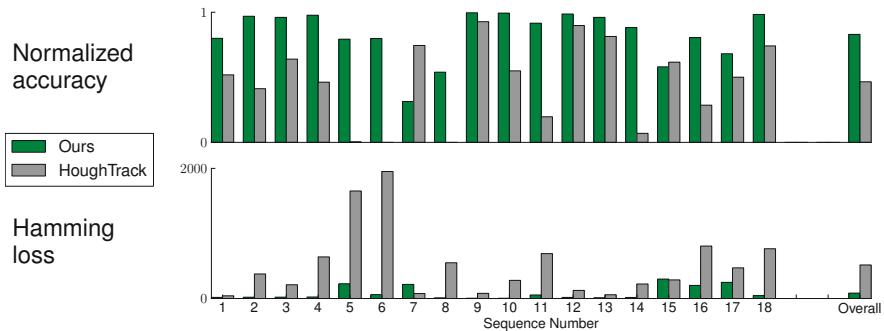


Fig. 4 Comparison of our method with the most similar state-of-the-art work

Overall, our method demonstrates a normalized error reduction of about 65% compared to the state-of-the-art. Detailed results can be seen in Figure 4. Training of our method takes a total of about 10 minutes, and as this need only be done once, this is acceptable. At runtime, HoughTrack’s implementation takes about 200ms per frame, whereas our method’s implementation takes about 1200ms per frame.¹ While this is currently too slow for real-time usage, almost all of this time is spent computing features and only about a millisecond is spent on the graph cuts solver. This means real-time performance can be achieved with speedups of feature computation alone, most immediately by parallelization and by computing features only at the boundaries of the object where they are actually needed.

4.2 Qualitative Analysis

4.2.1 Strengths

Our results demonstrate that this method can be effective even in sequences which include significant non-rigid object transformations, occlusion, and a lack of visually distinguishing appearance. As seen in the illustration in Figure 2, depth provides

¹ Both implementations were compiled with optimizations turned on, run on the same machine, and given one thread.



Fig. 5 Visualization of results in which image appearance alone would lead to a very difficult segmentation problem, but which becomes relatively easy when reasoning about depth as well

a very powerful cue as to what neighboring points are likely to share the same label. In addition to depth discontinuities, surface normals (Figure 3) can provide useful information about object boundaries, enabling the segmentation and tracking of objects for which using visual appearance alone would be extremely challenging, such as that of Figure 5. While this sequence would be relatively easy for off-the-shelf 3D rigid object trackers such as that in PCL [17], the tracking of deformable objects such as cats and humans would not be possible.

Our approach is general enough to handle both objects with few visually distinguishing characteristics and objects which are non-rigid; see Figure 6 for examples. In particular, the interface between the cat and the girl in sequences (A) and (B), and the interface between the cat and the bag in sequence (C) are maintained correctly. The method also works for objects without significant depth, such as the paper on the desk in sequence (G), and can recover from heavy occlusion as in sequence (F). The hat in sequence (H) undergoes deformation and heavy RGB blurring due to darkness of the environment, yet is tracked correctly. Sequence (D) shows the tracking of a piece of bread with Nutella while it is being eaten.

4.2.2 Weaknesses and Future Work

As is typical in tracking tasks, stability versus permissiveness is a tradeoff. In some cases, it is not well defined whether a recently-unoccluded, disconnected set of points near the foreground object should be part of the foreground or part of the background. An example of this occurs in sequence (E), when the cat's head becomes self-occluded, then unoccluded, and the system cannot determine that these points should be assigned to foreground. The error is then propagated. Similarly, the girl's far arm in sequence (A) is lost after being occluded and then unoccluded. Rapid motion appears to exacerbate this problem. While partial occlusion can be handled, as in sequence (F), the current implementation has no facility for re-acquisition if the target is completely lost. Finally, thin parts of objects are often problematic, as it seems the edge potentials are not well connected enough in these areas.

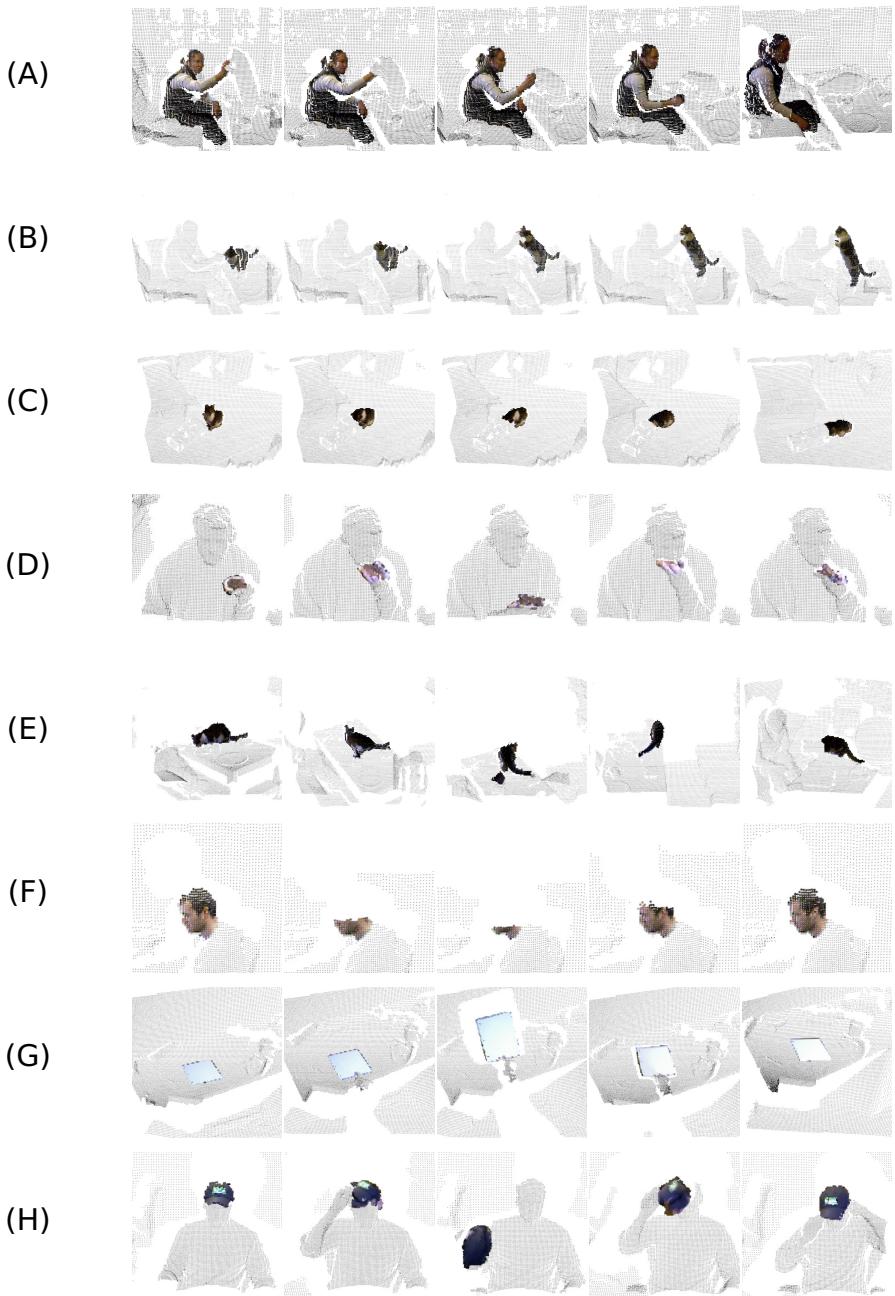


Fig. 6 Visualization of results. The first frame in each sequence (far left) is the seed frame. Foreground points are shown in bold and color while background points are shown small and gray. Best viewed on-screen.

It is likely that improvements to node and edge potentials could resolve these limitations. Using longer range edges (rather than a simple 2D grid connecting neighbors only) could improve performance on thin objects and quickly moving objects, and improvements in the image-appearance-based patch classifier could result in better segmentations of objects that self-occlude parts frequently. A node potential based on LINE-MOD or a discriminative tracker such as [10] could solve the re-acquisition problem, and could be run on object parts rather than the object as a whole to enable usage on non-rigid objects.

The feature-rich representation we use presents a challenge and an opportunity. There are a large number of parameters in the computation pipeline which cannot be learned via the structural SVM (*e.g.* the σ 's discussed in Section 3.3). There is also substantial freedom in the structure of the computation pipeline. Choosing the structure and the parameters by hand (as we have done here) is possible, but onerous; there is an opportunity to learn these in a way that maximizes accuracy while respecting timing constraints.

5 Conclusions

We have presented a novel method of segmenting and tracking deformable objects in RGBD, making minimal assumptions about the input data. We have shown this method makes a significant quantitative improvement over the most similar state-of-the-art work in segmentation and tracking of non-rigid objects. A solution to this task would have far-reaching ramifications in robotics, computer vision, and graphics, opening the door for easier-to-train and more reliable object recognition, model capture, and tracking-based semi-supervised learning. While there remains more work to be done before a completely robust and real-time solution is available, we believe this approach to be a promising step in the right direction.

References

1. Aeschliman, C., Park, J., Kak, A.: A probabilistic framework for joint segmentation and tracking. In: CVPR (2010)
2. Bai, X., Wang, J., Simons, D., Sapiro, G.: Video snapcut: robust video object cutout using localized classifiers. In: SIGGRAPH (2009)
3. Bibby, C., Reid, I.: Robust Real-Time Visual Tracking Using Pixel-Wise Posteriors. In: Forsyth, D., Torr, P., Zisserman, A. (eds.) ECCV 2008, Part II. LNCS, vol. 5303, pp. 831–844. Springer, Heidelberg (2008)
4. Bibby, C., Reid, I.: Real-time tracking of multiple occluding objects using level sets. In: CVPR (2010)
5. Boykov, Y., Kolmogorov, V.: An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. IEEE Transactions on Pattern Analysis and Machine Intelligence 26, 359–374 (2001)
6. Budvytis, I., Badrinarayanan, V., Cipolla, R.: Semi-supervised video segmentation using tree structured graphical models. In: CVPR (2011)

7. Godec, M., Roth, P., Bischof, H.: Hough-based tracking of non-rigid objects. In: ICCV (2011)
8. Grabner, H., Grabner, M., Bischof, H.: Real-Time Tracking via On-line Boosting. In: British Machine Vision Conference (2006)
9. Joachims, T., Finley, T., Yu, C.-N.: Cutting-plane training of structural svms. *Machine Learning* 77(1), 27–59 (2009)
10. Kalal, Z., Matas, J., Mikolajczyk, K.: P-n learning: Bootstrapping binary classifiers by structural constraints. In: CVPR (2010)
11. Kwon, J., Lee, K.M.: Tracking of a non-rigid object via patch-based dynamic appearance modeling and adaptive basin hopping monte carlo sampling. In: CVPR (2009)
12. Ozysal, M., Fua, P., Lepetit, V.: Fast keypoint recognition in ten lines of code. In: CVPR (2007)
13. Petrovskaya, A., Thrun, S.: Model based vehicle detection and tracking for autonomous urban driving. In: Autonomous Robots (2009)
14. Prisacariu, V.A., Reid, I.D.: Pwp3d: Real-time segmentation and tracking of 3d objects. In: BMVC (2009)
15. Ren, X., Malik, J.: Tracking as repeated figure/ground segmentation. In: CVPR (2007)
16. Rother, C., Kolmogorov, V., Blake, A.: “grabcut”: interactive foreground extraction using iterated graph cuts. In: ACM SIGGRAPH 2004 Papers (SIGGRAPH 2004), pp. 309–314. ACM, New York (2004)
17. Rusu, R.B., Cousins, S.: 3D is here: Point Cloud Library (PCL). In: IEEE International Conference on Robotics and Automation (ICRA), Shanghai, China, May 9-13 (2011)
18. Shotton, J., Fitzgibbon, A., Cook, M., Sharp, T., Finocchio, M., Moore, R., Kipman, A., Blake, A.: Real-time human pose recognition in parts from single depth images. In: CVPR (2011)
19. Stalder, S., Grabner, H., Gool, L.V.: Beyond semi-supervised tracking: Tracking should be as simple as detection, but not simpler than recognition. In: International Conference on Computer Vision Workshop on On-line Learning for Computer Vision (September 2009)
20. Szummer, M., Kohli, P., Hoiem, D.: Learning CRFs Using Graph Cuts. In: Forsyth, D., Torr, P., Zisserman, A. (eds.) ECCV 2008, Part II. LNCS, vol. 5303, pp. 582–595. Springer, Heidelberg (2008)
21. Taskar, B., Chatalbashev, V., Koller, D., Guestrin, C.: Learning structured prediction models: A large margin approach. In: ICML (2005)
22. Teichman, A., Levinson, J., Thrun, S.: Towards 3D object recognition via classification of arbitrary object tracks. In: International Conference on Robotics and Automation (2011)
23. Teichman, A., Thrun, S.: Tracking-based semi-supervised learning. In: Robotics: Science and Systems (2011)
24. Tsai, D., Flagg, M., Rehg, J.: Motion coherent tracking with multi-label mrf optimization. In: BMVC (2010)
25. Tsochantaridis, I., Joachims, T., Hofmann, T., Altun, Y.: Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research* 6, 1453–1484 (2005)
26. Urmson, C.: The Google Self-Driving Car Project (2011)

The Path Inference Filter: Model-Based Low-Latency Map Matching of Probe Vehicle Data

Timothy Hunter, Pieter Abbeel, and Alexandre M. Bayen

Abstract. We consider the problem of reconstructing vehicle trajectories from sparse sequences of GPS points, for which the sampling interval ranges between 10 seconds and 2 minutes. We introduce a new class of algorithms, collectively called *path inference filter* (PIF), that maps streaming GPS data in real-time, with a high throughput. We present an efficient Expectation Maximization algorithm to train the filter on new data without ground truth observations. The path inference filter is evaluated on a large San Francisco taxi dataset. It is deployed at an industrial scale inside the *Mobile Millennium* traffic information system, and is used to map fleets of vehicles in San Francisco, Sacramento, Stockholm and Porto.

1 Introduction

The paradigm of connected vehicles, the progressive integration of smartphones and car infrastructure, the rise of Web 2.0, and the progressive emergence of automation onboard vehicles have created a very fertile ground for GPS data sources from probe vehicles to be collected at an unprecedented scale. Yet, the lack of ubiquitous connectivity, the parsimonious use of bandwidth and smartphone battery limits, and the lack of system reliability often makes this data sparse. In particular, it is very common today for GPS traces to be sampled at very low frequencies (on the order of minutes), leading to challenges in using this data. While some studies [13] predict an 80% penetration of the cellphone market by GPS enabled devices by 2015, it is not clear that this figure will translate into ubiquitous GPS data for traffic information systems. Furthermore, millions of fleet vehicles today produce GPS traces sampled at low frequencies, see for example [4], and the paradigm of connected (and automated) vehicles does not automatically translate into high fidelity GPS traces.

Timothy Hunter · Pieter Abbeel · Alexandre M. Bayen
Department of Electrical Engineering and Computer Sciences,
University of California at Berkeley
e-mail: {tjhunter, pabbeel, bayen}@eecs.berkeley.edu

Two of the common problems which occur when dealing with these GPS traces are the correct mapping of these observations to the road network, and the reconstruction of the trajectories of the vehicles from these traces. This problem is similar to numerous other problems also encountered in robotics, for example tracking of *people of interest* (POI) by the military, in which POIs disappear between sparse samples and analysts must reconstruct the path followed.

We present a new class of algorithms, called the *path inference filter*, that solve this problem in an efficient way. There are two difficulties associated with this problem. First, there may be many possible projections of the noisy position onto the map. Second, the number of possible paths between consecutive pairs of positions is potentially very high (given that in urban environments, a vehicle could typically travel several blocks between measurements). Instead of handling both problems separately, as is commonly done in traffic modeling studies (which leads to significant pitfalls), the method solves both problems at once, which increases the efficiency of the algorithm and uses the conjunction of both unknowns (projections and paths) in a unified manner to resolve this inference problem.

Specific instantiations of this algorithm have been deployed as part of *Mobile Millennium*, which is a traffic estimation and prediction system developed at UC Berkeley [2]. *Mobile Millennium* infers real-time traffic conditions using GPS measurements from drivers running cell phone applications, taxicabs, and other mobile and static data sources. This system was initially deployed in the San Francisco Bay area and later expanded to other locations. In addition to GPS information, the data points collected by *Mobile Millennium* contain other attributes such as heading, speed, etc. We will show how this additional information can be integrated in the rest of the framework presented into this article.

In the case of high temporal resolution (typically, a frequency greater than an observation per second), some highly successful methods have been developed for continuous estimation [16]. In the case of low frequency sampled data, simple deterministic algorithms to reconstruct trajectories fail due to misprojection or shortcuts (Figure 1). Such shortcomings have motivated our search for a principled approach that jointly considers the mapping of observations to the network and the reconstruction of the trajectory.

Related Work. Researchers started systematic studies after the introduction of the GPS system to civilian applications in the 1990s [12]. Early geometry projection approaches were later refined to use more information such as heading and road curvature. This greedy matching, however, leads to poor trajectory reconstruction since it does not consider the path leading up to a point. New deterministic algorithms emerged to directly match partial trajectories to the road by using the topology of the network [6], and were soon expanded into probabilistic frameworks. A number of implementations were

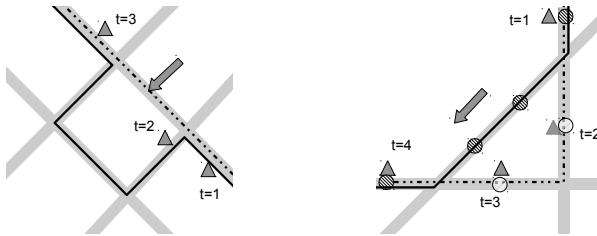


Fig. 1 Examples of failures when using intuitive algorithms to project each GPS measurement to the closest link. The raw GPS measurements are the triangles, the true trajectory is the dashed line, and the reconstructed trajectory is the continuous line. The hashed circles are the states chosen by this reconstruction algorithm. (a) The trajectory is reconstructed using a closest-point, shortest path algorithm. Due to GPS noise, the point at $t = 2$ is closer to the orthogonal road and forces the algorithm to add a wring left turn. (b) Example of error when considering the shortest paths with multiple potential projections.

explored, among other particle filters [7], Kalman filters [11], and Hidden Markov Models [3].

Two types of information are missing in a sequence of GPS readings: the exact location of the vehicle on the road network when the observation was emitted, and the path followed from the previous location to the new location. These problems are correlated. The aforementioned approaches focus on high-frequency sampling observations, for which the path followed is extremely short (less than a few hundred meters, with very few intersections). In this context, there is usually a dominant path that starts from a well-defined point, and Bayesian filters accurately reconstruct paths from the observations [11, 16]. When sampling rates are lower and observed points are further apart, however, a large number of paths are possible between two points. Researchers have recently focused on efficiently identifying these correct paths and have separated the joint problem of finding the paths and finding the projections into two distinct problems. The first problem is path identification and the second step is projection matching [3, 15]. Some interesting approaches mixing points and paths that use a voting scheme have also recently been proposed [19].

Contributions of the Article. The *path inference filter* is a probabilistic framework that aims at recovering trajectories and road positions from low-frequency probe data in real time. The performance of the filter degrades gracefully as the sampling frequency decreases, and it can be tuned to different scenarios (such as real time estimation with limited computing power or offline, high accuracy estimation).

The filter falls into the general class of *Conditional Random Fields* [9]. Our framework can be decomposed into the following steps:

- *Map matching*: each GPS measurement from the input is projected onto a set of possible candidate states on the road network.

- *Path discovery*: admissible paths are computed between pairs of candidate points on the road network.
- *Filtering*: probabilities are assigned to the paths and the points using both a stochastic model for the vehicle dynamics and probabilistic driver preferences learned from data.

The path inference filter presents a number of compelling advantages over the work found in the current literature:

1. The approach presents a general framework grounded in established statistical theory that encompasses numerous approaches presented in the literature. In particular, it combines information about paths, points and network topology in a single unified notion of *trajectory*.
2. Nearly all work on Map Matching is segmented into (and presents results for) either high-frequency or low-frequency sampling. After training, the path inference filter performs competitively across all frequencies.
3. As will be seen in Section 3, most existing approaches (which are based on Hidden Markov Models) do not work well at lower frequencies due to the *selection bias problem*. Our work directly addresses this problem by performing inference on a Random Field.
4. The path inference filter can be used with complex path models such as the ones used in [3]. In the present article, we restrict ourselves to a class of models (the exponential family distributions) that is rich enough to provide insight in the driving patterns of the vehicles. Furthermore, when using this class of models, the learning of new parameters leads to a convex problem formulation that is fast to solve. In addition, this algorithm efficiently learns parameters in an unsupervised setting.
5. With careful engineering, it is possible to achieve high mapping throughput on large-scale networks. The implementation in *Mobile Millennium* achieves an average throughput of hundreds of GPS observations per second on a single core in real time. Furthermore, the algorithm scales well on multiple cores and has achieved average throughput of several thousands of points per second on a multicore architecture.
6. The path inference filter is designed to work across the full spectrum of accuracy versus latency. As will be shown, approximate 2-lagged smoothing is nearly as precise as full smoothing: we can still achieve good mapping accuracy while delaying computations by only one or two time steps.

2 Path Discovery

The road network is described as a directed graph $\mathcal{N} = (\mathcal{V}, \mathcal{E})$ in which the nodes are the street intersections and the edges are the streets, referred to in the text as the *links* of the road network. Every location on the road network is completely specified by a given link l and a non-negative offset o on this link. At any time, the *state* x of a vehicle consists of its location on the road network and some other optional information such as speed, or heading.

For our example, we consider that the state is simply the location on one of the road links (which are directed): $x = (l, o)$.

From GPS Points to Discrete Vehicle States. The points are mapped to the road following a Bayesian formulation. This process is represented by a probability distribution $\omega(g|x)$ that, given a state x , returns a probability distribution over all possible GPS observations g . Such distributions ω will be described in Section 3. Additionally, we may have some *prior knowledge* over the state of the vehicle, coming for example from previous GPS observations. This knowledge can be encoded in a *prior distribution* $\Omega(x)$. Under this general setting, the state of a vehicle, given a GPS observation, can be computed using Bayes' rule: $\pi(x|g) \propto \omega(g|x) \Omega(x)$. The letter π will define general probabilities, and their dependency on variables will always be included. This posterior distribution is usually complicated, owing to the mixed nature of the state (links and offsets). Instead of working with the posterior density directly, we represent it by a discrete distribution over a few well-chosen representative locations: for each link l_i , one or more states from this link are elected to represent the posterior distribution of the states on this link $\pi(o|g, l = l_i)$. In our model, the distribution over the offsets is unimodal and can be approximated with a single sample taken at the most likely offset: $\forall l_i, o_{i\text{posterior}}^* = \text{argmax}_o \pi(o|g, l = l_i)$, which implicitly depends on the prior distribution Ω . In practice, the prior is slowly varying comparatively to the length of the link, and can be considered constant as a first approximation on each link. This is why we can approximate $o_{i\text{posterior}}^*$ by the most likely offset with respect to the observation distribution: $o_{i\text{posterior}}^* \approx o_{i\text{obs}}^* = \text{argmax}_o \omega(x = (o, l_i) | g)$. This approximation is illustrated in Figure 2.

In practice, the probability distribution $\pi(x|g)$ decays rapidly, and can be considered overwhelmingly small beyond a certain distance from the observation g . Links located beyond a certain radius need not be considered valid projection links, and may be discarded.

In the rest of the article, the boldface symbol \mathbf{x} will denote a (finite) collection of states associated with a GPS observation g that we will use to represent the posterior distribution $\pi(x|g)$, and the integer I will denote its cardinality: $\mathbf{x} = (x_i)_{1:I}$. These points are called *candidate state projections for the GPS measurement g* . These discrete points will then be linked together through trajectory information that takes into account the trajectory and the dynamics of the vehicle.

From Discrete Vehicle States to Trajectories. At each time step t , a GPS point g^t is observed. This GPS point is then mapped onto I^t different candidate states denoted $\mathbf{x}^t = x_1^t \cdots x_{I^t}^t$. Because this set of projections is

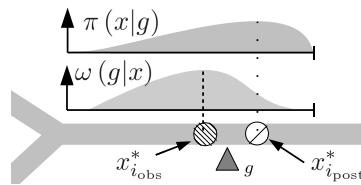


Fig. 2 Example of a measurement g on a link. The GPS measurement is the triangle denoted g .

finite, there is only a finite number J^t of paths that a vehicle can have taken while moving from some state $x_i^t \in \mathbf{x}^t$ to some state $x_{i'}^{t+1} \in \mathbf{x}^{t+1}$. We denote the set of *candidate paths* between the observation g^t and the next observation g^{t+1} by $\mathbf{p}^t = (p_j^t)_{j=1:J^t}$. Each path p_j^t goes from one of the projection states x_i^t of g^t to a projection state $x_{i'}^{t+1}$ of g^{t+1} . There may be multiple pairs of states to consider, and between each pair of states, there are typically several paths available (see Figure 3). Lastly, a *trajectory* is defined by the succession of states and paths, starting and ending with a state: $\tau = x_1 p_1 x_2 \cdots p_{t-1} x_t$ where x_1 is one element of \mathbf{x}^1 , p_1 of \mathbf{p}^1 , and so on.

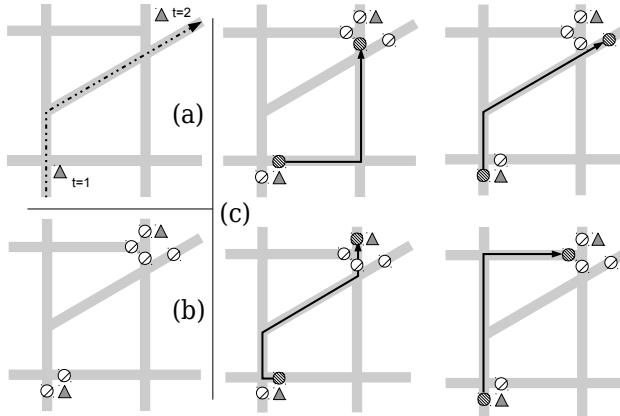


Fig. 3 Example of path exploration between two observations. On the upper left corner, the true trajectory and two associated GPS observations. On the lower left corner, the set of candidate projections associated with each observation. On the right, a few examples of computed paths.

Due to speed limits leading to upper bounds on achievable travel times on the network, there is only a finite number of paths a vehicle can take during a time interval Δt . Such paths can be computed using standard graph search algorithms. An algorithm that performs well in practice is the A* algorithm [8]. The cost metric we use here is the expected travel time on each link, and the heuristic is the shortest geographical distance, properly scaled so that it is an admissible heuristic. Due to the noise in the GPS observations, some failure cases need some special care. The supplementary materials available online (see the first page of the article) provide additional technical information about some ways to deal with these failure cases.

3 Discrete Filtering Using a Conditional Random Field

We have reduced the trajectory reconstruction problem to a discrete selection problem between candidate projection points interleaved with candidate paths. We now apply a probabilistic framework to infer the most likely trajectory τ^* or the marginal conditionals $\pi(x^t|g^{1:T})$ and $\pi(p^t|g^{1:T})$.



Fig. 4 Illustration of the graphical models used in the path inference filter (CRF, left), and commonly found in the literature (HMM, right), defined over a trajectory $\tau = x^1 p^1 x^2 p^2 x^3$ and a sequence of observations $g^{1:3}$

We endow the set of all possible paths on the road network with a probability distribution. The *transition model* η describes a distribution $\eta(p)$ defined over all possible paths p across the road network. This distribution is not a distribution over actually observed paths as much as a model of the *preferences* of the driver when given the choice between several options. In order to make the problem tractable, we introduce some conditional independences between states and paths: a path p^t is independent of all other paths and states given x^t and x^{t+1} , and a state x^t is completely known when either the preceding path p^{t-1} or the subsequent path p^t are known.

Each path must start at the start state and must end at the end state. We formally express the compatibility between a state x and the start and end states of a path p with the compatibility functions $\underline{\delta}$ and $\bar{\delta}$:

$$\underline{\delta}(x, p) = \begin{cases} 1 & \text{if } p \text{ starts at } x \\ 0 & \text{otherwise} \end{cases} \quad \bar{\delta}(p, x) = \begin{cases} 1 & \text{if } p \text{ ends at } x \\ 0 & \text{otherwise} \end{cases}$$

Given a sequence of observations $g^{1:T} = g^1 \dots g^T$ and an associated trajectory $\tau = x^1 p^1 \dots x^T$, we define the *unnormalized score*, or *potential*, of the trajectory as:

$$\phi(\tau | g^{1:T}) = \left[\prod_{t=1}^{T-1} \omega(g^t | x^t) \underline{\delta}(x^t, p^t) \eta(p^t) \bar{\delta}(p^t, x^{t+1}) \right] \times \omega(g^T | x^T)$$

The non-negative function ϕ is called the *potential function*. When properly scaled, the potential ϕ defines a probability distribution over all possible trajectories, given a sequence of observations: $\pi(\tau | g^{1:T}) = Z^{-1} \phi(\tau | g^{1:T})$. The variable $Z = \sum_{\tau} \phi(\tau | g^{1:T})$, called the *partition function*, is the sum of the potentials over all the compatible trajectories.

The potential function ϕ defines an unnormalized distribution over all trajectories. Such a probabilistic framework is a *Conditional Random Field* (CRF) [9], for which efficient inference algorithms exist.

The Case against the Hidden Markov Model Approach. The classical approach to filtering in the context of trajectories is based on *Hidden Markov Models* (HMMs), or their generalization, *Dynamic Bayesian Networks* (DBNs) [10]: a sequence of states and trajectories form a trajectory,

and the coupling of trajectories and states is done using transition models $\hat{\pi}(x|p)$ and $\hat{\pi}(p|x)$. See Figure 4 for a representation.

This results in a chain-structured directed probabilistic graphical model in which the path variables p^t are unobserved. Depending on the specifics of the transition models, $\hat{\pi}(x|p)$ and $\hat{\pi}(p|x)$, probabilistic inference has been done with Kalman filters [11], the forward algorithm or the Viterbi algorithm [3], or particle filters [7].

Hidden Markov Model representations, however, suffer from the *selection bias problem*, first noted in the labeling of words sequences [9], which makes them not the best fit for solving path inference problems. Consider the example trajectory $\tau = x^1 p^1 x^2$ observed in our data,

represented in Figure 5. For clarity, we consider only two states x_1^1 and x_2^1 associated with the GPS reading g^1 and a single state x_1^2 associated with g^2 . The paths $(p_j^1)_j$ between x^1 and x^2 may either be the lone path p_1^1 from x_1^1 to x_1^2 that allows a vehicle to cross the Golden Gate Park, or one of the many paths between Cabrillo Street and Fulton Street that go from x_2^1 to x^1 , including p_3^1 and p_2^1 . In the HMM representation, the transition probabilities must sum to 1 when conditioned on a starting point. Since there is a single path from x_2^1 to x^2 , the probability of taking this path from the state x_1^1 will be $\hat{\pi}(p_1^1|x_1^1) = 1$ so the overall probability of this path is $\hat{\pi}(p_1^1|g^1) = \hat{\pi}(x_1^1|g^1)$. Consider now the paths from x_2^1 to x_1^2 : a lot of these paths will have a similar weight, since they correspond to different turns and across the lattice of streets. For each path p amongst these N paths of similar weight, Bayes' assumption implies $\hat{\pi}(p|x_2^1) \approx \frac{1}{N}$ so the overall probability of this path is $\hat{\pi}(p|g^1) \approx \frac{1}{N} \hat{\pi}(x_2^1|g^1)$. In this case, N can be large enough that $\hat{\pi}(p_1^1|g^1) \geq \hat{\pi}(p|g^1)$, and the remote path will be selected as the most likely path.

Due to their structures, all HMM models will be biased towards states that have the least expansions. In the case of a road network, this can be pathological; this phenomenon is observed around highways for example. Our model, which is based on CRFs, does not have this problem since the renormalization happens just once and is over all paths from start to end, rather than renormalizing for every single state transition independently.

Trajectory Filtering and Smoothing. Computing the most likely trajectory $\tau^* = \operatorname{argmax}_\tau \pi(\tau|g^{1:T})$ is a particular instantiation of a standard dynamic programming algorithm called the *Viterbi algorithm* [5]. The posterior probability \bar{q}_i^t of the vehicle being at the state $x_i^t \in \mathbf{x}^t$ at time t , given all the observations $g^{1:T}$ of the trajectory, is defined up to some scaling factor as:



Fig. 5 Example of a failure case when using a Hidden Markov Model: the solid black path will be favored over all the other paths

$$\bar{q}_i^t \propto \pi(x_i^t | g^{1:T}) = \frac{1}{Z} \sum_{\tau=x^1 \dots p^{t-1} x_i^t p^t \dots x^T} \phi(\tau | g^{1:T})$$

A natural choice is to scale the distribution \bar{q}_i^t so that the probabilistic weight of all possibilities is equal to 1: $\sum_{1 \leq i \leq I^t} \bar{q}_i^t = 1$. The quantity \bar{q}_i^t has a clear meaning: it is the probability that the vehicle is in state x_i^t , when choosing amongst the set $(x_{i'}^t)_{1 \leq i' \leq I^t}$, given all the observations $g^{1:T}$. For each time t and each path index $j \in [1 \dots J^t]$, we also introduce the discrete distribution over the paths at time t given the observations $g^{1:T}$: $\bar{r}_j^t \propto \pi(p_j^t | g^{1:T})$ which are scaled so that $\sum_{1 \leq j \leq J^t} \bar{r}_j^t = 1$.

Finding the most likely trajectory is one of two classical inference problems [10]. The other one is to find for each time what the probability distribution over states is. When doing so conditioned on all past and future observations (in our application the GPS measurements $g^{1:T}$), it is called smoothing, when doing so conditioned on all past measurements only it is called filtering. Filtering can be achieved by a forward pass, and smoothing can be achieved by both the same forward pass as done for filtering and a backward pass and then combining their results [14].

The forward-backward algorithm as applied to our CRF is presented in Algorithm 1. The path inference algorithm has time complexity $O(TUV)$ with U the maximum number of paths at one time step, and V the maximum number of paths originating from a single point (which is usually small). We refer the reader to the supplementary material for a more complete overview of the algorithm. Furthermore, the path inference algorithm can be adapted to pure filtering or lagged-smoothing scenarios to get timely estimates as new data comes in. We also provide a reference implementation of all variations [1].

Observation Model. We now describe the observation model ω . The observation probability is assumed to only depend on the distance between the point and the GPS coordinates. We take an isoradial Gaussian noise model $\omega(g|x) = \frac{1}{\sqrt{2\pi}\sigma} \left(-\frac{1}{2\sigma^2} d(g, x)^2 \right)$ in which the function d is the distance function between geocoordinates. In a first approximation, the standard deviation σ is assumed to be constant over all the network. This model works surprisingly well despite some known limitations such as urban canyoning. A more robust model, such as the exponential distribution, can also be used. It would be interesting to see how the choice of a more robust model affects the quality of the output.

Driver Model. The driver model assigns a weight to any path on the road network. We consider a model in the *exponential family*, in which the weight distribution over any path p only depends on a selected number of features $\varphi(p) \in \mathbb{R}^K$ of the path: $\eta(p) \propto \exp(\mu^T \varphi(p))$. Feature functions considered in the present article include the length of the path, the mean speed and travel time, the number of stop signs and signals, and the number of turns to

Algorithm 1. Description of the forward-backward recursion algorithm

Given a sequence of observations $g^{1:T}$, a sequence of sets of candidate projections $\mathbf{x}^{1:T}$ and a sequence of sets of candidate paths $\mathbf{p}^{1:T-1}$.

Initialize the forward state distribution:

$$\forall i = 1 \dots I^1: \bar{q}_i^1 \leftarrow \omega(x_i^1 | g^1)$$

For every time step t from 1 to $T - 1$:

Compute the forward probability over the paths:

$$\forall j = 1 \dots J^t: \bar{r}_j^t \leftarrow \eta(p_j^t) \left(\sum_{j: \delta(x_i^t, p_j^t)=1} \bar{q}_i^t \right)$$

Compute the forward probability over the states:

$$\forall i = 1 \dots I^{t+1}: \bar{q}_i^{t+1} \leftarrow \omega(x_i^{t+1} | g^{t+1}) \left(\sum_{j: \delta(p_j^t, x_i^{t+1})=1} \bar{r}_j^t \right)$$

Initialize the backward state distribution

$$\forall i = 1 \dots I^T: \bar{q}_i^T \leftarrow 1$$

For every time step t from $T - 1$ to 1:

Compute the forward probability over the paths:

$$\forall j = 1 \dots J^t: \bar{r}_j^t \leftarrow \eta(p_j^t) \left(\sum_{j: \delta(p_j^t, x_i^{t+1})=1} \bar{q}_i^{t+1} \right)$$

Compute the forward probability over the states:

$$\forall i = 1 \dots I^t: \bar{q}_i^t \leftarrow \omega(x_i^{t+1} | g^{t+1}) \left(\sum_{j: \delta(x_i^t, p_j^t)=1} \bar{r}_j^t \right)$$

For every time step t :

$$\forall j = 1 \dots J^t: \bar{r}_j^t \leftarrow \bar{r}_j^t \cdot \bar{r}_j^t$$

Normalize \bar{r}^t

$$\forall i = 1 \dots I^t: \bar{q}_i^t \leftarrow \bar{q}_i^t \cdot \bar{r}_i^t$$

Normalize \bar{q}^t

Return the set of vectors $(\bar{q}^t)_t$ and $(\bar{r}^t)_t$

the right or to the left. The distribution is parametrized by a vector $\mu \in \mathbb{R}^K$, called the *behavioral parameter vector*.

Training. The procedure detailed so far requires the calibration of the observation model and the path selection model by setting some values for the weight vector μ and the standard deviation σ . There is a striking similarity between the state variables $x^{1:T}$ and the path variables $p^{1:T}$ — especially between the forward and backward equations introduced in Algorithm 1. Consider $\epsilon = \sigma^{-2}$ and θ the stacked vector of the desired parameters $\theta = (\epsilon \mu^T)^T$. One then realizes that the potential ϕ defines an exponential family distribution over the variables $\mathbf{x}^{1:T}, \mathbf{p}^{1:T-1}$, and that the vector θ is the natural parameter of the distribution. It ensues that maximizing the likelihood of observations with respect to θ is a convex problem [17]. Furthermore, the elements of the objective function (gradient, Hessian) can be efficiently computed using dynamic programming. The complete derivations can be found in the supplementary materials, and our reference implementation [1] provides an implementation of the likelihood maximization procedure.

Usually, only the GPS observations $g^{1:T}$ are available; the choices of x_i^t and p_j^t are hidden. In this case, we estimate a good value of θ using the *Expectation Maximization* (EM) algorithm, in which we take the expectation over the each

possible choice x_i^t and p_j^t . This is done by computing the marginal conditional probabilities $\bar{q}_i^t = \pi(x_i^t | g^{1:T}; \theta)$ and $\bar{r}_i^t = \pi(p_j^t | g^{1:T}; \theta)$, using the forward-backward algorithm (Algorithm 1).

4 Results from Field Operational Test

We evaluate the path inference filter with two datasets collected from the same source (taxi cabs): a smaller set at high frequency, called “Dataset 1”, and a larger dataset sampled at 1 minute for which we do not know ground truth, called “Dataset 2”. For the collection of Dataset 1, ten San Francisco taxicabs were fit with high frequency GPS (1 second sampling rate) during a two-day experiment. Together, they collected about 200,000 measurement points.

The second dataset consists of one day of one-minute samples of 600 taxis from the same fleet, which represents 600,000 GPS points, collected the same month.

Experiment Design. The path inference filter was first run using hand-tuned parameters on all the samples to build a set of ground truth trajectories. The trajectories were then downsampled to different temporal resolutions (2 seconds to two minutes) to test the filter in different configurations. We performed cross-validation to test the impact of the sampling rate, the computing strategy (“online” or pure filtering, “1-lag” and “2-lag” for fixed-lagged smoothing, Viterbi and “offline” or smoothing). We also tested different models: some deterministic models (“Hard closest point” [6], “Closest point”, and “Shortest path” [18]) were selected as baselines. In addition, we ran two models from the exponential family:

- “Simple”: A simple model that considers two features that could probably have been tuned by hand with some effort (but were learned in our experiments): the length of the path and the distance of a point projection to its GPS coordinate.
- “Complex”: A more complex model with a more diverse set of ten features, which makes manual tuning impractical. In addition to the two features of the simple model, this model in particular includes the number of signals, turns and stop signs on the path, the class of the road and some average speed information.

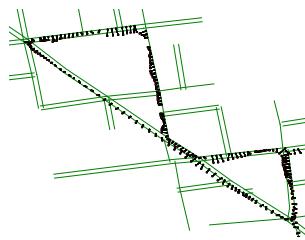


Fig. 6 Example of points collected in “Dataset 1”, in the Russian Hill neighborhood in San Francisco. The (red) dots are the GPS observations (collected every second), and the green lines are road links that contain a state projection. The black lines show the most likely projection of the GPS points on the road network.

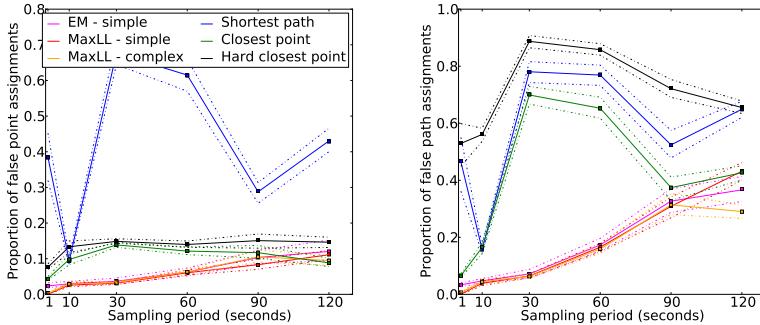


Fig. 7 Proportion of point misses (left) and path misses (right) using trajectory reconstruction (Viterbi algorithm) for different sampling rates, as a percentage of incorrect point reconstructions for each trajectory (positive, smaller is better). The solid line denotes the median, the squares denote the mean and the dashed lines denote the 95% confidence interval. The black curve is the performance of a greedy reconstruction algorithm, and the colored plots are the performances of probabilistic algorithms for different features and weights learned by different methods.

These two models were also trained in a supervised setting, leading to the “MaxLL-Simple” and “MaxLL-Complex” models, respectively. The simple model was trained using Expectation Maximization on “Dataset 1” and lead to “EM-Simple”. Due to convergence issues, the EM algorithm was used on the complex model using “Dataset 2”. This set of parameters is presented under the label “EM-Complex”. From a practical perspective, all the deterministic models can be well approximated using the Simple model by setting some large or small values to its parameters.

These models were evaluated under a number of metrics:

- The proportion of path and point misses: for each trajectory, it is the number of times the most likely path or the most likely state was not the true one, divided by the number of time steps in the trajectory.
- The log-likelihood of the true point projection and of the true path projection. This is indicative of how often the true point or the true path is identified by the model.
- The entropy of the path distribution and of the point distribution. This statistical measure indicates the confidence assigned by the filter to its result. A small entropy (close to 0) indicates that one path is strongly favored by the filter against all the other ones, whereas a large entropy indicates that all paths or points are considered equal.

Results. Given the number of parameters to adjust, we only present the most salient results here.

The raw accuracy of the filter, in terms of point misses and path misses, is presented in Figure 7. As expected, the error rate is 0 for high frequencies (low sampling period): all the points are correctly identified by all the algorithms.

In the low frequencies (high sampling periods), the error is still low (around 10%) for the trained models, and also for the greedy model (“Hard closest point”). For sampling rates between 10 seconds and 90 seconds, trained models (“Simple” and “Complex”) show a much higher performance compared to untrained models (“Hard closest point”, “Closest point” and “Shortest path”). In higher sampling regions, there are significantly more paths to consider and the error increases substantially. Nevertheless, the probabilistic models still perform very well: even at 2 minute intervals, they are able to recover about 75% of the true paths.

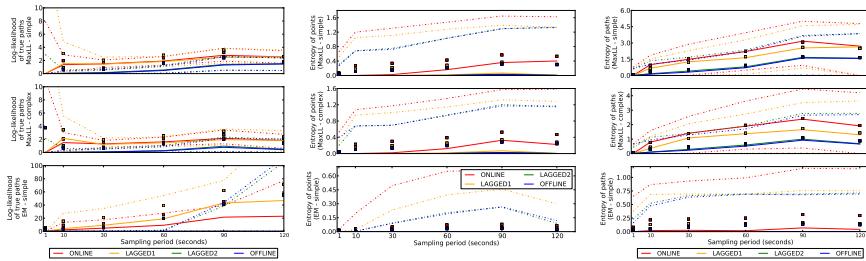


Fig. 8 On the left, negative of the log likelihood of true paths for different strategies and different sampling rates (positive, lower is better). On the center, entropy of point distributions and on the right, entropy of path distributions. The solid line denotes the median, the squares denote the mean and the dashed lines denote the 90% confidence interval.

We now turn our attention to the resilience of the models, i.e. how they perform when they make mistakes. We use two statistical measures: the (log) likelihood of the true paths, and the entropy of the distribution of points or paths (Figure 8). Note that in a perfect reconstruction with no ambiguity, the log likelihood would be zero. Interestingly, the log likelihoods appear very stable as the sampling interval grows: our algorithm will continue to assign high probabilities to the true projections even when many more paths can be used to travel from one point to the other. The performance of the simple and the complex models improves greatly when some backward filtering steps are used, and stays relatively even across different time intervals.

The paths inferred by the filter are also never dramatically different: at two minute time intervals (for which the paths are 1.7km on average), the returned path spans more than 80% of the true path on average. Using the complex model decreases this relative error by more than 15%. We refer the reader to the supplementary materials for a longer discussion on the performance of the filter.

In the case of the complex model, the weights can provide some insight into the features involved in the decision-making process of the driver. In particular, for extended sampling rates ($t=120s$), some interesting patterns appear. For example, drivers show a clear preference to turn on the right as

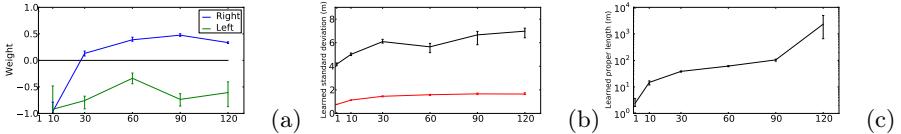


Fig. 9 Examples of learned feature weights: right turn preferences (a), standard deviation (b), and characteristic length (c). The error bars indicate the complete span of values computed for each time.

opposed to the left, as seen in Figure 9. This may be attributed in part to the difficulty in crossing an intersection in the United States.

Unsupervised Learning Results. The filter was also trained for the simple and complex models using Dataset 2. This dataset does not include true observations but is two orders of magnitude larger than Dataset 1 for the matching sampling period (1 minute). The unsupervised training finds some weight values similar to those found with supervised learning. The magnitude of these weights is larger than in the supervised settings, however. Indeed, during the E step, the algorithm is free to assign any sensible value to the choice of the path. This may lead to a self-reinforcing behavior and the exploration of a bad local minimum.

As Figure 10 shows, however, a large training dataset puts unsupervised methods on par with supervised methods as far as performance metrics are concerned.

| | False points | False paths |
|--------------------|-----------------|-----------------|
| EM-Complex (large) | 5.8±0.4% | 16.0±1.0% |
| EM-Simple (large) | 6.3±0.4% | 17.1±0.9% |
| EM-Simple | 6.2±0.6% | 17.3±1.3% |
| MaxLL-Complex | 6.2±0.3% | 15.8±0.7 |
| MaxLL-Simple | 6.1±0.3% | 16.5±0.7 |

Fig. 10 Proportion of true points and true paths incorrectly identified, for different models evaluated with 1-minute sampling (lower is better)

Key Findings. Our algorithm can reconstruct a sensible approximation of the trajectory followed by the vehicles analyzed, even in complex urban environments. In particular:

- An intuitive deterministic heuristic (“Hard closest point”) dramatically fails for paths at low frequencies, less so for points. It should not be considered for sampling intervals larger than 30 seconds.
- A simple probabilistic heuristics (“closest point”) gives good results for either very low frequencies (2 minutes) or very high frequencies (a few seconds) with more 75% of paths and 94% points correctly identified. However, the incorrect values are not as close to the true trajectory as they are with more accurate models (“Simple” and “Complex”).

- For the medium range (10 seconds to 90 seconds), trained models (either supervised or unsupervised) have a greatly improved accuracy compared to untrained models, with 80% to 95% of the paths correctly identified by the former.
- For the paths that are incorrectly identified, trained models (“Simple” or “Complex”) provide better results compared to untrained models (the output paths are closer to the true paths, and the uncertainty about which paths may have been taken is much reduced). Furthermore, using a complex model (“Complex”) improves these results even more by a factor of 13-20% on all metrics.
- For filtering strategies: online filtering gives the worst results and its performance is very similar to 1-lagged smoothing. The slower strategies (2-lagged smoothing and offline) outperform the other two by far. Two-lagged smoothing is nearly as good as offline smoothing, except in very high frequencies (less than 2 second sampling) for which smoothing clearly provides better results. We thus recommend two-lagged smoothing for most applications.
- Using a trained algorithm in a purely unsupervised fashion provides an accuracy as good as when training in a supervised setting - assuming enough data is available. The models produced by EM are equally good in terms of raw performance (path and point misses), but they may be overconfident.

5 Conclusions

We have presented a novel class of algorithms to track moving vehicles on a road network: the *path inference filter*. This algorithm first projects the raw points onto candidate projections on the road network and then builds candidate trajectories to link these candidate points. An observation model and a driver model are then combined in a Conditional Random Field to infer the most probable trajectories.

The algorithm exhibits robustness to noise as well as to the specificities of driving in urban road networks. It is competitive over a wide range of sampling rates (1 second to 2 minutes) and greatly outperforms intuitive deterministic algorithms. Furthermore, given a set of ground truth data, the filter can be automatically tuned using a fast supervised learning procedure. Alternatively, using enough GPS data with no ground truth, it can be trained using unsupervised learning. Experimental results show that the unsupervised learning procedure compares favorably against learning from ground truth data.

This algorithm supports a range of trade-offs between accuracy, timeliness and computing needs. It extends the current state of the art [20, 19] in its most accurate setting. The results are supported by the theoretical foundations of Conditional Random Fields. Because no standardized benchmark exists,

the authors have released an open-source implementation of the filter to foster comparison with other methodologies using other datasets [1].

The authors have written an industrial-strength version in the Scala programming language, deployed in the *Mobile Millennium* system. This multi-core version maps GPS points at a rate of several thousands of GPS points per second for the San Francisco Bay area and other large urban areas.

References

1. Supporting code for the path inference filter,
<https://github.com/tjhunter/Path-Inference-Filter/>
2. Bayen, A., Butler, J., Patire, A., et al.: Mobile Millennium final report. Technical report, University of California, Berkeley, CCIT Research Report UCB-ITS-CWP-2011-6 (2011)
3. Bierlaire, M., Flötteröd, G.: Probabilistic multi-modal map matching with rich smartphone data. In: STRC 2011 (2011)
4. The Cabspotting program, <http://cabspotting.org/>
5. Forney Jr., G.D.: The Viterbi algorithm. Proceedings of the IEEE 61(3), 268–278 (1973)
6. Greenfeld, J.S.: Matching GPS observations to locations on a digital map. In: 81th Annual Meeting of the Transportation Research Board (2002)
7. Gustafsson, F., Gunnarsson, F., Bergman, N., Forssell, U., Jansson, J., Karlsson, R., Nordlund, P.J.: Particle filters for positioning, navigation, and tracking. IEEE Transactions on Signal Processing 50(2), 425–437 (2002)
8. Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE transactions on Systems Science and Cybernetics 4(2), 100–107 (1968)
9. Lafferty, J.D., McCallum, A., Pereira, F.C.N.: Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In: Proceedings of the Eighteenth International Conference on Machine Learning (ICML 2001), pp. 282–289. Morgan Kaufmann Publishers Inc., San Francisco (2001)
10. Murphy, K.P.: Dynamic Bayesian Networks: Representation, Inference and Learning. PhD thesis, University of California at Berkeley (1994)
11. Ochieng, W.Y., Quddus, M., Noland, R.B.: Map-matching in complex urban road networks. Revista Brasileira de Cartografia 55(2) (2009)
12. Quddus, M.A., Ochieng, W.Y., Zhao, L., Noland, R.B.: A general map matching algorithm for transport telematics applications. GPS Solutions 7(3), 157–167 (2003)
13. Schrank, D.L., Lomax, T.J., and Texas Transportation Institute: 2009 Urban mobility report. The Texas A&M University (2009)
14. Seymore, K., McCallum, A., Rosenfeld, R.: Learning hidden Markov model structure for information extraction. In: AAAI-1999 Workshop on Machine Learning for Information Extraction, pp. 37–42 (1999)
15. Thiagarajan, A., Ravindranath, L.S., LaCurts, K., Toledo, S., Eriksson, J., Madden, S., Balakrishnan, H.: Vtrack: Accurate, energy-aware traffic delay estimation using mobile phones. In: 7th ACM Conference on Embedded Networked Sensor Systems (SenSys), Berkeley, CA (November 2009)
16. Thrun, S.: Probabilistic robotics. Communications of the ACM 45(3), 52–57 (2002)

17. Wainwright, M.J., Jordan, M.I.: Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning* 1(1-2), 1–305 (2008)
18. White, C.E., Bernstein, D., Kornhauser, A.L.: Some map matching algorithms for personal navigation assistants. *Transportation Research Part C: Emerging Technologies* 8(1-6), 91–108 (2000)
19. Yuan, J., Zheng, Y., Zhang, C., Xie, X., Sun, G.Z.: An interactive-voting based map matching algorithm. In: 2010 Eleventh International Conference on Mobile Data Management (MDM), pp. 43–52. IEEE (2010)
20. Zheng, Y., Quddus, M.A.: Weight-based shortest-path aided map-matching algorithm for low-frequency positioning data. In: Transportation Research Board 90th Annual Meeting (2011)

Predicting Pedestrian Trajectories Using Velocity-Space Reasoning

Sujeong Kim, Stephen J. Guy, Wenxi Liu, Rynson W.H. Lau,
Ming C. Lin, and Dinesh Manocha

Abstract. We introduce a novel method to predict pedestrian trajectories using agent-based velocity-space reasoning for improved human-robot interaction. This formulation models the trajectory of each moving pedestrian in a robot's environment using velocity obstacles and learns the simulation parameters based on tracked data. The resulting motion model for each agent is computed using statistical inferencing techniques from noisy data. This includes the combination of Ensemble Kalman filters and maximum likelihood estimation algorithm to learn individual motion parameters at interactive rates. We highlight the performance of our motion model in real-world crowded scenarios. We also compare its performance with prior techniques and demonstrate improved accuracy in the predicted trajectories.

1 Introduction

Robots are increasingly used in everyday life. As more robots are introduced into human surroundings, it becomes imperative to develop reliable and safe human-robot interaction. The robots should be able to successfully navigate in a dynamic environment with multiple moving humans to reach their goal positions. In order to develop such systems, the robots should have a good understanding of its environment and the ability to sense, track, and predict the position of all moving humans in its workspace to perform collision-free navigation.

Sensing and tracking moving humans is a topic that has been studied in both robotics and computer vision, e.g. [18, 25, 15]. A key part of these methods is often to use a prior motion model fitted for the scenario in question. However, these

Sujeong Kim · Stephen J. Guy · Ming C. Lin · Dinesh Manocha
University of North Carolina at Chapel Hill,
e-mail: {sujeong, sjguy, lin, dm}@cs.unc.edu

Wenxi Liu · Rynson W.H. Lau
City University of Hong Kong,
e-mail: wenxiliu2@student.cityu.edu.hk, rynson.lau@cityu.edu.hk

motion priors are usually generalized motions and not specific to individual's movement, therefore they do not perform well in tracking unusual (abnormal) behaviors or in capturing person-to-person differences. These behaviors, such as some people moving against the flow or quick movements to avoid collisions with other people, are common even in moderately crowded scenarios.

In this work, we introduce a novel motion model built on agent-based, velocity-space reasoning, combined with Bayesian statistical inference to provide a per-person motion model for each individual in a robot's environment. Unlike previous methods that use general motion priors or simple motion models, our method is able to reproduce an individual's motions with its own characteristics, also capable of dynamically adjusting the motion model for each individual in the presence of sensor noise and model uncertainty.

More specifically our approach works as follows. We assume at any given time, a robot has past observations on each person (represented as an agent) in the environment and wants to make some predictions about motion for each agent in the next timestep. We apply Ensemble Kalman Filtering (EnKF) to estimate the parameters for the human motion model based on *reciprocal velocity obstacle* (RVO) [2, 29] in a crowded scene that optimally matches the data observed thus far. Then, using these RVO parameters adaptively computed, we can simulate the crowd motion to estimate the future trajectory of all individuals in the scene and estimate factors, such as their current goal. We demonstrate that our per-agent learning method generates more accurate motion predictions than previous models, especially for complex scenarios, such as those with occlusions, low-resolution sensors, and missing data.

The rest of our paper is organized as follows. Section 2 reviews related work. Section 3 gives a brief overview of the RVO-based, per-agent motion model for a crowd. Section 4 describes our approach on incorporating an adaptive machine-learning framework with RVO-based multi-agent simulation, and Section 5 presents some results on observed (video recorded) data.

2 Related Work

In this section, we give an overview of prior work on motion models in robotics and crowd simulation and their application to pedestrian or people tracking.

2.1 Motion Models

There is an extensive body of work in robotics, multi-agent systems, crowd simulation, and computer vision on modeling pedestrians' motion in crowded environments. Here we provide a brief review of some recent work in modeling pedestrians' motion. In the fields of pedestrian dynamics and crowd simulation, many models have been proposed [26, 21]. These works could be broadly classified into a few main categories: potential-based works which models virtual agents in crowd as particles with potentials and forces [12], boid-like approaches which create simple rules to steer agents [24], geometric model which computes collision-free velocity[3, 29],

and field-based works which generate fields based on continuum theory or fluid models [28]. Among these works, velocity-based motion models [13, 14, 3, 29, 23] have been successfully applied in the analysis and simulation of crowd behaviors, and been shown to have efficient implementations [11].

2.2 People Tracking with Motion Model

Most related works in tracking people include [8, 27, 7] that make simple assumptions on the pedestrian motion model, in order to improve the tracking quality. In recent years, better pedestrian motion models have been deployed in robotics and computer vision literature for tracking people. For example, long-term motion planning models have been proposed to combine with tracking. Bruce, et al. [5] and Gong et al. [9] estimate people's destinations and in turn predict the pedestrian motion along the path from the current position towards the estimated goal position. Ziebart et al. [30] considers conditioned action as well as the trajectory of people for the prediction. Liao et al. [17] extracts a Voronoi graph from environment and predicts people's motion along the edges following the topological shape of the environment. In contrast to these methods, people tracking techniques using short-term motion models have also been studied as well. Bennewitz et al. [1] apply Expectation Maximization clustering to learn typical motion patterns from trajectories of people and employs Hidden Markov Models to predict the motion of people for a mobile robot. Luber et al. [18] apply Helbing's social force model to track people based on a Kalman-filter based tracker. Mehran et al. [20] also apply social force model to detect people's abnormal behaviors from video. Pellegrini et al. [22] use an energy function to build up a goal-directed short-term collision avoidance motion model and Linear Trajectory Avoidance to improve their people tracking quality from video. [29].

3 Background and Overview

In this section, we briefly discuss the crowd simulation method we use, and provide overview of our method for training this motion model based on observations. We first provide some definitions of important terms used in this section.

A *pedestrian* is a human entity that shares an environment with the robot. We treat pedestrians as autonomous entities that seek to avoid collisions with each other. We denote n as the number of pedestrians in the environment. We assume a robot's *sensor* produces a noisy estimate of the position of pedestrians. Furthermore, we assume the amount of sensor noise is known. We also assume the robot has an estimate of the environment represented as a series of connected line segments.

3.1 RVO Crowd Simulations

As part of our approach, we need to use an underlying crowd simulation technique to model individual goals and the interactions between people. For this model,

we choose a velocity-space reasoning technique based on Reciprocal Velocity Obstacles (RVO) [29]. RVO-based collision avoidance has previously been shown to reproduce important pedestrian behaviors such as lane formation, speed-density dependencies, and variations in human motion styles [11, 10].

Our implementation of RVO is based on the publicly available RVO2-Library (<http://gamma.cs.unc.edu/RVO2>). This library implements an efficient variation of RVO that uses a set of linear collision avoidance constraints on an agents velocities known as *Optimal Reciprocal Collision Avoidance* (ORCA) constraints [29]. Each agent is assumed to have a position, velocity, and a preferred velocity. If an agent's preferred velocity is forbidden by the ORCA constraints, that agent chooses the closest velocity which is not forbidden. Formally:

$$\mathbf{v}^{new} = \underset{\mathbf{v} \notin ORCA}{\operatorname{argmin}} \|\mathbf{v} - \mathbf{v}_{pref}\|. \quad (1)$$

This process is illustrated in Fig. 1a.

An agent's position is updated by integration of the new velocity computed in Eqn. 1. An agent's preferred velocity is assumed to change slowly, and is modeled by a small amount of noise each timestep. More details of the mathematical formulation is given in Sec 4. A through derivation of the ORCA constraints are given in [29].

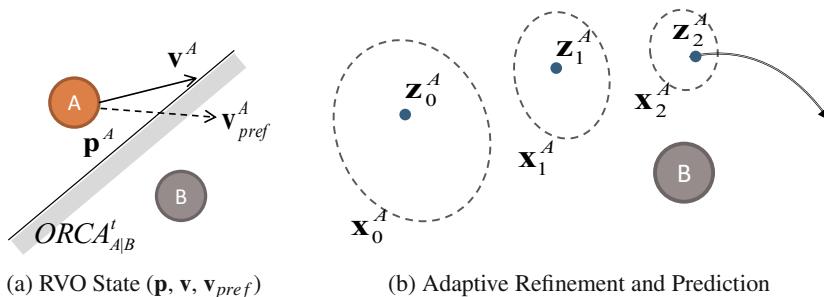


Fig. 1 (a) Overview of RVO Simulation illustrating an agent's position (\mathbf{p}), preferred velocity (\mathbf{v}_{pref}) and actual velocity (\mathbf{v}). If the ORCA collision-avoidance constraints prevent an agent from taking their preferred velocity, as shown here, the actual velocity will be the closest allowable velocity. These elements combine to form the RVO state \mathbf{x} . (b) As new data is observed (blue dot) BRVO refines its estimate of a distribution of likely values of the RVO states (dashed ellipse). These parameters are then used with RVO to predict likely paths (as indicated by arrow).

We use the RVO2-Library to represent the state of each sensed pedestrian in a robot's environment. The state of pedestrian at timestep k is represented as a 6D vector \mathbf{x}_k which consists of an agent's 2D position, 2D velocity, and 2D preferred

velocity as defined by RVO2-Library. Our human motion model, called *Bayesian-RVO* or *BRVO*, seeks to adaptively find the RVO state that best represents all the previously observed motion data (with sensing uncertainty) for each pedestrian in the robot's environment.

3.2 Problem Definition

As discussed above, we define the computation of the BRVO motion model as a state estimation problem. Formally, the problem we seek to solve is as follows. Given a set of observations, denoted $\mathbf{z}_0 \cdots \mathbf{z}_t$, for each pedestrian, what is the RVO state \mathbf{x}_t that best reproduces the motion seen so far. Given this estimate we predict the future motion of each agent (person) by using the RVO simulation model to determine the likely future path of each pedestrian.

We propose an iterative solution to this problem. We assume a robot working under a sense-plan-act loop. During the sensing step, the robot measures new (noisy) positions of each pedestrian, updates its estimate of each person's state, and creates a new plan taking into account the expected motion of nearby pedestrians.

4 Bayesian-RVO

In this section, we provide the mathematical formulation of our Bayesian-RVO motion model, or **BRVO**. This model combines an Ensemble Kalman Filtering approach with the Expectation-Maximization algorithm to estimate the best approximated RVO state of each agent, as well as the uncertainty in the model.

4.1 Model Overview

Our model performs Bayesian learning for each agent. We assume each agent can be represented with their own RVO state vector \mathbf{x} . Given an agent's RVO state \mathbf{x}_k at timestep k , we use the RVO collision-avoidance motion model, denoted here as f , to predict the agent's next state \mathbf{x}_{k+1} . We denote the error f has in predicting the RVO state at each timestep as \mathbf{q} . This leads to our motion model of:

$$\mathbf{x}_{k+1} = f(\mathbf{x}_k) + \mathbf{q} \quad (2)$$

Additionally, we assume the sensing of the robot can be represented by a function h that projects the predicted state \mathbf{x}_k to an observed state which we will denote as \mathbf{z}_k . In general, sensing devices produce noisy readings leading to an error between the observed state and the ground truth, we denote this error as \mathbf{r} . This assumption leads to our sensing model as:

$$\mathbf{z}_k = h(\mathbf{x}_k) + \mathbf{r} \quad (3)$$

An overview of this adaptive motion prediction model is given in Figure 2.

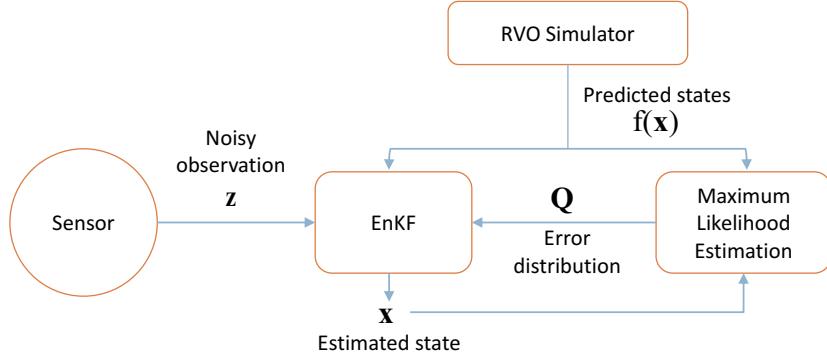


Fig. 2 Overview of the Adaptive Motion Model. We estimate current state \mathbf{x} via an iterative process. Given noisy data observed by the sensor, RVO as a motion model, and the error distribution matrix \mathbf{Q} , we estimate current state. The error distribution matrix \mathbf{Q} is recomputed based on the difference between the observed state and the prediction $f(\mathbf{x})$ and used to refine current estimation of \mathbf{x} .

Simplifying Assumptions The model given by Eqns. 2 and 3 is very general. In order to efficiently estimate the state \mathbf{x}_k from the observations \mathbf{z}_k we must make some simplifying assumptions, which allow us to develop a suitable learning approach for our adaptive model. First we assume that the error terms q and r are independent at each timestep, and follow a zero-meaned Gaussian distribution with covariances Q and R respectively. That is:

$$\mathbf{q} \sim N(0, Q) \quad (4)$$

$$\mathbf{r} \sim N(0, R) \quad (5)$$

We make a further assumption that the sensor error r is known or can be well estimated. This is typically possible by making repeated measurements of known data points to establish an average error. In many cases, this error will be provided by the manufacturer of the sensing device. This error will fully specify the matrix R .

To summarize, the h function is specified by the robot's sensors, and the matrix R characterizes the estimated accuracy of these sensors. The f function is a motion model will be used to predict the motion of each agent and Q is the accuracy of this model.

Our BRVO framework uses the RVO-based simulation model to represent the function f and Ensemble Kalman Filtering to estimate the simulation parameters which best fit the observed data. In addition, we adapt the EM-algorithm to estimate the model error Q for each agent. Better estimating Q improves the Kalman Filtering process, which in turn improves the predictions given by BRVO. This process is used iteratively to predict the next state and refine the state estimation for each agent, as depicted in Fig 1b. More specifically, we perform EnKF and EM step for each

agent, separately, but taking account all the agents in the motion model $f(x)$. It gives more flexibility in cases like dynamically changing scenes, such as agents entering and leaving the scene in the middle of the sequences, because the computations are done with fixed size per-agent matrix.

4.2 State Estimation

Optimal estimation of the simulation state \mathbf{x}_k is possible when f and h are linear functions by using Kalman Filtering. However, our BRVO approach uses RVO for the simulation model f and allows an arbitrary sensing model h which creates a non-linear system with no known method of finding an optimal estimate.

The BRVO motion model uses an extension to Kalman Filtering known as Ensemble Kalman Filter (EnKF) as a model for state estimation. The motivation for this algorithmic choice is two-fold. First, EnKF makes the same assumptions we laid forth in Sec 4.1. That is, a (potentially) non-linear f and h combined with a Gaussian representation of error. Secondly, as compared to methods such as particle filters, EnKF is very computationally efficient, providing more accuracy for a given number of samples. This is an important consideration for low-to-medium power onboard computers commonly found on a mobile robot.

At a high level, EnKF works by representing the potential state of an agent at each timestep as an ensemble or collection of discrete samples. Each sample is updated according to the motion model f . A mean and standard deviation of the samples is computed at every timestep for use in the predictor-correction equations. For a more detailed explanation of EnKF, we refer readers to a standard textbook in statistical inference such as [6].

State Representation. We represent the state of each agent as the set of RVO parameters as discussed Sec 3.1. Therefore the state of an agent, \mathbf{x} , is six dimensional:

$$\mathbf{x} = \begin{bmatrix} \mathbf{p} \\ \mathbf{v} \\ \mathbf{v}_{pref} \end{bmatrix}, \quad (6)$$

where \mathbf{p} is the agent's position, \mathbf{v} the velocity, and \mathbf{v}_{pref} the preferred velocity. To summarize the crowd dynamics model f :

$$f\left(\begin{bmatrix} \mathbf{p} \\ \mathbf{v} \\ \mathbf{v}_{pref} \end{bmatrix}\right) = \begin{bmatrix} \mathbf{p} + \mathbf{v}\Delta t \\ \operatorname{argmin}_{\mathbf{v} \in OCRA} \|\mathbf{v} - \mathbf{v}_{pref}\| \\ \mathbf{v}_{pref} \end{bmatrix}. \quad (7)$$

For the experiments in this paper, we assume that the robot has the ability to sense the relative positions of pedestrians. This assumption leads to a simple h function of

$$h\left(\begin{bmatrix} \mathbf{p} \\ \mathbf{v} \\ \mathbf{v}_{pref} \end{bmatrix}\right) = \mathbf{p}. \quad (8)$$

In general, our BRVO framework makes no assumption about the linearity of the sensing model. More complex sensing functions can be represented (for example, integrating multiple sensors) by modifying the function h in accordance with the new sensing model.

4.3 Maximum Likelihood Estimation

The accuracy of the states estimated by the EnKF algorithm is a function of the parameters defined in Eqns 2-5: f , Q , h , and R . While the sensing function h and the error distribution R are determined by the sensor's specifications, f is determined by the motion model chosen. However, the choice of motion prediction error distribution Q is still a free parameter. We propose a method of estimating the optimal value for Q based on the Expectation Maximization or EM-algorithm [19].

The EM-algorithm is a generic framework for learning (locally) optimal parameters by following a gradient decent approach. The algorithm alternates between two steps: an E-step which computes expected values and an M-step which computes the parameters which maximize the likelihood of the values computed during the E-step.

In our case, the E-step is exactly the EnKF algorithm. This step estimates the most likely state given the parameters Q . For the M-step, we need to compute the Q which maximizes the likelihood of values estimated from EnKF. This probability will be maximized with a Q that best matches the observed error between the predicted state and the estimated state. We can compute this value simply by finding the average error for each sample in the ensemble at each timestep for each agent.

By iteratively performing the E and M steps we will continuously improve our estimate of Q which will in turn improve the quality of the learning and the predictiveness of the method. In theory, one should iterate over the E and M steps until convergence. In practice, the process converges fairly rapidly. Due to the online process nature of our approach, we limit the number of iterations to three, which we found to be empirically sufficient. We analyze the resulting improvement produced by the EM algorithm in Section 5.1.

4.4 Implementation

A pseudo code for our BRVO algorithm is given in Algorithm 15. We represent the distribution of likely RVO states as an ensemble of m samples. We set $m = 1000$, for the results shown in section 5. Lines 2 through 6 preform a stochastic prediction of the likely next state. Lines 7 through 10 correct this predicted value based on the observed data from the robot's sensor. Lines 11 through 14 preform a maximum likelihood estimation of the uncertainty in the prediction.

Algorithm 1. Bayesian-RVO

Input: Observed positions over time $\mathbf{z}_1 \dots \mathbf{z}_t$, crowd motion simulator f , estimated initial error variance \mathbf{Q} , sensor function h , sensor noise \mathbf{R} , and the number of samples m . **Output:** Estimated agent's state distributions $\mathbf{x}_1 \dots \mathbf{x}_t$

```

foreach  $k \in 1 \dots t$  do
    // EnKF Predict Step
    foreach  $i \in 1 \dots m$  do
        Draw  $\mathbf{q}_{k-1}^{(i)}$  from  $\mathbf{Q}$ ,  $\hat{\mathbf{x}}_k^{(i)} = f(\hat{\mathbf{x}}_{k-1}^{(i)}) + \mathbf{q}_{k-1}^{(i)}$ ;
        Draw  $\mathbf{r}_k^{(i)}$  from  $\mathbf{R}$ ,  $\hat{\mathbf{z}}_k^{(i)} = h(\hat{\mathbf{x}}_k^{(i)}) + \mathbf{r}_k^{(i)}$ ;
         $\bar{\mathbf{z}}_k = \frac{1}{m} \sum_{i=1}^m \hat{\mathbf{z}}_k^{(i)}$ ;
         $Z_k = \frac{1}{m} \sum_{i=1}^m (\hat{\mathbf{z}}_k^{(i)} - \bar{\mathbf{z}}_k)(\hat{\mathbf{z}}_k^{(i)} - \bar{\mathbf{z}}_k)^T$ ;
    // EnKF Correct Step
     $\bar{\mathbf{x}}_k = \frac{1}{m} \sum_{i=1}^m \hat{\mathbf{x}}_k^{(i)}$ ;
     $\Sigma_k = \frac{1}{m} \sum_{i=1}^m (\hat{\mathbf{x}}_k^{(i)} - \bar{\mathbf{x}}_k)(\hat{\mathbf{x}}_k^{(i)} - \bar{\mathbf{x}}_k)^T$ ;
    foreach  $i \in 1 \dots m$  do
         $\hat{\mathbf{x}}_k^{(i)} = \hat{\mathbf{x}}_k^{(i)} + \Sigma_k Z_k^{-1} (\mathbf{z}_k - \hat{\mathbf{z}}_k^{(i)})$ ;
    // Maximum Likelihood Estimation
     $\mathbf{Q}_{k-1} = \mathbf{Q}$ ;
    foreach  $i \in 1 \dots m$  do
         $\mathbf{Q}_k += (\hat{\mathbf{x}}_k^{(i)} - f(\hat{\mathbf{x}}_{k-1}^{(i)}))(\hat{\mathbf{x}}_k^{(i)} - f(\hat{\mathbf{x}}_{k-1}^{(i)}))^T$ ;
     $\mathbf{Q} = \frac{k-1}{k} \mathbf{Q}_{k-1} + \frac{1}{k} \mathbf{Q}_k$ 

```

5 Results

In this section, we show some comparisons and results that show the advantage of our BRVO model. We analyze various attributes of our model across a variety of datasets. First, we isolate the effect of the EM loop for estimating the prediction uncertainty \mathbf{Q} . Next, we demonstrate the ability of our approach to cope with sensor noise. Additionally, we analyze how varying density and varying sampling rates each affect the results. Finally, we provide a quantitative comparison to two recent techniques.

We focus our analysis on three different datasets, illustrated in Fig. 3.

Campus. Video data of students on campus recorded from the top of the ETH main building in Zurich was extracted by manual notation every 0.4 second [22]. We extract three sequences from this data, each containing about 10 seconds of pedestrian interaction: Campus-1 (7 pedestrians), Campus-2 (18 pedestrians), Campus-3 (11 pedestrians).

Bottleneck. Optical tracking equipment captured the motion of participant in a lab-environment [4]. Participants moved through a bottleneck opening into a narrow passage. Data was collect with a passage width of 1.0 meter and 2.5 meter, denoted

Bottleneck-100 and Bottleneck-250 respectively. Both studies have about 170 participants, and we use the data subsampled at a rate of 0.4 second.

Street. This video is recorded from a street view, of low density pedestrian traffic, with manual motion tracking [16]. The dataset contains motion of 148 pedestrians over a period of roughly 5 minutes.

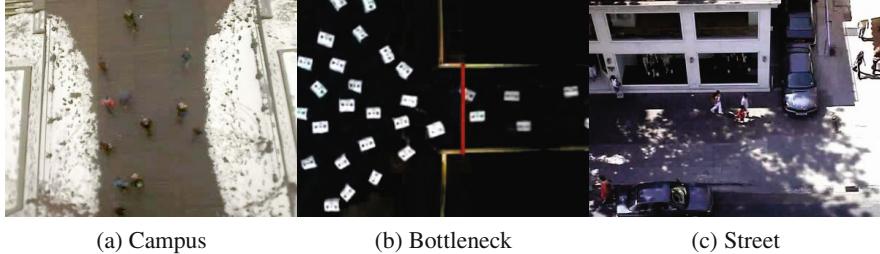


Fig. 3 Benchmarks used for our experiments (a) In the Campus dataset, a sparse set of students walk pass a fixed camera. (b) In the Bottleneck dataset, multiple cameras track participants walking into a narrow hallway. (c) In the Street dataset, a fixed camera tracks pedestrian motion on a street.

We include comparison with Constant Velocity and Constant Acceleration models as they are simple to implement, and provide a common benchmark to recent approaches with similar comparisons. For constant motion models, we used the last observed velocity and acceleration, respectively, for prediction. In other words, states are updated using the last observed velocity for Constant Velocity model, and the last observed acceleration for Constant Acceleration model.

5.1 Learning Maximum Likelihood (EM)

We show that our method can refine itself by using the EM algorithm. As discussed in Section 4.3, EM step reduces the difference between the observed state and the prediction based on our motion model. Figure 4a visually illustrate the effect of the EM algorithm; as new data is presented and pedestrians interact with each other, the estimated uncertainty, \mathbf{Q} , decreases. Without EM step, the same, initially given estimated uncertainty is used as \mathbf{Q} without refining its values.

We can analyze the effect of estimating \mathbf{Q} (as discussed in Section 4.3) by removing this step from our approach. We compare the performance of BRVO on the Campus sequences with and without this EM feedback loop.

Similar to the above experiments, BRVO learns for the first 5 frames, and predicts position of later frames. We test the improvement on varying level of noise. The same initial estimated error \mathbf{Q} is used for all cases. We use Gaussian noise with standard deviation 0.05m, 0.1m, 0.15m to the ground truth data. Figure 4b show

the measured improvement in term of reduced prediction error. As the chart shows, using the EM loop to estimate the uncertainty leads to more accurate predictions. We also observe that the improvement increases under larger amounts of noise.

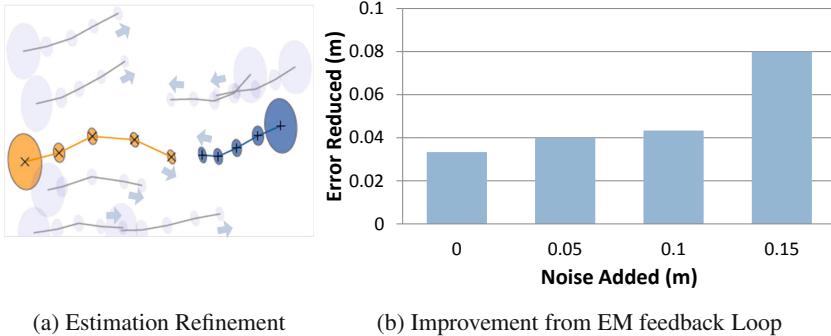


Fig. 4 The effect of EM algorithm. (a) This figure shows the trajectory of each agent and the estimated error distribution (ellipse) for the first five frames of Campus-3 data. The estimated error distributions gradually reduces as agents interact. (b) The improvement provided by the EM feedback loop for various amounts of noise. As the noise increases, this feedback becomes more important.

5.2 Noisy Data

Sensor noise is an important concern in many robotic systems. To analyze how BRVO responds to noise in the data, we compare its performance across varying levels of synthetic noise. Again, BRVO learns for first 5 frames, and predicts positions on later frames. Figure 5 shows average prediction error across all the Campus sequences for BRVO, Constant Acceleration, and Constant Velocity models. In all cases, adding noise increases the prediction error. However, BRVO is minimally imprecise compared to other methods.

Figure 6 shows the percentage of correctly predicted path within varying accuracy threshold. At an accuracy threshold of 0.5m, BRVO far outperforms ContAcc and ConstVel models (44% vs 8% and 11% respectively) even with little noise. With larger amounts of noise, these difference is magnified further.

5.3 Varying Density Scenario

We use the Bottleneck scenarios to analyze the effect of varying densities. This scenario shows a variety of densities: before the passage, there is a high density of crowd from people backing up and waiting, as people pass through the entrance, the density drops to a more medium level, only a few people at a time enter the hallway resulting in a low density. We compute the error of BRVO for each of these regions of the scenario, for both the 1m and 2.5m hallway.

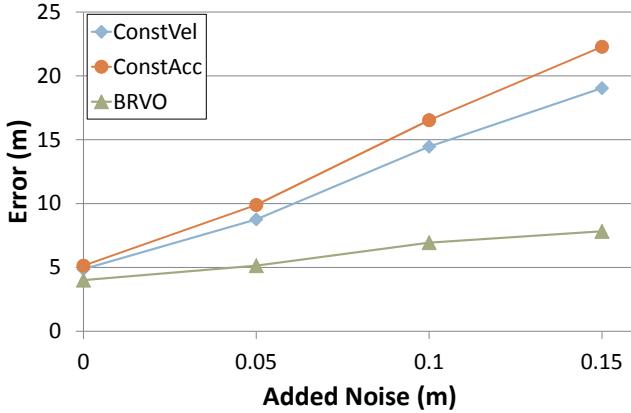


Fig. 5 Mean prediction error (lower is better). Prediction error after 7 frames (2.8s) on Campus dataset. As compared to constant velocity and constant acceleration models, BRVO can better cope with varying amount of noises in the data.

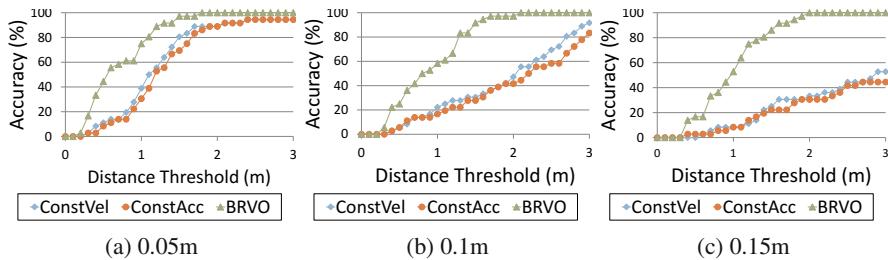


Fig. 6 Prediction Accuracy (higher is better) (a-c) Shows prediction accuracy across various accuracy thresholds. The analysis is repeated at three noise levels. For all accuracy thresholds, for all noise levels BRVO produces more accurate predictions than constant velocity or acceleration models. The advantage is most significant for large amounts of noise in the sensor data as in (c).

Figure 7 compares our error to Constant Velocity and Acceleration models. Both Constant Velocity and Constant Acceleration model have large variations in error for different density, but BRVO performs about equally well across all the densities as it dynamically adapts the parameters each frame.

5.4 Varying Sampling Rates

Our method also works well with very large timesteps. To demonstrate this aspect, we show the results on the Street dataset with varying sampling intervals to sub-sample the data. We chose Street scenario, the highest framerate at 0.04s per frame. Figure 8 shows the graph of the mean error versus sampling interval. From the result, we can

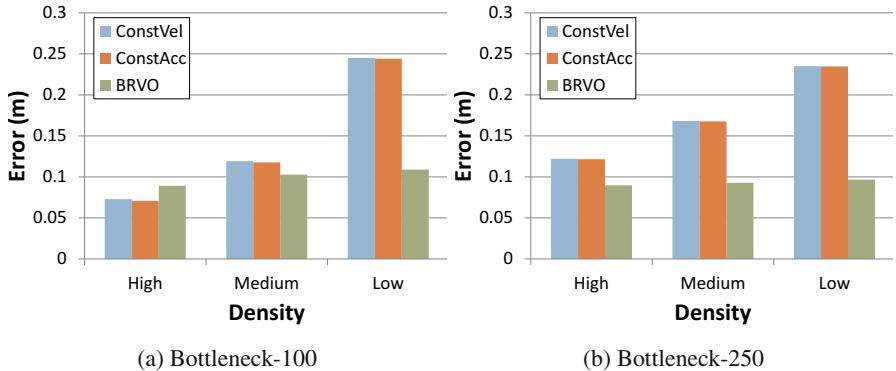


Fig. 7 Error at Various Densities (lower is better). In high-density scenarios there is little motion and simple models such as constant velocity perform about as well as BRVO. However, in low and medium densities where there is more motion BRVO provides more accurate motion prediction than the other models. In general, BRVO performs consistently well across various densities.

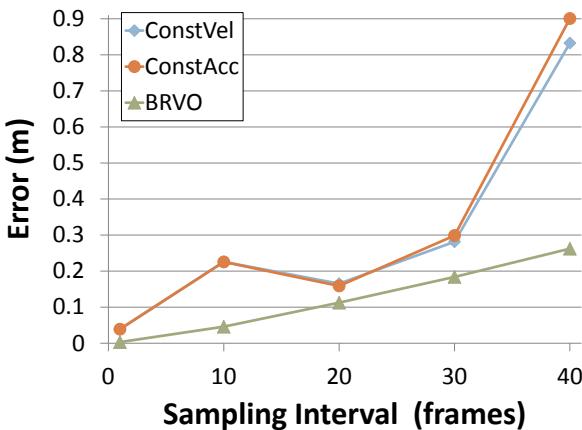


Fig. 8 Error vs Sampling Interval As the sampling interval increases the error of Constant Velocity and Constant Acceleration estimations grows much larger than that of BRVO.

see that our method performs very well compared to Constant Velocity and Constant Acceleration model in every sampling interval.

6 Conclusion

We have presented a method to predict pedestrian trajectories using agent-based velocity-space reasoning. The BRVO model we introduced is an online motion prediction method, which learns per-agent parameters even without prior knowledge

about the environment. We demonstrated the ability of this approach to perform well on several different datasets, with varying amounts of sensor noise, interaction levels, and density. Specifically, we showed our approach performs very well with noisy data and low framerate scenarios. We also showed that we can handle dynamic scenarios with temporal and spatial variance in density and speed. BRVO assumes no prior knowledge of the scenario to which the model is applied to. As such, it is well suited for a mobile robot that may frequently encounter new obstacles.

In the future, we would like to integrate our BRVO motion model for on-line/offline tracking. The improved motion model should increase tracking performance in typical environments but also in challenging scenarios, such as occluded scenes, low-resolution input, and data with missing frames, where the predictions from BRVO can help cope with these challenging issues. Additionally, we would like to incorporate our motion prediction model with an online planner for robots navigating in crowded scenes.

Acknowledgements. We are grateful to the reviewers for their comments, we would like to thank Jia Pan and David Wilkie for their help. This research is supported in part by ARO Contract W911NF-04-1-0088, NSF awards 0917040, 0904990, 100057 and 1117127, and Intel.

References

1. Bennewitz, M., Burgard, W., Cielniak, G., Thrun, S.: Learning motion patterns of people for compliant robot motion. *The International Journal of Robotics Research* 24(1), 31–48 (2005)
2. Van den Berg, J., Lin, M., Manocha, D.: Reciprocal velocity obstacles for real-time multi-agent navigation. In: IEEE International Conference on Robotics and Automation, ICRA 2008, pp. 1928–1935 (2008)
3. van den Berg, J., Patil, S., Sewall, J., Manocha, D., Lin, M.: Interactive navigation of individual agents in crowded environments. In: Symp. on Interactive 3D Graphics and Games (I3D 2008) (2008)
4. Boltes, M., Seyfried, A., Steffen, B., Schadschneider, A.: Automatic extraction of pedestrian trajectories from video recordings. *Pedestrian and Evacuation Dynamics* 2008, 43–54 (2010)
5. Bruce, A., Gordon, G.: Better motion prediction for people-tracking (2004)
6. Casella, G., Berger, R.: Statistical inference. Duxbury advanced series in statistics and decision sciences. Thomson Learning (2002)
7. Cui, J., Zha, H., Zhao, H., Shibusaki, R.: Tracking multiple people using laser and vision (2005)
8. Fod, A., Howard, A., Mataric, M.: A laser-based people tracker (2002)
9. Gong, H., Sim, J., Likhachev, M., Shi, J.: Multi-hypothesis motion planning for visual object tracking
10. Guy, S., Kim, S., Lin, M., Manocha, D.: Simulating heterogeneous crowd behaviors using personality trait theory. In: Symp. on Computer Animation, pp. 43–52 (2011)
11. Guy, S.J., Chhugani, J., Curtis, S., Dubey, P., Lin, M., Manocha, D.: PLEdestrians: a least-effort approach to crowd simulation. In: Symp. on Computer Animation, pp. 119–128 (2010)

12. Helbing, D., Molnar, P.: Social force model for pedestrian dynamics. *Physical Review E* 51(5), 4282 (1995)
13. Karamouzas, I., Heil, P., van Beek, P., Overmars, M.: A predictive collision avoidance model for pedestrian simulation. *Motion in Games*, 41–52 (2009)
14. Karamouzas, I., Overmars, M.: A velocity-based approach for simulating human collision avoidance (2010)
15. Kratz, L., Nishino, K.: Tracking pedestrians using local spatio-temporal motion patterns in extremely crowded scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (99), 1 (2011)
16. Lerner, A., Fitusi, E., Chrysanthou, Y., Cohen-Or, D.: Fitting behaviors to pedestrian simulations. In: Symp. on Computer Animation, pp. 199–208 (2009)
17. Liao, L., Fox, D., Hightower, J., Kautz, H., Schulz, D.: Voronoi tracking: Location estimation using sparse and noisy sensor data (2003)
18. Luber, M., Stork, J., Tipaldi, G., Arras, K.: People tracking with human motion predictions from social forces. In: 2010 IEEE International Conference on Robotics and Automation (ICRA), pp. 464–469 (2010)
19. McLachlan, G.J., Krishnan, T.: The EM Algorithm and Extensions, 2nd edn. Wiley Series in Probability and Statistics. Wiley-Interscience
20. Mehran, R., Oyama, A., Shah, M.: Abnormal crowd behavior detection using social force model. In: IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2009, pp. 935–942 (2009)
21. Pelechano, N., Allbeck, J., Badler, N.: Virtual crowds: Methods, simulation, and control. *Synthesis Lectures on Computer Graphics and Animation* 3(1), 1–176 (2008)
22. Pellegrini, S., Ess, A., Schindler, K., Van Gool, L.: You'll never walk alone: Modeling social behavior for multi-target tracking. In: 2009 IEEE 12th International Conference on Computer Vision, pp. 261–268 (2009)
23. Pettré, J., Ondřej, J., Olivier, A.H., Cretual, A., Donikian, S.: Experiment-based modeling, simulation and validation of interactions between virtual walkers. In: Symp. on Computer Animation, pp. 189–198 (2009)
24. Reynolds, C.W.: Steering behaviors for autonomous characters. In: Game Developers Conference (1999), <http://www.red3d.com/cwr/steer/gdc99>
25. Rodriguez, M., Ali, S., Kanade, T.: Tracking in unstructured crowded scenes. In: 2009 IEEE 12th International Conference on Computer Vision, pp. 1389–1396 (2009)
26. Schadschneider, A., Klingsch, W., Klüpfel, H., Kretz, T., Rogsch, C., Seyfried, A.: Evacuation dynamics: Empirical results, modeling and applications. Arxiv preprint arXiv:0802.1620 (2008)
27. Schulz, D., Burgard, W., Fox, D., Cremers, A.: People tracking with mobile robots using sample-based joint probabilistic data association filters. *The International Journal of Robotics Research* 22(2), 99 (2003)
28. Treuille, A., Cooper, S., Popović, Z.: Continuum crowds. *ACM SIGGRAPH 2006 Papers*, 1160–1168 (2006)
29. Van Den Berg, J., Guy, S., Lin, M., Manocha, D.: Reciprocal n-body collision avoidance. *Robotics Research*, 3–19 (2011)
30. Ziebart, B.D., Ratliff, N., Gallagher, G., Mertz, C., Peterson, K., Bagnell, J.A., Hebert, M., Dey, A.K., Srinivasa, S.: Planning-based prediction for pedestrians. In: Proceedings of the 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2009, pp. 3931–3936. IEEE Press (2009)

Erratum: Direct Trajectory Optimization of Rigid Body Dynamical Systems through Contact

Michael Posa and Russ Tedrake

Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, MA 02193
`{mposa, rust}@mit.edu`

E. Frazzoli et al. (Eds.): *Algorithmic Foundations of Robotics X*, STAR 86, pp. 527–542.
DOI: 10.1007/978-3-642-36279-8_32 © Springer-Verlag Berlin Heidelberg 2013

DOI 10.1007/978-3-642-36279-8_38

The Authors, Editors, and Publisher wish to include a reference to a body of work that was announced to us following publication of this article that independently produced similar results, and was in fact published before our manuscript (1-3). This work formulated nonlinear programs (NLP) in the mathematical programming with equilibrium constraints (MPEC) framework, and included formulations for autonomous and non-autonomous discontinuous transitions with or without unilateral constraints, for both elastic and inelastic collisions, and for time-stepping-based shooting methods.

These methods are the first we know of to remove the combinatorial complexity of hybrid mode scheduling and produce a polynomial time NLP. Our manuscript presents a different but similar framework, which differs by emphasizing sparseness in the program and solution techniques used by modern sequential quadratic programming solvers to scale the solution techniques to large systems (up to 22 discrete variables and over 4 million resulting modes).

1) Yunt, Kerim; Glocker, Christoph. Trajectory optimization of mechanical hybrid systems using SUMT. In: Advanced Motion Control, 2006. 9th IEEE International Workshop on. IEEE, 2005. S. 665–671.

2) Yunt, Kerim; Glocker, Christoph. A combined continuation and penalty method for the determination of optimal hybrid mechanical trajectories. In: IUTAM Symposium on Dynamics and Control of Nonlinear Systems with Uncertainty. Springer Netherlands, 2007. S. 187–196.

3) Yunt, Kerim; An Augmented Lagrangian-based Shooting Method for the Optimal Trajectory Generation of Switching Lagrangian Systems. *Dynamics of Continuous, Discrete and Impulsive Systems Series B: Applications & Algorithms* 18 (2011) 615–645, 2011 Watam Press.

The original online version for this chapter can be found at
http://dx.doi.org/978-3-642-36279-8_32

Author Index

- Abbeel, Pieter 591
Alamdari, Soroush 139
Amato, Nancy M. 297
Ames, Aaron D. 511
Arnold, Maxim 349
- Balkcom, Devin 331
Bandyopadhyay, Tirthankar 475
Baryshnikov, Yuliy 349
Bayen, Alexandre M. 591
Bekris, Kostas E. 279
Bhattacharya, Subhrajit 245
Bialkowski, Joshua 365
Bilstein, Siegfried 397
Bowman, Sean L. 559
Bretl, Timothy 71
- Chirikjian, Gregory S. 459
Chitta, Sachin 381
Chou, Michael 53
Corbett, Sean 53
Cornejo, Alejandro 397
- Denny, Jory 297
Disser, Yann 415
Dobson, Andrew 279
- Eagle, Emily 53
Erickson, Lawrence H. 427
- Fata, Elaheh 139
Frazzoli, Emilio 365, 475
Fudge, Elizabeth 397
- Geraerts, Roland 227
Ghrist, Robert 245
Glotzer, Dylan 53
Guy, Stephen J. 609
- Halperin, Dan 191, 313
Han, Li 53
Hauser, Kris 1
Hemmer, Michael 313
Hesch, Joel A. 559
Hsu, David 475
Huang, Yaonan 427
Hunter, Timothy 591
- Isler, Volkan 263
- Karaman, Sertac 365
Karamouzas, Ioannis 227
Khabbazian, Majid 397
Kim, Sujeong 609
Kottas, Dimitrios G. 559
Kramer, Jake 53
Krontiris, Athanasios 279
Kumar, Vijay 175, 209, 245
Kurniawati, Hanna 493
- Lau, Rynson W.H. 609
LaValle, Steven M. 123, 157, 349, 427
Lee, Wee Sun 475
Levihn, Martin 19
Lin, Ming C. 609
Lindsey, Quentin 209
Liu, Wenxi 609
Long, Andrew W. 459

- Lopez-Padilla, Rigoberto 123
Lynch, Andrew J. 397
- Majumdar, Anirudha 543
Manocha, Dinesh 381, 609
McCarthy, Zoe 71
McLurkin, James 397
Michael, Nathan 175
Mihalák, Matúš 415
Milenkovic, Victor 37
Moran, Jonathan 53
Murrieta-Cid, Rafael 123
- Noori, Narges 263
- Otte, Michael 365
- Pan, Jia 381
Pasupuleti, Murali 511
Patrikalakis, Nicholas M. 493
Perrin, Nicolas 89
Pietras, Christopher 53
Platt, Robert 443
Posa, Michael 527
- Roumeliotis, Stergios I. 559
Rudolph, Lee 53
Rus, Daniela 475
- Sacks, Elisha 37
Salzman, Oren 313
Scholz, Jonathan 19
Smith, Stephen L. 139
Solovey, Kiril 191
Stilman, Mike 19
- Tareen, Ammar 53
Tedrake, Russ 527, 543
Teichman, Alex 575
Thrun, Sebastian 575
Trac, Steven 37
Turpin, Matthew 175
- Valko, Matthew 53
van der Stappen, A. Frank 227
- Wang, Weifu 331
Widmayer, Peter 415
Wolfe, Kevin C. 459
Won, Kok Sung 475
- Yadukumar, Shishir Nadubettu 511
Yamane, Katsu 105
Yu, Jingjin 157, 427
- Zheng, Yu 105