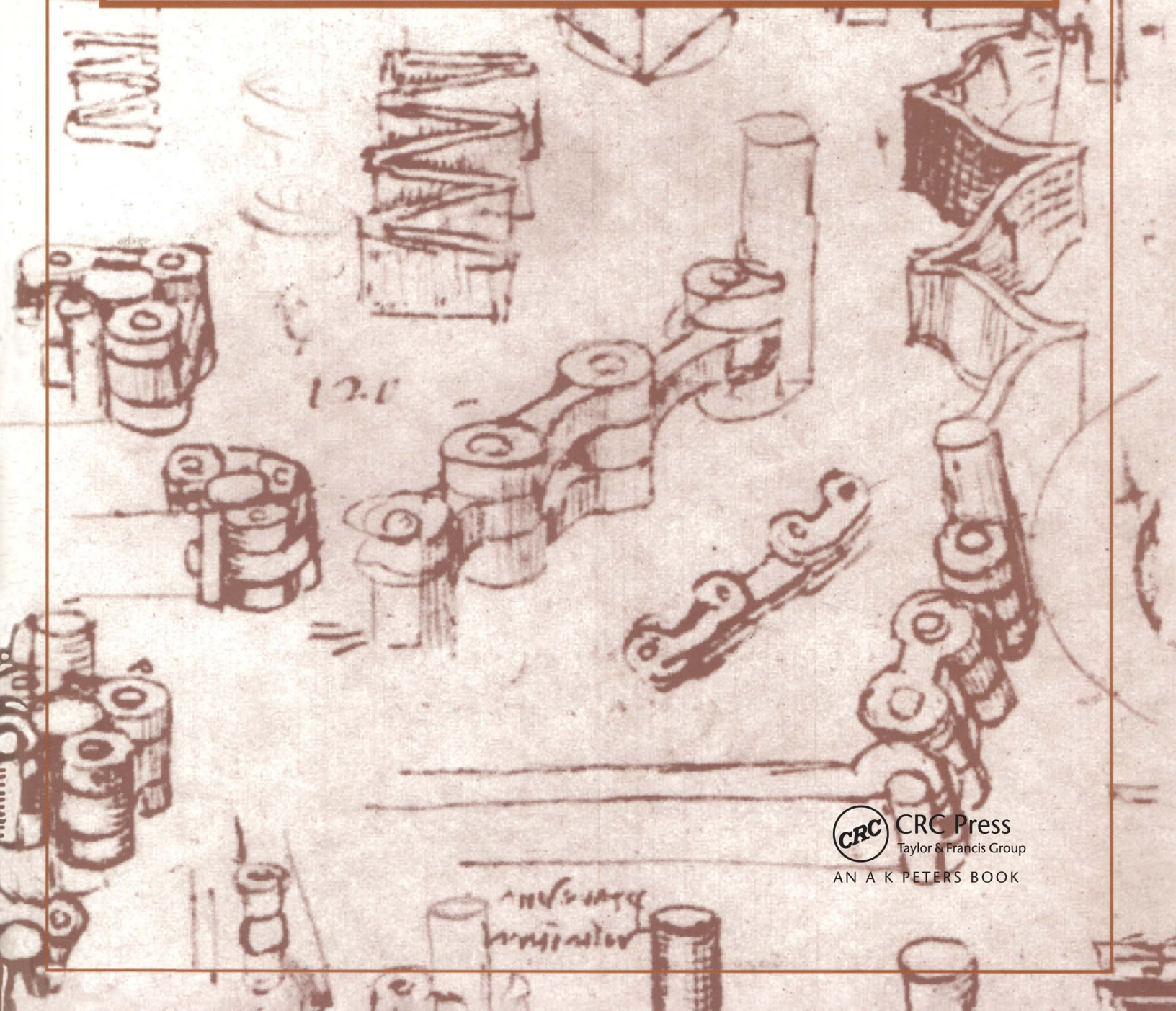
ROBOTICS

THE ALGORITHMIC PERSPECTIVE

◇ 1998 WAFR ◇

edited by PANKAJ K. AGARWAL, LYDIA E. KAVRAKI, *and* MATTHEW T. MASON



CRC Press
Taylor & Francis Group
AN A K PETERS BOOK

Robotics:
The Algorithmic Perspective

Taylor & Francis
Taylor & Francis Group
http://taylorandfrancis.com

Robotics: The Algorithmic Perspective

The Third Workshop on the Algorithmic Foundations of Robotics

edited by

Pankaj K. Agarwal
Duke University

Lydia E. Kavraki
Rice University

Matthew T. Mason
Carnegie Mellon University

CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business
AN A K PETERS BOOK

98
The Workshop on the Algorithmic Foundations of Robotics

First published 1998 by A. K. Peters, Ltd.

Published 2018 by CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

© 1998 by Taylor & Francis Group, LLC
CRC Press is an imprint of Taylor & Francis Group, an Informa business

No claim to original U.S. Government works

ISBN-13: 978-1-5688-1081-2 (hbk)

This book contains information obtained from authentic and highly regarded sources. Reasonable efforts have been made to publish reliable data and information, but the author and publisher cannot assume responsibility for the validity of all materials or the consequences of their use. The authors and publishers have attempted to trace the copyright holders of all material reproduced in this publication and apologize to copyright holders if permission to publish in this form has not been obtained. If any copyright material has not been acknowledged please write and let us know so we may rectify in any future reprint.

Except as permitted under U.S. Copyright Law, no part of this book may be reprinted, reproduced, transmitted, or utilized in any form by any electronic, mechanical, or other means, now known or hereafter invented, including photocopying, microfilming, and recording, or in any information storage or retrieval system, without written permission from the publishers.

For permission to photocopy or use material electronically from this work, please access www. copyright.com (http://www.copyright.com/) or contact the Copyright Clearance Center, Inc. (CCC), 222 Rosewood Drive, Danvers, MA 01923, 978-750-8400. CCC is a not-for-profit organization that provides licenses and registration for a variety of users. For organizations that have been granted a photocopy license by the CCC, a separate system of payment has been arranged.

Trademark Notice: Product or corporate names may be trademarks or registered trademarks, and are used only for identification and explanation without intent to infringe.

Visit the Taylor & Francis Web site at
http://www.taylorandfrancis.com

and the CRC Press Web site at
http://www.crcpress.com

Library of Congress Cataloging-in-Publication Data

Workshop on the Algorithmic Foundations of Robotics (3rd : 1998 : Houston, Tex.)
Robotics : the algorithmic perspective : 1998 Workshop on the Algorithmic Foundations of Robotics / edited by Pankaj K. Agarwal, Lydia E. Kavraki, Matthew T. Mason.
p. cm.
Includes bibliographical references and index.
ISBN 1-56881-081-4
1. Robotics—Congresses. 2. Algorithms—Congresses. I. Agarwal, Pankaj K. II. Kavraki, Lydia. III. Mason, Matthew T. IV. Title.
TJ210.3.W664 1998
629.8'92—dc21 98-20972
CIP

Cover illustration: Mechanical drawing of Leonardo da Vinci. From Leonardo da Vinci, Cod. Atl., fol. 357 r-a.Reynal and Company, New York, p. 498. Copyright in Italy by the Istituto Geograqfico De Agostini - Novara - 1956.

Contents

Foreword *vii*

Participants *ix*

Karl F. Böhringer and Bruce Randall Donald
Algorithmic MEMS 1

Dan S. Reznik and John F. Canny
Universal Part Manipulation in the Plane with a Single Horizontally-Vibrating Plate 21

Jonathan Luntz, William Messner, and Howie Choset
Velocity Field Design for the Modular Distributed Manipulator System (MDMS) 35

John Reif and Hongyan Wang
The Complexity of the Two Dimensional Curvature-Constrained Shortest-Path Problem 49

Jürgen Sellen
Approximation and Decision Algorithms for Curvature-Constrained Path Planning: A State-Space Approach 59

Hazem Jaouni, Maher Khatib, and Jean-Paul Laumond
Elastic Bands for Nonholonomic Car-Like Robots: Algorithms and Combinatorial Issues 69

Richard M. Murray, Joel W. Burdick, Scott D. Kelly, and James Radford
Trajectory Generation for Mechanical Systems with Application to Robotic Locomotion 81

Wesley H. Huang and Matthew T. Mason
Mechanics, Planning, and Control for Tapping 91

Moëz Cherif and Kamal K. Gupta
Planning for In-Hand Dextrous Manipulation 103

Michael Erdmann
Shape Recovery from Passive Locally Dense Tactile Data 119

John J. Craig
Geometric Algorithms in AdeptRAPID 133

David Hsu, Lydia E. Kavraki, Jean-Claude Latombe, Rajeev Motwani, and Stephen Sorkin
On Finding Narrow Passages with Probabilistic Roadmap Planners 141

Nancy M. Amato, O. Burchan Bayazit, Lucia K. Dale, Christopher Jones, and Daniel Vallejo
OBPRM: An Obstacle-Based PRM for 3D Workspaces 155

Joe Warren
Subdivision: A Technique for Modeling Curved Shape 169

Shankar Krishnan, Amol Pattekar, Ming C. Lin, and Dinesh Manocha
Spherical Shell: A Higher Order Bounding Volume for Fast Proximity Queries 177

Leonidas J. Guibas
Kinetic Data Structures: A State of the Art Report 191

Frank Hoffmann, Christian Icking, Rolf Klein, and Klaus Kriegel
The Polygon Exploration Problem: A New Strategy and a New Analysis Technique 211

John Reif and Zheng Sun
The Computational Power of Frictional Mechanical Systems 223

Kevin M. Lynch
Issues in Nonprehensile Manipulation 237

Yan-Bin Jia and Michael Erdmann
Local Observability of Rolling 251

Herbert Edelsbrunner
Geometry for Modeling Biomolecules 265

Robert-Paul Berretty, Ken Goldberg, Mark H. Overmars, and A. Frank van der Stappen
Algorithms for Fence Design 279

Jingliang Chen, Ken Goldberg, Mark H. Overmars, Dan Halperin, Karl F. Böhringer, and Yan Zhuang
Shape Tolerance in Feeding and Fixturing 297

Lydia E. Kavraki, Florent Lamiraux, and Christopher Holleman
Towards Planning for Elastic Objects 313

Anton Deguet, Ammar Joukhadar, and Christian Laugier
Models and Algorithms for the Collision of Rigid and Deformable Bodies 327

Steven M. LaValle
Numerical Computation of Optimal Navigation Functions on a Simplical Complex 339

Jocelyne Troccaz and Yves Delnondedieu
Synergistic Robots for Surgery: An Algorithmic View of the Approach 351

Alexander B. Kyatkin and Gregory S. Chirikjian
Fourier Methods on Groups: Applications in Robot Kinematics and Motion Planning 363

Keith Kotay, Daniela Rus, Marsette Vona, and Craig McGray
The Self-Reconfiguring Robotic Molecule: Design and Control Algorithms 375

Author Index 389

Foreword

Robotic technology is becoming almost commonplace as sensors, motors, and computers become smaller and cheaper. However, robotic technology depends not only on the physical hardware, but also on the software to perceive, plan, and control. Progress on the algorithmic foundations of robotics is critical.

The Third Workshop on the Algorithmic Foundations of Robotics was held at Rice University in Houston on March 5–7, 1998. As in the second workshop, the program consisted of both invited and contributing papers. The invited talks were given by John J. Craig (San Jose), Bruce R. Donald (Dartmouth), Herbert Edelsbrunner (Urbana–Champaign), Leonidas J. Guibas (Stanford), Richard M. Murray (Pasadena), Jocelyne Troccaz (Grenoble), and Joe Warren (Houston). The contributing papers were selected from 33 high-quality submissions by a program committee consisting of, in addition to the editors of this volume, John-Daniel Boissonnat (Sophia-Antipolis), Karl F. Böhringer (Seattle), John F. Canny (Berkeley), Gregory S. Chirikjian (Baltimore), Daniel Koditschek (Ann Arbor), Jean-Claude Latombe (Stanford), Jean-Paul Laumond (Toulouse), Rajeev Motwani (Stanford), Mark H. Overmars (Utrecht), Jean Ponce (Urbana–Champaign), Daniela Rus (Dartmouth), Russell Taylor (Baltimore), and Jeff Trinkle (College Station).

This volume contains the full versions of both the invited and contributing papers. They give a comprehensive overview of the state of the art in robot algorithms. The papers cover core problems in robotics, such as motion planning, sensor-based planning, manipulation, and assembly planning, and also examine the application of robotics algorithms in other domains, such as molecular modeling, geometric modeling, and computer-assisted surgery.

It was a pleasure to organize the workshop and to work with such a talented group of people. We thank all the authors who submitted papers to the workshop, the program committee members for their hard work in reviewing and selecting the papers, all the speakers for sharing their work with us, and all the participants for making the workshop so exciting. Finally, we thank the Computer Science Department and the Computer and Information Technology Institute at Rice University for their hospitality and support.

Pankaj K. Agarwal (Durham)
Lydia E. Kavraki (Houston)
Matthew T. Mason (Pittsburgh)

August, 1998

Taylor & Francis
Taylor & Francis Group
http://taylorandfrancis.com

Participants

Pankaj K. Agarwal	Duke University (pankaj@cs.duke.edu)
Juan-Manuel Ahuactzin	Universidad de las Americas, Puebla (jmal@mail.pue.udlap.mx)
Nancy M. Amato	Texas A&M University (amato@cs.tamu.edu)
Nina Amenta	University of Texas-Austin (amenta@cs.utexas.edu)
Boris Aronov	Polytechnic University (aronov@ziggy.poly.edu)
O. Burchan Bayazit	Texas A&M University (burchanb@cs.tamu.edu)
Robert-Paul Berretty	Utrecht University (robert-paul@sc.ruu.nl)
David Binger	Centre College (binger@centre.edu)
Karl F. Böhringer	University of California-Berkeley (karl@ieor.berkeley.edu)
Robert Burridge	Texas Robotics and Automation Center (burridge@hypercon.com)
John F. Canny	University of California-Berkeley (jfc@cs.berkeley.edu)
Jingliang Chen	University of California-Berkeley (jlchen@ieor.berkeley.edu)
Moëz Cherif	Simon Fraser University (cherif@cs.sfu.ca)
Gregory S. Chirikjian	Johns Hopkins University (gregc@jhu.edu)
Sunghee Choi	University of Texas-Austin (sunghee@cs.utexas.edu)
Howie Choset	Carnegie Mellon University (choset@cs.cmu.edu)
John J. Craig	Adept (jjc@silma.com)
Lucia K. Dale	Texas A&M University (dalel@cs.tamu.edu)
Bruce R. Donald	Dartmouth College (brd@cs.dartmouth.edu)
Herbert Edelsbrunner	University of Illinois at Urbana-Champaign (edels@cs.uiuc.edu)
Michael Erdmann	Carnegie Mellon University (me@cs.cmu.edu)
Fathi Ghorbel	Rice University (ghorbel@rice.edu)
Ken Goldberg	University of California-Berkeley (goldberg@ieor.berkeley.edu)
Robert W. Ghrist	University of Texas-Austin (ghrist@fireant.ma.utexas.edu)
Leonidas J. Guibas	Stanford University (guibas@cs.stanford.edu)
Kamal K. Gupta	Simon Fraser University (kamal@cs.sfu.ca)
Dan Halperin	Tel Aviv University (halperin@math.tau.ac.il)
Hirohisa Hirukawa	Electrotechnical Laboratory, AIST, MITI (hirukawa@etl.go.jp)
Frank Hoffman	Freie Universitat Berlin (hoffmann@inf.fu-berlin.de)
Christopher Holleman	Rice University (rusty@cs.rice.edu)
David Hsu	Stanford University (dyhsu@cs.stanford.edu)
Wesley H. Huang	Carnegie Mellon University (mongoose@cs.cmu.edu)
Christian Icking	Fern Universitat Hagen (icking@feu.edu)
James Jennings	Tulane University (jennings@eecs.tulane.edu)
Yan-Bin Jia	Carnegie Mellon University (jia@cs.cmu.edu)
Pernilla Johansson	Prosolvia Systems (pernilla@prosolvia.se)
Ioannis Kakadiaris	University of Houston (ioannisk@uh.edu)
Lydia E. Kavraki	Rice University (kavraki@cs.rice.edu)
Oussama Khatib	Stanford University (khatib@cs.stanford.edu)
Florent Lamiraux	Rice University (lamiraux@cs.rice.edu)
Jean-Claude Latombe	Stanford University (latombe@cs.stanford.edu)

Christian Laugier	INRIA-Alpes (christian.laugier@inriaples.fr)
Jean-Paul Laumond	LAAS-CNRS (jpl@laas.fr)
Steven M. LaValle	Iowa State University (lavalle@cs.iastate.edu)
Ming C. Lin	UNC-Chapel Hill (lin@cs.unc.edu)
Alex Lopez-Ortiz	University of New Brunswick (alopez-o@unb.ca)
Jonathan Luntz	Carnegie Mellon University (jl9f+@andrew.cmu.edu)
Kevin M. Lynch	Northwestern University (kmlynch@nwu.edu)
Dinesh Manocha	UNC-Chapel Hill (dm@cs.unc.edu)
Matthew T. Mason	Carnegie Mellon University (matt.mason@cs.cmu.edu)
Craig McGray	Dartmouth College (craig@cs.dartmouth.edu)
William Messner	Carnegie Mellon University (bmessner@andrew.cmu.edu)
Richard M. Murray	California Institute of Technology (murray@indra.caltech.edu)
Mark H. Overmars	Utrecht University (markov@cs.ruu.nl)
Jean Ponce	University of Illinois at Urbana-Champaign (ponce@cs.uiuc.edu)
John Reif	Duke University (reif@cs.duke.edu)
Dan S. Reznik	University of California-Berkeley (dreznik@cs.berkeley.edu)
Daniela Rus	Dartmouth College (rus@cs.dartmouth.edu)
Jürgen Sellen	ESG GmbH (jsellen@esg-gmbh.de)
Wookho Son	Texas A&M University (wooks@cs.tamu.edu)
Jeff Trinkle	Texas A&M University (trink@cs.tamu.edu)
Jocelyne Troccaz	TIMC-Faculte de Medicine (jocelyne.troccaz@imag.fr)
Daniel Vallejo	Texas A&M University (dvallejo@cs.tamu.edu)
A. Frank van der Stappen	Utrecht University (frankst@cs.ruu.nl)
Joe Warren	Rice University (jwarren@rice.edu)
Patrick Xavier	Sandia National Laboratories (pgxavie@isrc.sandia.gov)

Algorithmic MEMS

Karl Friedrich Böhringer, *University of California at Berkeley, Berkeley, CA, USA*
Bruce Randall Donald, *Dartmouth College, Hanover, NH, USA*

As improvements in fabrication technology for MEMS (microelectromechanical systems) increase the availability and diversity of these micromachines, engineers are defining a growing number of tasks to which they can be put. The idea of carrying out tasks using large coordinated systems of MEMS units motivates the development of automated, algorithmic methods for designing and controlling these groups of devices. We report here on progress towards algorithmic MEMS, *taking on the challenge of design, control, and programming of massively-parallel arrays of microactuators.*

We report on these developments in this focused survey paper, based on the research results originally reported in our 1994 paper [24] and developed further in [19, 20, 26, 21, 17, 18, 24, 12, 14, 15, 9]. We describe how arrays of MEMS devices can move and position tiny parts, such as integrated circuit chips, in flexible and predictable ways by oscillatory or ciliary action. The theory of programmable force fields can model this action, leading to algorithms for various types of micromanipulation that require no sensing of where the part is. Experiments support the theory. We also report on how the theory and results can be generalized to the macroscopic scale [11, 10].

1 Introduction

The recent, rapid innovations and improvements in microfabrication technology have resulted in an ever-growing complexity of available MEMS devices, mandating the use of automated, algorithmic methods for their design and control. This paper surveys our progress towards *algorithmic MEMS*, by taking on the challenge of design and control of massively-parallel micro actuator arrays. Our goal is to implement task-level, sensorless manipulation strategies with arrays of microfabricated actuators. The *theory of programmable force fields*[1] [19] arguably represents the first systematic attack on massively-parallel, distributed manipulation based on geometric and physical reasoning. Applications such as parts-feeding can be formulated in terms of the force fields required. Hence, programmable force fields act as an *abstraction bar-*

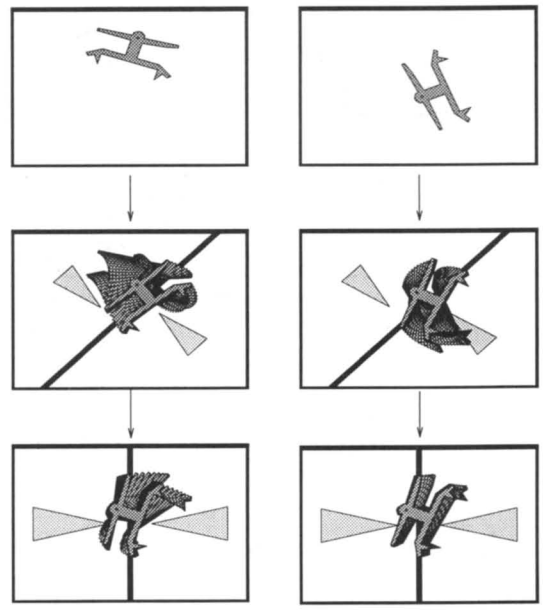


Figure 1: *Sensorless parts orienting using force vector fields: The part reaches unique orientation after two subsequent squeezes. There exist such orienting strategies for all polygonal parts. See* `www.cs.dartmouth.edu/~brd/demo/MicroManipulation` *for an animated simulation.*

[1]The theory of programmable force fields for micro manipulation tasks was introduced in [24].

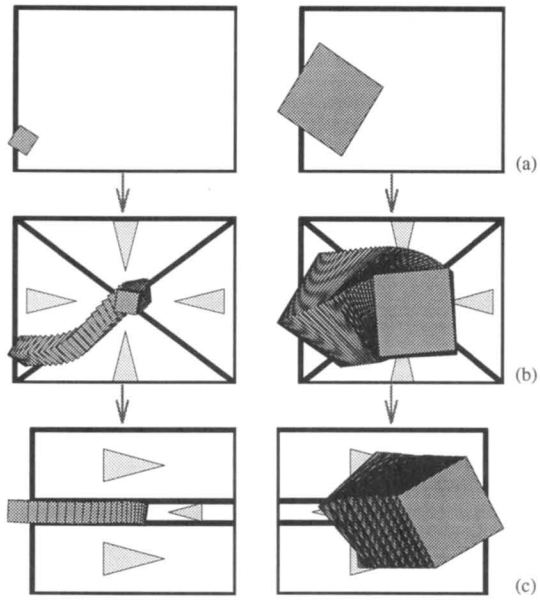


Figure 2: *Sensorless sorting using force vector fields: parts of different sizes are first centered and subsequently separated depending on their size.*

rier between applications requiring array micromanipulation and their implementation with MEMS devices. Such abstraction barriers permit hierarchical design, and allow application designs with greater independence from underlying device technology.

We are interested in the algorithmic content of MEMS control strategies. This paper surveys our work on showing how to quantify it, by analyzing the complexity of (1) computing a manipulation plan, (2) the generated plans (i.e., force field sequences) and (3) the individual fields. We survey our results on obtaining upper and lower complexity bounds on plans and fields, and in finding trade-offs between these different kinds of complexity. We review how to solve the problems of planning and control of micro actuator arrays for a wide range of tasks, and present a tutorial on algorithms that automatically generate manipulation plans for translating, orienting, centering, and sorting small parts [17, 19].

When a part is placed on the array, the programmed vector field induces a force and moment upon it. Over time, the part may come to rest in a dynamic equilibrium state. By chaining together sequences of vector fields, the equilibrium states of a part in the field may be cascaded to obtain a desired final state—for example, this state may represent a unique orientation or pose of the part (see Figures 1 and 2a-b). The resulting strategies work from any initial configuration (pose) of the part, require no sensing, and enjoy efficient planning algorithms. A system with such a behavior exhibits the *feeding property* [2]:

> A system has the *feeding property* over a set of parts $\mathcal{P}$ and a set of initial configurations $\mathcal{I}$ if, given any part $P \in \mathcal{P}$, there is some output configuration $\mathbf{q}$ such that the system can move P to $\mathbf{q}$ from any location in $\mathcal{I}$.

In Section 5 we present several force vector fields that possess the feeding property. Some of these fields are more powerful part feeders, as they allow the choice of a specific output configuration $\mathbf{q}$. Our work on programmable force fields is related to nonprehensile manipulation [32, 61, 35]: in both cases, parts are manipulated without form or force closure.

Even though our fields are typically not smooth, it is possible to define a *potential* for certain fields (as a unique path integral), and to show that when an array ($\mathbb{R}^2$) potential exists, it always lifts to a smooth configuration space ($\mathbb{R}^2 \times \mathbb{S}^1$) potential for the manipulated part [19]. Vector fields with potential have been shown to be theoretically well-suited for manipulation strategies, by classifying a sub-family of potential fields in which every part has stable equilibria [19]. Hence, such fields have been proposed for manipulation tasks in which we desire to cascade equilibria in order to uniquely pose or orient a part (for details see [24, 19, 20]). The theory of programmable force fields permits calculation of stable equilibria for macroscopic parts in potential fields[2] (see Figure 15). In our experiments, we employed vector fields with potential for parts-orientation and -posing tasks, and the theory was

[2]While this question has been well-studied for a point mass in a field, the issue is more subtle when lifted to a body with finite area, due to the moment covector. See [19] for details.

used to predict the equilibrium poses of specific parts (Figure 14). The poses predicted by the equilibrium analysis were observed in our experiments (Figures 13-left).

Perhaps surprisingly, the theory also predicts the existence of pathological fields which do not induce well-behaved equilibria. In particular, there exist perfectly plausible vector fields which induce *no* stable equilibrium in very simple parts. Although these fields are very simple, they result in limit cycles and quite complex behavior. We implemented such fields on an organic micro cilia array (see Section 3.2) [59, 26, 21]. Vector fields *without* potential were employed to cast parts into limit cycles, e.g. "infinite" rotation using a skew-symmetric squeeze field. The predicted behavior (Figure 16) for such "unstable" vector fields was also observed (Figure 13-right). This shows that rather complex—but potentially useful—behavior can be generated using very simple fields.

We survey recent experiments in implementing this theory using microfabricated actuator arrays [19, 26, 21]. In these experiments, strategies were programmed in a fine-grained SIMD fashion by specifying planar force vector fields. These programmable fields were implemented by moving the individual actuators in a cyclic, gait-like fashion. Motion in non-principal (e.g. diagonal) directions was effected by a pairwise coupling of the actuators to implement *virtual actuators* and *virtual gaits*, (analogous to the virtual legs employed by Raibert's hopping and running robots [52]). The tasks of parts-translation, -rotation, -orientation, and -centering were demonstrated using small IC dice.

The theory of programmable force fields and virtual gaits gives a method for controlling a very large number of distributed actuators in a principled, geometric, task-level fashion. Whereas many control theories for multiple independent actuators break down as the number of actuators becomes very large, our systems will only become more robust as the actuators become denser and more numerous.

2 Research Growth

When we began working in MEMS in 1991, it was not immediately clear what fundamental algorithmic or computational problems arose in this new area. Even after obtaining our first results on the theory of programmable force vector fields in 1993 [22], the received view in the community was that the chief computational issues in MEMS arose in (1) design, and (2) simulation. Indeed, it is these problems that motivated our initial efforts (see, e.g., [5, 6, 8, 7]): in 1990, Ralph Merkle of Xerox PARC and Kris Pister of Berkeley urged us to apply our results on the design and simulation [33] of snap fasteners to MEMS [51].

At that time, work on force fields for manipulation had been limited to the artificial potential fields first developed by Khatib, Koditschek, and Brooks.[3] While potential fields have been widely used in robot control [42, 43, 56, 53], micro-actuator arrays present us with the ability to *explicitly* program the applied force *at every point* in a vector field. Whereas previous work had developed control strategies with *artificial* potential fields, our fields are non-artificial (i.e., *physical*). Artificial potential fields require a tight feedback loop, in which, at each clock tick, the robot senses its state and looks up a control (i.e., a vector) using a state-indexed navigation function (i.e., a vector field). In contrast, physical potential fields employ no sensing, and the motion of the manipulated object evolves open-loop (for example, like a particle in a gravity field). This alone makes our application of potential field theory to micro-devices a different, and algorithmically challenging enterprise. Such fields can be composed using addition, sequential composition, or "parallel" composition by superposition of controls.

After our first paper appeared in January 1994 [24], it became clear that there exist deep and challenging algorithmic problems in MEMS and programmable vector fields, at the intersection of combinatorial algorithms, geometry, dynamical systems, and distributed

[3] A notable exception are the three-dimensional force fields used by Joffe and his collaborators at JPL [39], where AC magnetic fields are used to orient and assemble ferromagnetic parts – in 3D!

systems. A flurry of papers has emerged on new algorithms, new analysis, and new arrayed devices for programmable vector fields. During the next few years, the authors continued working with Noel MacDonald at the Cornell Nanofabrication Facility to develop and test new arrays of MEMS microactuators for programmable vector fields [18, 18, 19, 20, 23]. At the same time, we worked with Greg Kovacs' group at the National Nanofabrication Facility at Stanford, to develop a control system for MEMS organic ciliary arrays, and to perform manipulation experiments with these arrays to manipulate IC dice using array-induced force fields [21, 26]. In parallel, we worked with Ken Goldberg at Berkeley and Vivek Bhatt at Cornell to generalize the theory to macroscopic devices, by developing algorithms for transversely vibrating plates in order to implement programmable vector fields [11, 10]. Finally, Danny Halperin, working with the authors, developed new upper and lower bounds, output-sensitive algorithms, and a precise computational-geometric analysis of the area bisectors arising in squeeze-field algorithms [14, 15, 16].

Other groups have also been active in developing new devices, analysis, and algorithms. Ken Goldberg worked with John Canny's group at Berkeley, to continue research on using vibrating plates for manipulation, showing that longitudinal vibrations can generate a rich vocabulary of programmable vector fields [55]. In addition, they developed sophisticated dynamic models and dynamic simulators for both MEMS devices and macroscopic vibrating plates [54]. Lydia Kavraki explored the power of continuous vector fields, and demonstrated an elliptical potential field capable of posing any part into one of two equilibrium states [40]. Peter Will and his colleagues at ISI have explored a number of different MEMS array designs, as well as algorithms and analysis for programmable vector fields [45, 28, 30, 29]. Andy Berlin, David Biegelsen, P. Cheung, and Warren Jackson at Xerox PARC have developed a novel MEMS microactuator array based on controllable air jets, with integrated control and sensing circuitry [4, 27]. Working at CMU, Bill Messner and Jonathan Luntz developed a small room whose floor is tiled with controllable, programmable, macroscopic wheels that can be driven and steered to manipulate large objects such as boxes [46]. Such a system can implement programmable vector fields. Together with Howie Choset, they analyzed the resulting dynamical system to obtain interesting results on controllability and programmable vector field algorithms based on conservative vs. non-conservative fields [47]. Daniela Rus has developed three-dimensional reconfigurable tilings of actuators that could deliver 3D programmable vector fields for manipulation and locomotion [44]. Working with the Berkeley Sensors and Actuators Group (BSAC), Karl Böhringer and Ken Goldberg explored how MEMS devices employing electrostatic fringing fields can be used to implement programmable force fields for parts manipulation and self-assembly [13, 25].

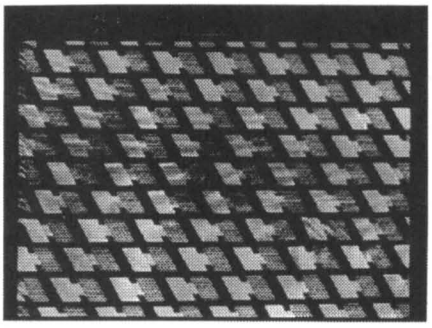


Figure 3: *Prototype* M-CHIP. *A large unidirectional actuator array (scanning electron microscopy). Each actuator is* $180 \times 240\,\mu m^2$ *in size. Detail from a* $1\,in^2$ *array with more than 11,000 actuators. For more pictures on device design and fabrication see* `www.cs.dartmouth.edu/~brd/demo/MicroActuators`.

In short, there has been an explosion of new and exotic arrayed devices for both MEMS manipulation and macroscopic manipulation. The theory of programmable vector fields has been applied and extended to a variety of devices and systems. It is somewhat remarkable that the same analysis tools, fields, and algorithms apply to such a wide range of systems.

3 Experimental Devices and Setup

Several groups have described efforts to apply MEMS (micro electro mechanical system) actuators to po-

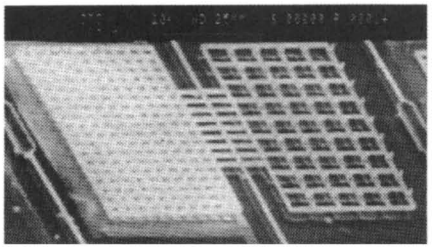


Figure 4: *Released asymmetric actuator for the* M-CHIP *(scanning electron microscopy). Left: Dense grid (*$10\,\mu m$ *spacing) with aluminum electrode underneath. Right: Grid with* $5\,\mu m$ *high poles.*

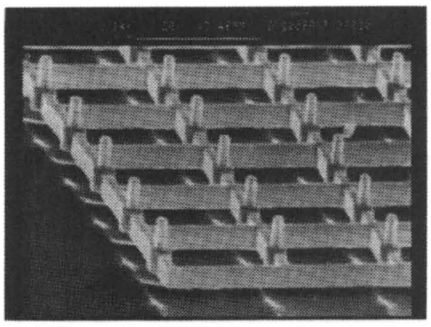


Figure 5: *Released* M-CHIP *actuator consisting of single-crystal silicon with* $5\,\mu m$ *high tips, suspended* $5\,\mu m$ *above the silicon substrate.*

sitioning, inspection, and assembly tasks with small parts [50, 3, 37, 24, 45, for example]. However, the fabrication, control, and programming of micro-devices that can interact and actively change their environment remains challenging. Problems arise from

1. the limited range of motion and force that can be generated with microactuators,
2. the lack of sufficient sensor information with regard to manipulation tasks,
3. design limitations and geometric tolerances due to the fabrication process, and
4. uncertain material properties and the lack of adequate models for mechanisms at very small scales.

In the following subsections we describe two different types of actuator arrays: single-crystal silicon (SCS) electrostatic actuators, and combined thermobimorph and electrostatic organic micro cilia.

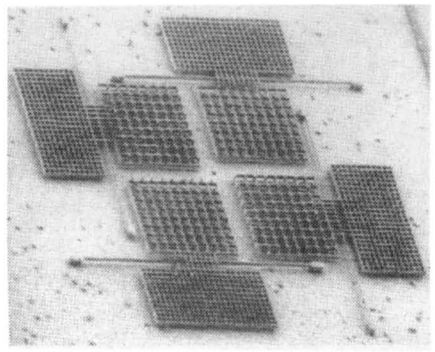


Figure 6: M-CHIP *prototype motion pixel consisting of actuators oriented in four different directions.*

3.1 Single-crystal silicon electrostatic actuators

We now survey recent work on single-crystal silicon electrostatic MEMS actuators, conducted in collaboration with Noel MacDonald at the Cornell Nanofabrication facility [18, 18, 19, 20, 24, 23].

The M-CHIP (Manipulation Chip, see Figure 3) is fabricated using a SCREAM (Single-Crystal Silicon Reactive Etching and Metallization) process developed in the Cornell Nanofabrication Facility [60, 58]. The SCREAM process is low-temperature, and does not interfere with traditional VLSI [57]. Hence it opens the door to building monolithic microelectromechanical systems with integrated microactuators and control circuitry on the same wafer.

The design is based on microfabricated torsional resonators [49, 48]. Each unit device consists of a rectangular grid etched out of single-crystal silicon suspended by two rods that act as torsional springs (Figure 4). The grid is about $200\,\mu m$ long and extends $120\,\mu m$ on each side of the rod. The rods are $150\,\mu m$ long. The current asymmetric design has $5\,\mu m$ high protruding tips on one side of the grid that make contact with an object lying on top of the actuator (Figure 5). The other side of the actuator consists of a denser grid above an aluminum electrode. If a voltage is applied between silicon substrate and electrode, the dense grid above the electrode is pulled downward by the resulting electrostatic force. Simultaneously the other side of the device (with the tips) is deflected out of the plane by

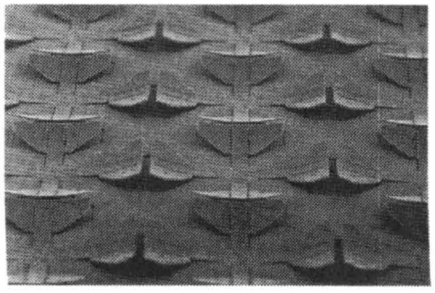


Figure 7: *Portion of a polyimide cilia array (SEM micrograph). Four orthogonally-oriented actuators are integrated into a motion pixel, which covers a surface area of approximately* $1.1 \times 1.1\, mm^2$.

several μm. Hence an object can be lifted and pushed sideways by the actuator.

Because of its low inertia (resonance in the high kHz range) the M-CHIP can be driven in a wide frequency range from DC to several $100\, kHz$ AC. The actuators need not be operated at resonance: They can also be servoed to periodically "hit" an object on top, hence applying both lateral and vertical forces. Calculations, simulations and experiments have shown that the force generated with a torsional actuator is approximately $10\, \mu N$, which corresponds to a force-per-area ratio of $100\, \mu N/mm^2$, large enough to levitate e.g. a piece of paper ($1\, \mu N/mm^2$) or a silicon wafer ($10\, \mu N/mm^2$).

Each actuator can generate motion in one specific direction if it is activated; otherwise it acts as a passive frictional contact. Figure 3 shows a small section of a unidirectional actuator array, which consists of more than 11,000 individual actuators. The combination and selective activation of several actuators with different motion bias allows us to generate various motions in discrete directions, spanning the plane (Figure 6). In initial manipulation experiments, small pieces of glass (size: several mm^2, mass: about $1\, mg$) were lifted within the motion range of the actuators (several μm) and pushed sideways by several $100\, \mu m$.

The fabrication process and mechanism analysis are described in more detail in [24, 23, 17].

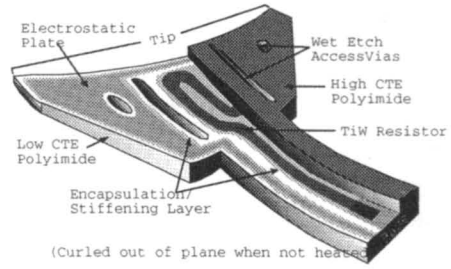


Figure 8: *Organic thermal and electrostatic microactuator. Half of the upper polyimide and silicon nitride encapsulation/stiffening layer are shown removed along the cilium's axis of symmetry to show details.*

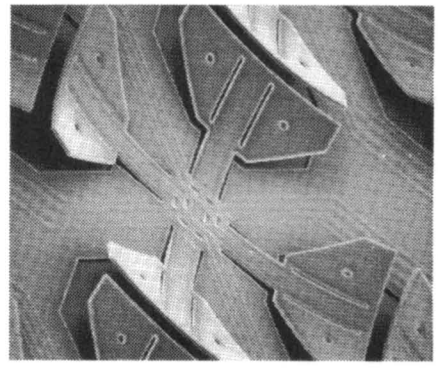


Figure 9: *Polyimide cilia motion pixel (SEM micrograph). Four actuators in a common center configuration make up a motion pixel. Each cilium is* $430\, \mu m$ *long and bends up to* $120\, \mu m$ *out of the plane.*

3.2 Combined thermobimorph and electrostatic polyimide cilia arrays

We now survey recent work on using MEMS organic cilliary arrays [59] for massively parallel microanipulation, conducted in collaboration with John Suh and Greg Kovacs at the Stanford University National Nanofabrication facility [21, 26].

At Stanford, surface micromachining techniques were used to create organic micro cilia arrays, consisting of polyimide as the primary structural material and aluminum as a sacrificial layer. The fabrication process (developed at the Center for Integrated Systems, Stanford University) was designed to be compatible with CMOS or BiCMOS circuits which could be pre-fabricated on a silicon substrate [59].

Actuator cilia. Each cilium consists of two layers of polyimide with different thermal expansion coefficients. The cilium also contains a Titanium-Tungsten (Ti:W) resistive heater loop for thermal actuation, Aluminum electrodes for electrostatic (low-power) hold-down, and a silicon nitride encapsulation/stiffening layer (Figure 8). For a detailed description of the fabrication process see [59]. Vertical and horizontal displacements of the cilia tips are a function of the thermal mismatch in the actuator layers. For room temperature, these values can be calculated as $\delta_v \approx 120\,\mu m$ and $\delta_h \approx 20\,\mu m$ [59]. Inspection under the scanning electron microscope (SEM) has verified these calculations.

The lifting capacity of an actuator can be estimated as the force required to deflect the actuator's tip to the substrate. The actuator load capacity has been calculated as $F_l = 76\,\mu m$, which gives a force-per-area ratio of $4 \times 76\,\mu N/(1.1\,mm)^2 \approx 250\,\mu N/mm^2$.

Chip layout. The cilia array is composed of cells (*motion pixels*, each $1.1 \times 1.1\,mm^2$) which contain four orthogonally-oriented actuators (Figure 9). On the current cilia chip, the motion pixels are arranged in an 8×8 array which occupies approximately $0.77\,cm^2$ of a $1\,cm^2$ die. The four actuators of each pixel are independently activated by four thermal and four electrostatic control lines. Four 8×8 chips are diced and packaged together to make a quad-shaped 16×16 cilia array device, with a total of 1024 cilia. The device itself is attached to a hybrid package which is placed on a heat sink and thermo-electric cooler (Peltier effect module). The total input power to the chip can exceed $4\,W$, and without active cooling the package can become very hot. To observe the experiments, a long working distance microscope is connected to a CCD camera, and a video cassette recorder is used to monitor and record both the movements of an individual cilium and the objects conveyed by the array.

Controller. The manipulation results described below were accomplished with the cilia array device interfaced to an IBM 486 personal computer. The PC provides speed control via the drive pulse frequency and directional control interactively via keyboard or mouse, or by actuator programs that can be specified using the MEMSA (MEMS Array) language[4] which we developed at Stanford. The control software including the MEMSA interpreter was written in PASCAL. Thermal and electrostatic control line signals are sent via the PC parallel port to D-type flip-flops which activate power transistors. Currently up to 4 cilia arrays can be controlled simultaneously by using a multiplexer with 2 address bits.

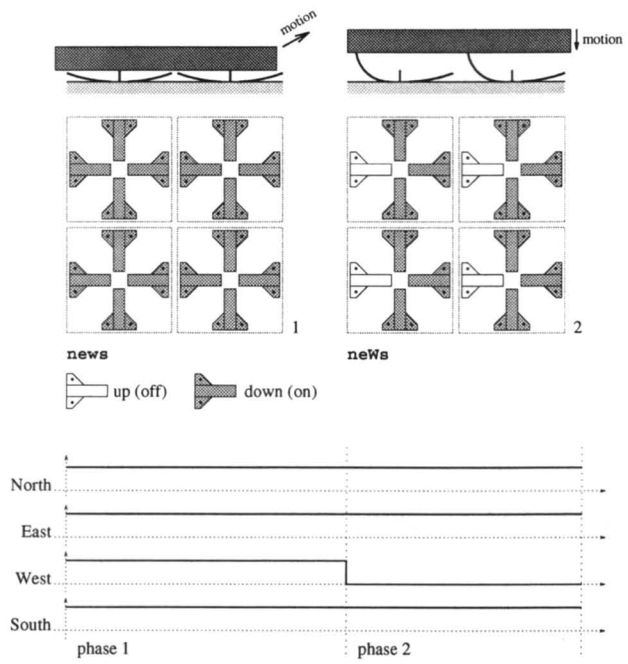


Figure 10: *Two-phase gait. The* `W` *actuator is repeatedly switched on and off, while the other actuators always remain on, resulting in a* `news neWs` *sequence.*

4 Low-level Control: Actuator Gaits

We now survey some basic concepts of actuator gaits using MEMS arrays for massively parallel microanipulation, developed in collaboration with Jonathan Suh and Greg Kovacs at the Stanford University National Nanofabrication facility [21, 26].

To induce motion on a part that is placed on the array, the cilia are actuated in a cyclic, gait-like fashion.

[4]MEMSA (named after MENSA) is the language for smart manipulation surfaces.

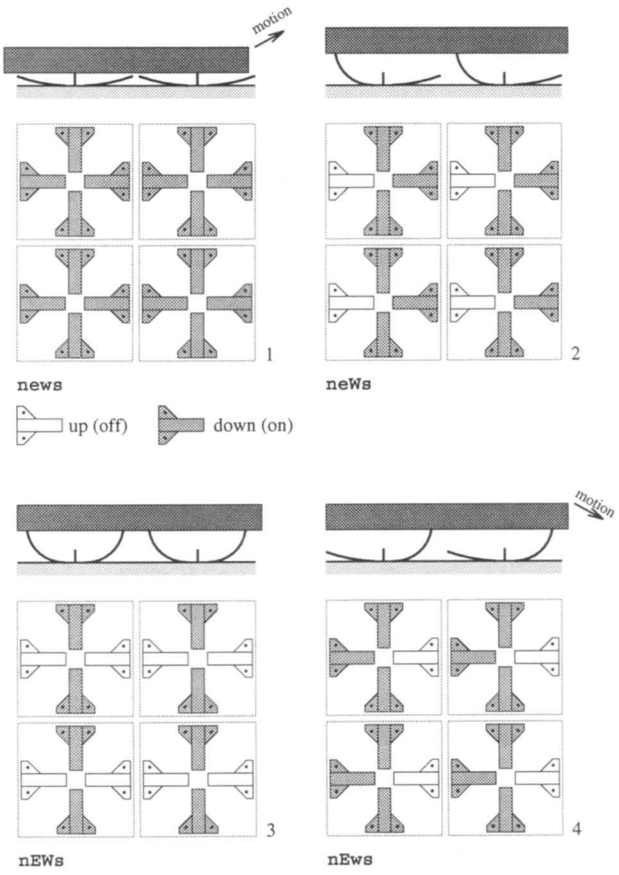


Figure 11: *Four-phase gait consisting of the four-pattern sequence* **news neWs nEWs nEws**.

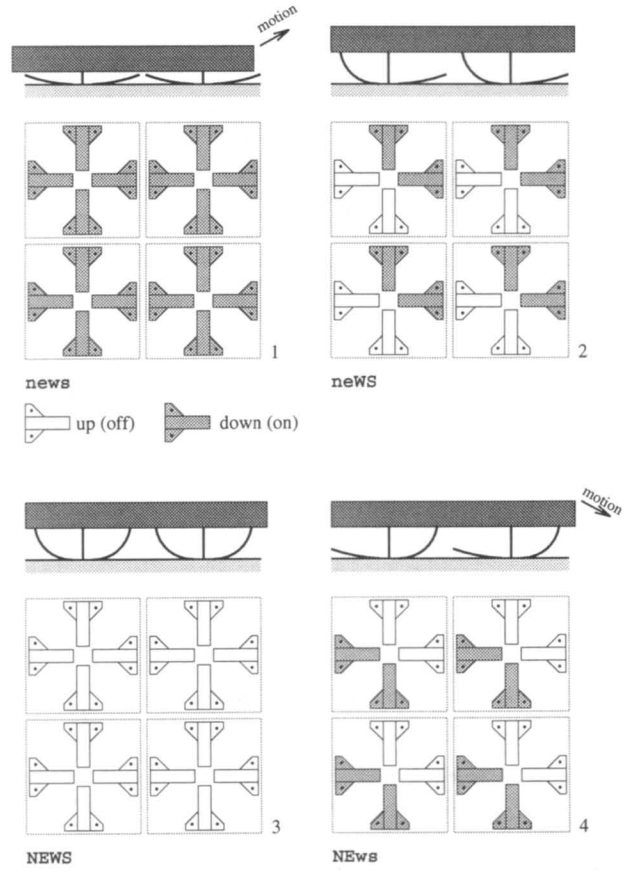


Figure 12: *Diagonal (virtual) gait consisting of the four-pattern sequence* **news neWS NEWS NEws**. *The* **N** *and* **E** *cilia, and the* **W** *and* **S** *cilia, are coupled to form virtual cilia in North-East and South-West directions.*

In each cycle, the part is moved in a certain direction by the motion of the actuators that are in contact with it. The speed of the moving part depends on the (horizontal) displacement of the actuators per cycle as well as the frequency of cycle repetition. It also depends on the surface properties and weight of the moving part.

Task: translation of parts in principal directions. The simplest gait is the *two-phase gait*, in which all actuators of the same orientation repeatedly stroke the part while the remaining actuators are held down. Assuming that the orthogonal cilia within a motion pixel are oriented at the principal compass points, let us use capital letters **NEWS** to denote the North, East, West, and South actuators in the up position, and lower-case letters **news** to denote the actuators in the lowered position. Then the two-phase gait to effect motion in the East direction would be **news neWs** (see Figure 10).

The *four-phase gait* consists of four different actuation phases **news neWs nEWs nEws** such that motion is induced during upward as well as downward strokes of the cilia (Figure 11, see also [3]). Note that the forces exerted on the moving part depend on the state of the motion pixel: e.g. in the transition from phase 1 to phase 2 the cilium **W** moves up while the opposing cilium **e** remains down. We denote the lateral force exerted on the part in this configuration $f_{\mathtt{W},\mathtt{e}}$. Analogously, during transitions 2–3, 3–4, and 4–1 we observe lateral forces $f_{\mathtt{E},\mathtt{W}}$, $f_{\mathtt{w},\mathtt{E}}$, and $f_{\mathtt{e},\mathtt{w}}$, respectively. $f_{\mathtt{W},\mathtt{e}}$ and

$f_{e,w}$ are in positive x-direction, while $f_{E,W}$ and $f_{w,E}$ are negative. Furthermore, from this analysis it follows that $|f_{W,e}| \gg |f_{e,w}| \gg |f_{E,W}| \approx |f_{w,E}| \approx 0$. Hence we expect a relatively large motion step $\Delta x_{W,e}$ during transition 1–2, and a smaller step $\Delta x_{e,w}$ during transition 4–1, while during the other transitions the part remains at its current location. This behavior has been observed in our experiments, where $\Delta x_{W,e}$ was measured between $3\,\mu m$ and $10\,\mu m$ depending on input power, frequency, surface properties and weight of the part. $\Delta x_{W,e}$ was usually about twice as large as $\Delta x_{e,w}$.

Task: translation of parts in arbitrary directions. Motion in non-principal (e.g. diagonal) directions is effected by a pairwise coupling of two cilia of each pixel, implementing *virtual cilia* analogous to Raibert's concept of virtual legs for hopping and running robots [52]. Hence, several cilia can be coordinated to emulate a virtual cilium, which generates a force corresponding to the vector sum of its components. The diagonal gait to effect motion in the North-East direction would be `news neWS NEWS NEws` where the virtual cilia are `NE` and `WS`. Consequently, we obtain a *virtual gait* that moves the part in a diagonal direction. Note that in a section view through the array looking in the North-West direction, this gait looks virtually identical to the four-phase gait depicted in Figure 11.

Motion in arbitrary directions can be induced by alternating gaits that interleave principal (or virtual) gaits of different directions. For example, a translation at 25° from the x-axis requires motion in the y-direction and x-direction at a ratio of $\tan 25° \approx 1 : 2$. Our control software determines the exact alternation analogously to the Bresenham line scan algorithm [36], which rasterizes lines at arbitrary angles, resulting in different fields that are interlaced in time.

Experiments and results. In collaboration with Jonathan Suh and Greg Kovacs at the Stanford University National Nanofabrication facility [21, 26], a large number of translation experiments have been performed in which two-phase and four-phase gaits were used to implement principal and virtual gaits. These experiments show that a first-order dynamical system models the device-part interaction well.[5] Therefore, when describing and predicting the motion of parts in force vector fields, we have based our theory on a first-order system (see Section 5).

Silicon chips were moved with a motion resolution of a few μm and speeds up to $200\,\mu m/sec$. Four-phase gaits proved more effective than two-phase gaits, because during the downward motion in the two-phase gait, the part tends to slip backwards. The four-phase gait avoids this effect, because other cilia hold the part in place during the transition 3–4. In the subsequent downward motion in the transition 4–1, the part is also moved forward (Figure 11).

The diagonal gait also has the lowest power consumption (not considering electrostatic hold-down), due to the fact that its duty cycle for cilia hold-down is lowest (50 %), compared to 75 % for the principal four-phase gait, and 87.5 % for the two-phase gait.

As expected, diagonal (virtual) gaits induced the largest and fastest motion because all four cilia of each pixel were activated, whereas in principal gaits only two cilia are actively used, while the others have to be held down continuously (Figure 11).

5 High-level Control: Vector Fields

We believe that vector fields can be used as an *abstraction barrier* between applications requiring array micro-manipulation and MEMS devices implementing the requisite mechanical forces. That is, applications such as parts-feeding can be formulated in terms of the vector fields required. This then serves as a specification which the underlying MEMS device technology

[5] While experiments with our MEMS arrays indicate they may be modelled as first order (heavily-damped or essentially quasi-static) systems, it is easy to imagine future MEMS arrays that are highly dynamic and which would require a detailed analysis of dynamic equilibria. Indeed, our vibrating plates exhibit more dynamics, although empirically the quasi-static analysis has been adequate so far [11, 10]. Some experimental evidence for quasi-staticity is discussed in [21, 26], and the effect of dynamics on the quasi-static model is discussed in [20]. Research on dynamic analysis of programmable vector fields is an active research area; see, e.g., [54, 55, 28, 30, 29, 47].

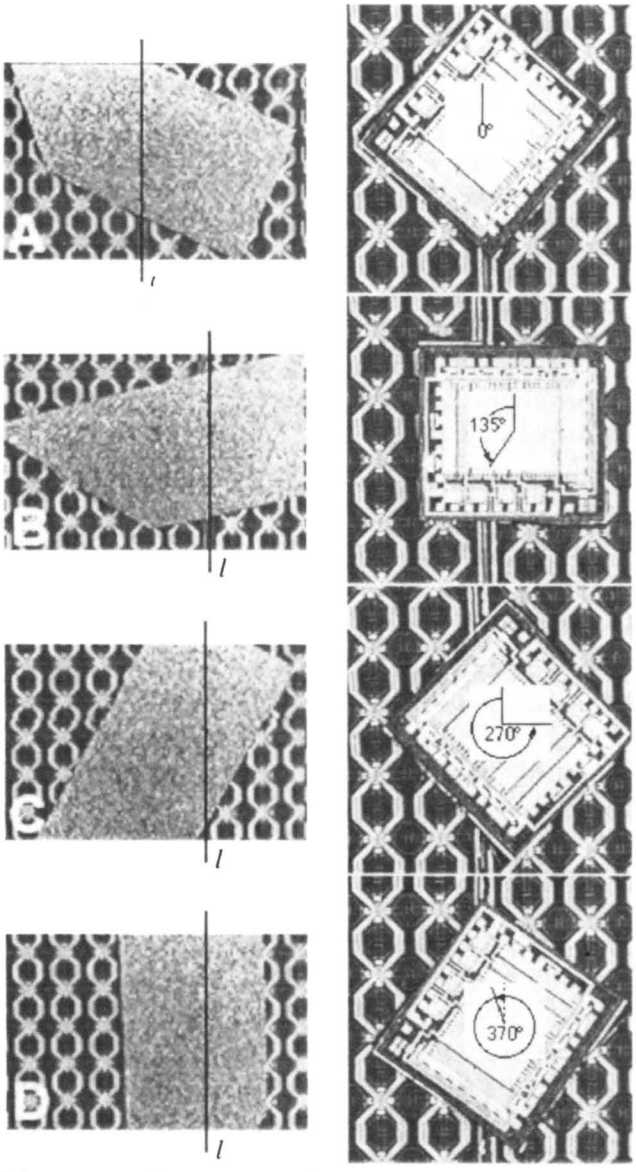


Figure 13: *Manipulation of silicon chips in programmable vector fields induced by a micro cilia array (microscope video images). Left: The chip is aligned with the vertical squeeze line l (marked by a dark line for clarification). Right: Rotating a square-shaped chip counterclockwise in a skewed squeeze field.*

must deliver. Conversely, the capabilities of MEMS array technologies for actuation can be formulated in terms of the vector fields they can implement. For example, limitations in force magnitude are naturally expressed in vector field terms, as "small" vector fields. Restrictions in directional selectivity and magnitude control can also be manifested as restrictions on the vector field abstraction (resulting in discretization in orientation or modulus). This means that MEMS designers can potentially ignore certain details of the application process, and instead focus on matching the required vector field specification. Then, once the capabilities of MEMS actuator arrays were published as vector fields and tolerances, an application designer could look in a catalog to choose a device technology based on the field specification it promises to implement. This would free application engineers from needing to know much about process engineering, in the same way that software and algorithm designers often abstract away from details of the hardware. Such an abstraction barrier could permit hierarchical design, and allow application designs with greater independence from the underlying device technology. At the same time, abstraction barriers could allow MEMS array technologies to be designed simultaneously with the (abstract) vector field control. This development pattern could be similar to the concurrent design of VLSI processors with their compilers, as is common in computer architecture.

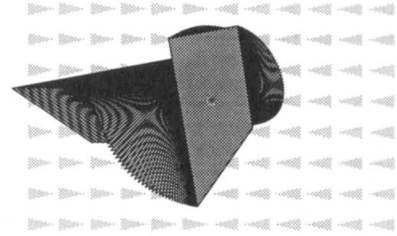


Figure 14: *Simulation of the alignment task with a squeeze field as shown in Figure 13-left.*

We now survey some basic concepts and theory in high-level (algorithmic) control, and describe experiments to test the validity of the theory. The experimental results surveyed below in Sec. 5 were conducted in collaboration with Jonathan Suh and Greg Kovacs at the Stanford University National Nanofabrication facility [21, 26]. In this survey, we emphasize the geometric and algorithmic aspects of the problem, as well connections and synergy with experimental work on design, fabrication, and programming of MEMS arrays.

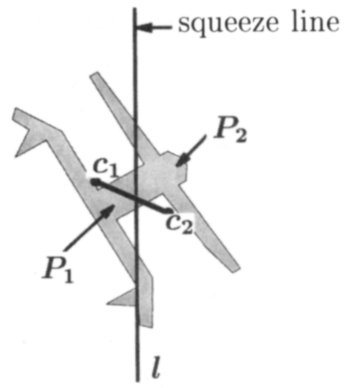


Figure 15: *Equilibrium condition: To balance force and moment acting on P in a unit squeeze field, the two areas P_1 and P_2 must be equal (i.e., l must bisect the part), and the line connecting the centers of area c_1 and c_2 must be perpendicular to the squeeze line l.*

5.1 Squeeze fields

In [24], the authors proposed a family of control strategies called *squeeze fields* and a planning algorithm for parts-orientation (see Figures 1 and 15).

Definition 1 [19] Given a straight line l, a *squeeze field* f is a two-dimensional force vector field in which, at each point, a unit force points perpendicularly towards l (on l the force is zero).

We refer to the line l as the *squeeze line*, because l lies in the center of the squeeze field.

Assuming quasi-static motion, an object will translate and rotate to an equilibrium configuration, as characterized in Figure 15. To predict the equilibria, we assume a uniform force distribution over the surface of P, which is a reasonable assumption for a flat part that is in contact with a large number of elastic actuators.

Definition 2 A part P is in *translational equilibrium* if the forces acting on P are balanced. P is in *orientational equilibrium* if the moments acting on P are balanced. *Total equilibrium* is simultaneous translational and orientational equilibrium.

Claim 3 [19] *Every squeeze field f (see Definition 5.1) has potential, of the form $U(\mathbf{z}) = \int_\alpha f \cdot ds$, where α is an arbitrary path to $\mathbf{z}$ from a fixed reference point. If $d_\mathbf{z}$ denotes the perpendicular distance of $\mathbf{z}$ from the squeeze line, then $U(d_\mathbf{z}) = |d_\mathbf{z}|$.*

Claim 4 [19] *Let P be a connected polygonal part with finite contact area and n vertices. Then in any squeeze field, P has $E = O(kn^2)$ orientation equilibria, where k is the maximum number of edges that a bisector of P can cross. If P is convex, then the number of equilibria is $O(n)$.*

Equilibria can be calculated numerically using the method in Figure 15: Given an arbitrary part at a fixed orientation, we translate it until its left and right sections have equal size. If the respective centers of gravity lie on a line perpendicular to the squeeze line, then the part is in equilibrium. For polygonal parts there exist analytical methods to compute the equilibria exactly (see [19] for a detailed algorithm and a derivation of the $O(kn^2)$ bound).

Claim 5 [19] *Let P be a polygon whose interior is connected. There exists a sensorless alignment strategy consisting of a sequence of squeeze fields that uniquely orients P up to symmetries.*

Proof: Claim 4 states that a squeeze field brings a polygon P into one of $E = O(kn^2)$ orientation equilibria. We define the *squeeze function* as this mapping from original orientation to equilibrium orientation. Hence, from Claim 4 it follows that the image of the squeeze function is a set of E discrete values. Given such a squeeze function, Goldberg has presented an algorithm for sensorless manipulation of polygons [38] that constructs an orienting strategy with $O(E)$ steps in $O(E^2)$ time. The output of this algorithm is a sequence of squeeze directions. When the corresponding squeeze fields are applied to the part P, the set of possible orientations is successively reduced until a unique orientation (up to symmetry) is reached. For details see [19] and [38]. □

Task: orienting and aligning parts

If a part is placed in a squeeze field, it will translate and rotate until a stable equilibrium is reached

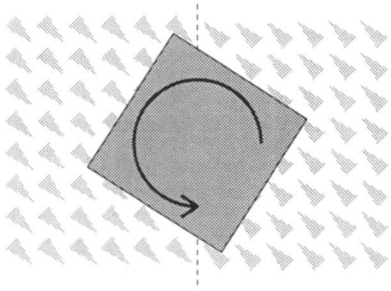


Figure 16: *Unstable square-shaped part in a skewed squeeze field ($\epsilon = -1$). The square with center on the squeeze line will rotate indefinitely. Moreover, it has* no *stable equilibrium in this field.*

(Claim 4). Parts may exhibit several equilibria, hence after one squeeze the part orientation may be ambiguous. This ambiguity can be removed with the strategies of Claim 5: by executing a sequence of squeezes at particular angles, the part is uniquely oriented (see Figure 1).

Experiments and results. The long, thin part depicted in Figures 13-left and 14 exhibits a unique stable equilibrium (modulo 180° field symmetry). When placed in a squeeze field, its longitudinal axis aligns with the squeeze line. This dynamic process is predicted by simulation in Figure 14, and verified in experiment (see Figure 13-left). This part alignment experiment has also been performed with similar results for several other small pieces of glass and silicon of a few *mm* length and several *mg* of mass.

5.2 Skewed squeeze fields

Definition 6 [19] A *skewed field* f_S is a vector field given by $f_S(x, y) = -\text{sign}(x)(1, \epsilon)$, where $0 \neq \epsilon \in \mathbb{R}$.

Claim 7 [19] *No skewed squeeze field has a potential.*

In a skewed squeeze field, it is easy to find a circular path along which the work integral is non-zero (e.g., along a circle with center on the squeeze line).

Claim 8 [19] *A skewed field induces* no *stable equilibrium on a disk-shaped part (for all $\epsilon \neq 0$).*

Force equilibrium is only possible if the center of the disk coincides with the squeeze line. In this position the disk experiences a non-zero moment if $\epsilon \neq 0$.

Claim 9 *A diagonally skewed field ($\epsilon = \pm 1$) induces* no *stable equilibrium on a square-shaped part.*

For a proof see [19].

Task: rotating parts

According to Claims 8 and 9, certain parts will rotate indefinitely in skewed squeeze fields (Figure 16). Note that even though our cilia device has more degrees of freedom, two areas of constant force are sufficient to implement a skewed field, resulting in a very simple task-level rotation strategy. In particular, the rotation algorithm resulting from the application of skew-symmetric squeeze fields is considerably simpler than rotation algorithms proposed in the MEMS literature (for example, the vortices suggested by Fujita [37] or by Liu and Will [45]). Vortices require at least four areas of the array to be pushing in different directions. That is, vortices can be implemented using four triangular or rectangular regions, upon each of which the vector field is constant. Skewed fields perform the same task with only two regions of constant force.

Experiments and results. Figure 13-right shows video frames of a $3 \times 3\, mm^2$ IC chip rotating on the squeeze line of a skewed field. During the experiments of approximately 10 min, several full rotations of the part were performed.

5.3 Radial fields

Definition 10 A radial field f is a two-dimensional force vector field such that $f(\mathbf{z}) = -\mathbf{z}/|\mathbf{z}|$ if $\mathbf{z} \neq 0$, and $f(\mathbf{0}) = \mathbf{0}$.

Claim 11 [19] *A radial field has a potential, $U(\mathbf{z}) = |\mathbf{z}|$.*

Claim 12 [19] *Given a polygonal part P in a radial field f, there exists a unique* pivot point *v of P such that P is in equilibrium if and only if v coincides with the center of the radial field.*

Claim 13 [19] *Let P be a polygonal part with n vertices, and let k be the maximum number of edges that a bisector of P can cross. There are at most $E = O(kn)$ stable equilibria in a field of the form $R + \delta S$ if S is a squeeze field, and δ is sufficiently small and positive.*

Proofs of the previous claims, and a numerical algorithm to compute the pivot point is given in [19]. Note that Claim 13 results in strategies for unique part posing in $O(E^2) = O(k^2n^2)$ steps.

Task: centering parts

Radial fields can be used to center a part. With the current four-quadrant cilia device, we have implemented an approximation of an ideal radial field similar to the field in Figure 2-b. Note that this approximate radial field has a potential.

Experiments and results. Small silicon and glass parts were centered using our cilia device. In this experiment, high positioning accuracy (in the μm range) was hard to achieve, because the center of the radial field coincides with the the location of the dice edges. Manual packaging of the four cilia chips leaves small gaps and non-planarities at these junctions. The next generation cilia device will overcome this problem, because it will allow us to implement the radial field with a single chip. Furthermore, because of its full pixel-wise programmability, the new chip will allow us to closely approximate ideal radial fields.

6 Polygon Bisectors and Force Equilibria

In the previous section we introduced vector fields for high-level control of micro actuator arrays. In particular, it was stated that in a squeeze field, polygonal parts have a (usually very small) finite number of orientation equilibria (see Claim 4). Because of this result, squeeze fields play a key role in manipulation strategies. In this section, we survey recent research on the combinatorial, geometric, and algorithmic properties of squeeze fields, by analyzing the *area bisectors* of a part. The results in Sec. 6 were obtained in collaboration with Danny Halperin [14, 15, 16].

There is a direct relationship between equilibria in squeeze fields and area bisectors. Recall Figure 15: a part is in force equilibrium if and only if the squeeze line bisects the part into two sections of equal area.

Definition 14 [14] Let P be a polygon in the plane, possibly with holes, and having n vertices in total. We denote by V the set of vertices of P. For a directed line λ in the plane, we denote by $h_l(\lambda)$ (resp. $h_r(\lambda)$) the open half-plane bounded by λ on the left- (resp. right-) hand-side of λ. The line λ is an *area bisector* of P if the area of $P \cap h_l(\lambda)$ is equal to the area of $P \cap h_r(\lambda)$.

A line λ partitions V into three sets (two of which may be empty): $V \cap h_l(\lambda)$, $V \cap \lambda$, and $V \cap h_r(\lambda)$. We say that two area bisectors of P are *combinatorially distinct* if the partitioning of V as above induced by the two bisectors is different. We say that two area bisectors of P are *combinatorially equivalent* if they induce the same partitioning of V. We assume that the polygon P is connected, and non-degenerate in the sense that the complement of P has the same boundary as P.

An obvious upper bound on the number of distinct area bisectors of a polygon with n vertices is $O(n^2)$—each combinatorial equivalence class of area bisectors is determined by a pair of vertices of the polygon. In Section 6.2 we describe how a polygon with n vertices can have $\Omega(n^2)$ distinct area bisectors [14]. (Note that the polygon in this construction is *simple*.)

We review an output-sensitive algorithm [14] for computing an explicit representation of all the area bisectors of a given polygon, by constructing the *bisector curve* β defined in a plane dual to the plane containing the polygon: the curve β is the union of points dual to area bisectors in the primal plane. The algorithm proceeds by constructing the *zone* of the curve β in an *arrangement of lines* [34] in the dual plane, where each line of the arrangement is the dual of a vertex of the polygon. A sketch of the algorithm is given in Section 6.3.

Area bisectors were considered by Díaz and O'Rourke [31]. However, their focus is on the continuous version of the *ham-sandwich cut* problem, and of

a problem they introduce of *orthogonal four-sections*; see [31] for more details. The problem discussed here can be viewed as a continuous version of the well-known *k-set* problem [34].

In the remainder of this section, we analyze the number of area bisectors, and hence bound the number of force equilibria. The algorithm for computing area bisectors can be used as a preliminary step in designing alignment plans (see Section 6.4).

6.1 Properties of area bisectors

In this section we review several properties of area bisectors of polygons. Some of the proofs have been omitted here; they can be found in [16].

Lemma 15 [14] *Let P be a non-degenerate polygon with n vertices. (1) There exist $O(n^2)$ combinatorially distinct ways in which a line can partition P. (2) Let A and B be the intersections of an area bisector λ with the boundary of the convex hull of P. As the slope of λ varies from $-\infty$ to $+\infty$, A and B progress monotonically counterclockwise on the boundary of the convex hull of P. (3) For every slope $\bar{x}$ there exists a unique bisector λ of P with slope $\bar{x}$.*

Lemma 16 [14] *Let P be a polygon with n vertices. Let s be a point in $\mathbb{R}^2$ and let λ be a line that intersects r edges of P. The area bisectors of P that are combinatorially equivalent to λ and pass through s are determined by the roots of a polynomial equation of degree r.*

It can be shown [16] that the bisectors of a polygon P can be described by a piece-wise algebraic curve, where each piece is described by a polynomial whose degree depends on the number of edges of P intersected by the corresponding bisectors.

6.2 Lower bound

As argued above, a polygon with n vertices can have at most $O(n^2)$ combinatorially distinct area bisectors. In this section we present an example of a simple polygon with n vertices where the bound $\Omega(n^2)$ is attained [14].

Consider Figure 17. All the vertices v_i, v_i', u_i and u_i' lie on a circle whose center is at c. The vertices w_j lie very close to c on a small circle whose center is c as well, along two convex polygonal chains.

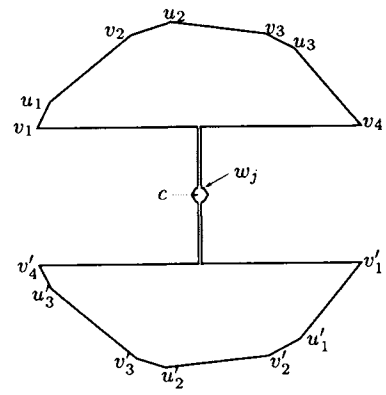


Figure 17: *A simple polygon with n vertices that has $\Omega(n^2)$ combinatorially distinct bisectors.*

We fix an integer m (that we will determine later; for the polygon in the figure $m = 3$). The distance between the vertices v_i and v_{i+1} is the same for $i = 1, \ldots, m$, and it is the same as the distance between v_i' and v_{i+1}' for $i = 1, \ldots, m$. The area of all triangles $v_i u_i v_{i+1}$ for $i = 1, \ldots, m$ is the same and is equal to the area of all triangles $v_i' u_i' v_{i+1}'$ for $i = 1, \ldots, m$. There are $2m$ vertices w_j near c and they are equally spaced on a small circle centered at c. As can be easily verified, for every pair of vertices v_i and v_i', there is a bisector passing through these points that passes also through the center point c. We next claim that as we rotate the bisector from v_i to v_{i+1} it will move off the center c and sweep m vertices w_j. The reason is that the angle $\angle u_i v_i v_{i+1}$ is larger than the angle $\angle u_i' v_i' v_{i+1}'$. Hence, as the bisector rotates, it will proceed 'faster' on the bottom part of our polygon than on the top part and therefore will sweep half of the vertices w_j on its way. Finally m is chosen such that (roughly) $n = 6m + 8$. The number of distinct area bisectors is evidently $\Omega(m^2) = \Omega(n^2)$.

6.3 Output-sensitive algorithm

It is convenient to study the algorithmic problem in a dual plane: a line $y = 2\bar{x}x - \bar{y}$ in the primal plane is transformed into the point $(\bar{x}, \bar{y})$ in the dual plane. A

point (x, y) in the primal plane is transformed into the line $\bar{y} = 2x\bar{x} - y$ in the dual. The dual of an object o will be denoted by o^*. If O is a set of objects in the plane, O^* will denote the set of dual objects.

Let P be a polygon with n vertices as defined in the Introduction, namely connected, non-degenerate and possibly with holes. In the dual plane every vertex v of P is transformed into a line v^* which is the collection of all points dual to lines in the primal plane that pass through v.

For any given direction there is a unique area bisector. We denote the oriented bisector of P that makes an angle θ with the positive x-axis by $B(\theta)$, and (because of symmetry) confine ourselves to the range $[-\pi/2, \pi/2)$ for θ. We denote the collection of points dual to area bisectors of P in that range by β. Note that any θ (besides $-\pi/2$) corresponds to an $\bar{x}$-coordinate in the dual plane.

The curve β is a piece-wise algebraic and $\bar{x}$-monotone curve (this is proved in [16]). We call β the *bisector curve* of P, as it gives a complete specification of all the area bisectors of the polygon P. We denote by κ the number of maximal connected algebraic pieces in β, where the function describing each piece is defined by the fixed set of edges that the corresponding set of bisectors cross. In this section we describe an output-sensitive algorithm from [14] to compute β. Since we aim for output-sensitivity, we cannot afford to compute the entire arrangement $\mathcal{A}(V^*)$ whose complexity may be $\Omega(n^2)$. We will discover the maximal pieces of β in their order along β, using two primitive operations: *ray shooting* among the lines V^*, and intersection of an algebraic curve with a straight line.

We choose an arbitrary direction $\theta_0 \in [-\pi/2, \pi/2)$ and look for the area bisector of P in that direction. This requires $O(n \log n)$ time since the polygon may have holes. Next, we obtain the set of edges crossed by $B(\theta_0)$. We denote by $E(\theta)$ the set of edges crossed by $B(\theta)$. The set $E(\theta_0)$ determines a function $\rho := \rho(\theta_0)$ describing the bisector curve β in a neighborhood of θ_0, as long as the set of edges crossed by the bisector does not change. In the dual plane the function ρ describes the curve β as long as we do not leave the face of $\mathcal{A}(V^*)$ which contains the point $p := (B(\theta_0))^*$.

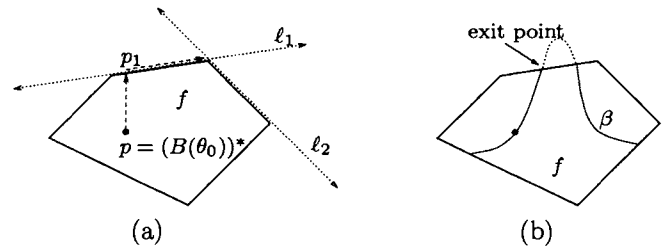


Figure 18: *Ray shooting to determine the face f containing p (a), and then finding the maximal pieces of β inside f and its exit points from f (b).*

Our next step is to construct the face f that contains the point p in $\mathcal{A}(V^*)$. We describe this procedure assuming f is bounded; the extension to unbounded faces is straightforward. We prepare in advance a data structure $R(V^*)$ that supports efficient ray shooting among the lines V^*. We shoot a ray from p in the upward vertical direction and identify the line $\ell_1 \in V^*$ supporting the edge of f above p. See Figure 18(a). We next proceed in clockwise direction along the boundary of f. From the point $p_1 \in \ell_1$ we shoot a ray in $\mathcal{A}(V^*)$ along ℓ_1, and identify the line ℓ_2 supporting the next edge on the boundary of f, and so on until we have returned to ℓ_1 and thus have identified the entire face f.

Now we determine the maximal connected pieces of $f \cap \beta$. For each edge on boundary of f we compute the intersection of its supporting line v^* with β. This computation is equivalent to finding the bisectors that pass through the vertex v, and intersecting the edges in $E(\theta_0)$. By Lemma 16, this reduces to solving a polynomial equation of degree r, where r is the number of edges in $E(\theta_0)$. We denote the time required to find these roots by $\psi(r)$.

We order the resulting intersections along the $\bar{x}$-axis. Since the curve β is $\bar{x}$-monotone, this ordered list of intersections provides a description of the curve β inside f, and indicates what are the neighboring faces that β crosses. We mark each of these additional faces by the point where β crosses out of f. We call each such point an *exit point*. See Figure 18(b).

Since f has already been constructed, we know for each exit point of β the line that contains it. Therefore we can construct each new face using ray shooting queries and proceed as above. We keep a data structure that describes all the faces of $\mathcal{A}(V^*)$ that have already been constructed so that we do not construct the same face twice.

The algorithm stops when we have identified all the intersection points of β with lines in V^*, and so we have also identified the *zone* of β in $\mathcal{A}(V^*)$, namely all the face of $\mathcal{A}(V^*)$ crossed by β.

Further details on the algorithm can be found in [16]. We summarize the algorithmic result in the following theorem.

Theorem 17 [14] *Let P be a non-degenerate polygon (possibly with holes) with n vertices, and such that any line crosses at most r edges of P. For any $\varepsilon > 0$ we can find a complete specification of the area bisectors of P in time $O(\kappa^{2/3}n^{2/3+\varepsilon} + \kappa\alpha(\kappa)\psi(r))$, where κ and $\psi(r)$ are as defined above, and $\alpha()$ is the functional inverse of Ackermann's function. If P is rectilinear, then the algorithms runs in time $O(\kappa^{2/3}n^{2/3+\varepsilon})$. The space required by the algorithm is $O(\kappa^{2/3}n^{2/3+\varepsilon})$.*

6.4 Moment equilibria and alignment plans

Recall from the beginning of this section that bisectors correspond to *force* equilibria of P in a squeeze field. For *total* equilibrium, the moment acting on the part has to be taken into account as well. In particular, not all of the force equilibrium configurations will be moment equilibria. For each maximal piece b of the bisector curve β there exists only a finite number of moment equilibria (we omit the proof here):

Claim 18 [14] *Let P be a polygon whose interior is connected. Let λ be a bisector of P that intersects r edges of P. There exist $O(r)$ lines λ' that are combinatorially equivalent to λ such that P is in total equilibrium when λ' coincides with the center line of a squeeze field.*

It follows that a squeeze field induces a finite number of total equilibria on a polygonal part P.

Much remains to be done. We have found that in practice, no non-symmetric part has more than 4 total equilibria (i.e., force and moment). Finding tight bounds on the number of total equilibria—that is, taking moment into account—will be critical in the future.

7 Conclusions

Table 1 summarizes fields and algorithms for manipulation tasks with programmable vector fields, and includes some additional recent results.

Less difficult tasks such as translation can be achieved with relatively simple fields and without any planning. More complex tasks such as centering or unique orienting require increasingly complex fields. However, planning complexity is e.g. higher for sequences of squeeze fields, and lower for the more complex combined radial + squeeze fields (for a thorough discussion of these fields see [19]). This illustrates a tradeoff between mechanical complexity (the dexterity and controllability of actuator arrray elements) and computational complexity (the algorithmic difficulty of synthesizing a strategy).

Universal feeder-orienter (UFO) devices. Our research leads to the question about the existence of a "universal feeder-orienter" (UFO) device that uniquely poses a part without the need of a clock, sensors, or programming [19, 20]. It was shown in [19] that every connected polygonal part P with n vertices has a finite number of stable orientation equilibria when P is placed into a squeeze field S. Based on this property we were able to generate manipulation strategies for unique part alignment. We then showed that by using a combined radial and squeeze field $R+\delta S$, the number of equilibria can be reduced to $O(kn)$. Using elliptic force fields $f(x,y) = (\alpha x, \beta y)$ such that $\alpha \neq \beta$ and $\alpha, \beta \neq 0$, this bound can be reduced to two [41, 40]. An "inertial" squeeze field $f(x,y) = (-\text{sign}(x)x^2, 0)$ uniquely orients a part modulo field symmetry π [21]. In a stable equilibrium, the part's major principal axis of inertia lines up with the squeeze line to minimize the second moment of inertia.

task	field(s)	complexity		
		fields	planning	plan steps
translate	constant	constant magnitude and direction	—	1
center	radial	constant magnitude, continuous directions	—	1
uniquely orient	seq. of squeezes	piecewise constant mag. and dir.	$O(k^2n^4)$	$O(kn^2)$
	inertial	smooth magnitude piecewise constant dir.	$O(1)$	$O(1)$
uniquely pose	seq. of radial+squeeze	piecewise continuous mag. and dir.	$O(k^2n^2)$	$O(kn)$
	elliptic	smooth mag. and dir.	$O(1)$	$O(1)$
	UFO	continuous mag. and dir.	—	1

Table 1: *Fields and algorithms for manipulation tasks with programmable vector fields.*

Does there exist a *universal field* that, for every part P, has only one unique equilibrium (up to part symmetry)? Such a field could be used to build a *universal parts feeder* [1] that uniquely positions a part without the need of a clock, sensors, or programming.

In [19], we proposed a combined radial and "gravitational" field $R + \delta G$ which might have this property. δ is a small positive constant, and G is defined as $G(x, y) = (0, -1)$. This device design is inspired by the "universal gripper" in [1]. Such a field could be obtained from a MEMS array that implements a unit radial force field. Instead of rectangular actuators in a regular grid, triangular actuators could be laid out in a polar-coordinate grid. The array could then be tilted slightly to obtain the gravity component. Hence such a device would be relatively easy to build. Alternatively, a resonating speaker, or a vibrating disk-shaped plate that is fixed at the center, might be used to create a radial force field. Extensive simulations show that for every part we have tried, one unique total equilibrium is always obtained. We are working toward a rigorous proof of this experimental observation.

Algorithmic MEMS In this paper, we have surveyed a family of ideas in Algorithmic MEMS, relating to programmable force fields. Naturally, in such a young and dynamic field, we expect a myriad of new geometric and algorithmic problems to emerge in the next few years.

Acknowledgements

We are very grateful to our collaborators and friends Noel MacDonald, John Suh, Greg Kovacs, Ken Goldberg, and Danny Halperin for sharing with us the research surveyed in this paper.

The authors would also like to thank Steve Glander, Robert Darling, Christopher Storment, and all the members of the Stanford Transducers Lab; the members of Noel MacDonald's research team; staff and users of the Cornell Nanofabrication Facility, the members of the Cornell Robotics and Vision Lab (CSRVL), the members of Donald Lab at Dartmouth, Lydia Kavraki for fruitful discussions, and Jean-Claude Latombe for his hospitality during our stay at the Stanford Robotics Laboratory.

Support for our research was provided in part by the NSF under grants no. IRI-8802390, IRI-9000532, IRI-9201699, IRI-9530785, IRI-9896020, and by a Presidential Young Investigator award to Bruce Donald, in part by NSF/ARPA SGER no. IRI-9403903, in part by an NSF Challenges in Computer and Information Science and Engineering (CISE) grant no. CDA-9726389, in part by an NSF CISE Postdoctoral Associateship to Karl Böhringer no. CDA-9705022, and in part by the AFOSR, the Mathematical Sciences Institute, Intel Corporation, and AT&T Bell Laboratories.

References

[1] T. L. Abell and M. Erdmann. A universal parts feeder, 1996. Personal communication / in preparation.

[2] S. Akella, W. H. Huang, K. M. Lynch, and M. T. Mason. Planar manipulation on a conveyor by a one joint robot with and without sensing. In *International Symposium of Robotics Research (ISRR)*, 1995.

[3] M. Ataka, A. Omodaka, and H. Fujita. A biomimetic micro motion system. In *Transducers — Digest Int. Conf. on Solid-State Sensors and Actuators*, pages 38–41, Pacifico, Yokohama, Japan, June 1993.

[4] D. Biegelsen, W. Jackson, A. Berlin, and P. Cheung. Air jet arrays for precision positional control of flexible media. In *Int. Conf. on Micromechatronics for Information And Precision Equipment (MIPE'97)*, Tokyo, Japan, July 1997.

[5] K.-F. Böhringer. Computational aspects of the design of micro-mechanical hinged structures. In *AAAI Fall Symposium Series — Design from Physical Principles, Working Notes*, Boston, MA, Oct. 1992. Abstract.

[6] K.-F. Böhringer. Application of computational topology to the design of microelectromechanical structures. In *Proceedings of the NSF Design and Manufacturing Systems Grantees Conference*, Charlotte, NC, Jan. 1993. National Science Foundation.

[7] K.-F. Böhringer. A computational approach to the design of micromechanical hinged structures. In *Proceedings of the ACM/SIGGRAPH Symposium on Solid Modeling and Applications*, Montréal, Québec, Canada, May 1993. ACM Press. Extended abstract.

[8] K.-F. Böhringer. Computational aspects of the design of microelectromechanical structures. In *ICRA Workshop on Geometric Algorithms for Manufacturing*, Atlanta, Georgia, May 1993. IEEE.

[9] K.-F. Böhringer. *Programmable Force Fields for Distributed Manipulation, and Their Implementation Using Microfabricated Actuator Arrays.* PhD thesis, Cornell University, Department of Computer Science, Ithaca, NY 14853, Aug. 1997.

[10] K.-F. Böhringer, V. Bhatt, B. R. Donald, and K. Y. Goldberg. Sensorless manipulation using transverse vibrations of a plate. *Algorithmica*, 1998. To appear in Special Issue on Algorithmic Foundations of Robotics.

[11] K.-F. Böhringer, V. Bhatt, and K. Y. Goldberg. Sensorless manipulation using transverse vibrations of a plate. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 1989–1996, Nagoya, Japan, May 1995.

[12] K.-F. Böhringer, R. G. Brown, B. R. Donald, J. S. Jennings, and D. Rus. Distributed robotic manipulation: Experiments in minimalism. In O. Khatib et al., editor, Experimental Robotics IV, *Lecture Notes in Control and Information Sciences 223*, pages 11–25. Springer Verlag, Berlin, 1997.

[13] K.-F. Böhringer, M. B. Cohn, K. Goldberg, R. Howe, and A. Pisano. Electrostatic self-assembly aided by ultrasonic vibration. In *AVS 44th National Symposium*, San Jose, CA, Oct. 1997.

[14] K.-F. Böhringer, B. R. Donald, and D. Halperin. The area bisectors of a polygon and force equilibria in programmable vector fields. In *13th ACM Symposium on Computational Geometry*, Nice, France, June 1997.

[15] K.-F. Böhringer, B. R. Donald, and D. Halperin. On the area bisectors of a polygon. In *Second CGC Workshop on Computational Geometry*, Durham, NC, Oct. 1997. Extended Abstract.

[16] K.-F. Böhringer, B. R. Donald, and D. Halperin. On the area bisectors of a polygon. *Discrete and Computational Geometry*, 1998. To appear.

[17] K.-F. Böhringer, B. R. Donald, and N. C. MacDonald. Single-crystal silicon actuator arrays for micro manipulation tasks. In *Proc. IEEE Workshop on Micro Electro Mechanical Systems (MEMS)*, pages 7–12, San Diego, CA, Feb. 1996.

[18] K.-F. Böhringer, B. R. Donald, and N. C. MacDonald. What programmable vector fields can (and cannot) do: Vector field algorithms for MEMS arrays and vibratory parts feeders. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 822–829, Minneapolis, MN, Apr. 1996.

[19] K.-F. Böhringer, B. R. Donald, and N. C. MacDonald. Upper and lower bounds for programmable vector fields with applications to MEMS and vibratory plate parts feeders. In J.-P. Laumond and M. Overmars, editors, *Algorithms for Robotic Motion and Manipulation*, pages 255–276. A. K. Peters, Wellesley, MA 02181, 1997.

[20] K.-F. Böhringer, B. R. Donald, and N. C. MacDonald. Programmable vector fields for distributed manipulation, with applications to MEMS actuator arrays and

vibratory parts feeders. *Int. Journal of Robotics Research*, 1998. To appear.

[21] K.-F. Böhringer, B. R. Donald, N. C. MacDonald, G. T. A. Kovacs, and J. W. Suh. Computational methods for design and control of MEMS micromanipulator arrays. *Computer Science and Engineering*, pages 17–29, January – March 1997.

[22] K.-F. Böhringer, B. R. Donald, R. Mihailovich, and N. C. MacDonald. A geometric theory of manipulation and control for microfabricated actuator arrays. Technical Report 93–87, Cornell University, Mathematical Sciences Institute, Ithaca, NY 14853, Nov. 1993.

[23] K.-F. Böhringer, B. R. Donald, R. Mihailovich, and N. C. MacDonald. Sensorless manipulation using massively parallel microfabricated actuator arrays. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 826–833, San Diego, CA, May 1994.

[24] K.-F. Böhringer, B. R. Donald, R. Mihailovich, and N. C. MacDonald. A theory of manipulation and control for microfabricated actuator arrays. In *Proc. IEEE Workshop on Micro Electro Mechanical Systems (MEMS)*, pages 102–107, Oiso, Japan, Jan. 1994.

[25] K.-F. Böhringer, K. Goldberg, M. B. Cohn, R. Howe, and A. Pisano. Parallel microassembly with electrostatic force fields. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, Leuven, Belgium, May 1998.

[26] K.-F. Böhringer, J. W. Suh, B. R. Donald, and G. T. A. Kovacs. Vector fields for task-level distributed manipulation: Experiments with organic micro actuator arrays. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 1779–1786, Albuquerque, New Mexico, Apr. 1997.

[27] P. Cheung, A. Berlin, D. Biegelsen, and W. Jackson. Batch fabrication of pneumatic valve arrays by combining mems with printed circuit board technology. In *Proc. Symposium on Micro-Mechanical Systems, ASME International Mechanical Engineering Congress and Exhibition*, pages 16–21, Dallas, TX, Nov. 1997.

[28] M. Coutinho and P. Will. The intelligent motion surface: a hardware/software tool for the assembly of meso-scale devices. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, Albuquerque, New Mexico, Apr. 1997.

[29] M. Coutinho and P. Will. Open loop dynamic control of parts moving on an intelligent motion surface along paths composed of straight line segments. In *IEEE Control and Decision Conference*, USA, 1997. Submitted for review.

[30] M. Coutinho and P. Will. Using dynamic vector force fields to manipulate parts on an intelligent motion surface. In *IEEE International Symposium on Automation and Task Planning*, Los Angeles, CA, 1997.

[31] M. Díaz and J. O'Rourke. Ham-sandwich sectioning of polygons. In *Proc. 2nd Canadian Conference on Computational Geometry*, pages 98–101, Ottawa, 1990.

[32] B. R. Donald, J. Jennings, and D. Rus. Information invariants for distributed manipulation. In K. Goldberg, D. Halperin, J.-C. Latombe, and R. Wilson, editors, *International Workshop on Algorithmic Foundations of Robotics (WAFR)*, pages 431–459, Wellesley, MA, 1995. A. K. Peters.

[33] B. R. Donald and D. K. Pai. The motion of compliantly connected rigid bodies in contact. *Int. Journal of Robotics Research*, July 1991.

[34] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*, volume 10 of *EATCS Monographs on Theoretical Computer Science*. Springer Verlag, Heidelberg, Germany, 1987.

[35] M. A. Erdmann. An exploration of nonprehensile two-palm manipulation: Planning and execution. Technical report, Carnegie Mellon University, Pittsburgh, PA, 1996.

[36] J. D. Foley, A. Van Dam, Feiner, and Hughes. *Computer Graphics: Principles and Practice*. Addison Wesley, 2nd edition, 1996.

[37] H. Fujita. Group work of microactuators. In *International Advanced Robot Program Workshop on Micromachine Technologies and Systems*, pages 24–31, Tokyo, Japan, Oct. 1993.

[38] K. Y. Goldberg. Orienting polygonal parts without sensing. *Algorithmica*, 10(2/3/4):201–225, August/September/October 1993.

[39] B. Joffe. Manipulation and identification of objects by magnetic forces. In *Proc. International Symposium on Magnetic Suspension Technology, NASA Conference Publication 3152 part 2*, pages 617–639, Langley, VA, Aug. 1991.

[40] L. Kavraki. Part orientation with programmable vector fields: Two stable equilibria for most parts. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, Albuquerque, New Mexico, Apr. 1997.

[41] L. E. Kavraki. On the number of equilibrium placements of mass distributions in elliptic potential fields. Technical Report STAN-CS-TR-95-1559, Department of Computer Science, Stanford University, Stanford, CA 94305, 1995.

[42] O. Khatib. Real time obstacle avoidance for manipulators and mobile robots. *Int. Journal of Robotics Research*, 5(1):90–99, Spring 1986.

[43] D. E. Koditschek and E. Rimon. Robot navigation functions on manifolds with boundary. *Advances in Applied Mathematics*, 1988.

[44] K. Kotay, D. Rus, M. Vona, and C. McGray. The self-reconfigurable robotic molecule. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, Belgium, May 1998.

[45] W. Liu and P. Will. Parts manipulation on an intelligent motion surface. In *IEEE/RSJ Int. Workshop on Intelligent Robots & Systems (IROS)*, Pittsburgh, PA, 1995.

[46] J. E. Luntz and W. Messner. A distributed control system for flexible materials handling. *IEEE Control Systems*, 17(1), Feb. 1997.

[47] J. E. Luntz, W. Messner, and H. Choset. Parcel manipulation and dynamics with a distributed actuator array: The virtual vehicle. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 1541–1546, Albuquerque, New Mexico, Apr. 1997.

[48] R. E. Mihailovich and N. C. MacDonald. Dissipation measurements of vacuum-operated single-crystal silicon resonators. *Sensors and Actuators*, 1996.

[49] R. E. Mihailovich, Z. L. Zhang, K. A. Shaw, and N. C. MacDonald. Single-crystal silicon torsional resonators. In *Proc. IEEE Workshop on Micro Electro Mechanical Systems (MEMS)*, pages 155–160, Fort Lauderdale, FL, Feb. 1993.

[50] K. S. J. Pister, R. Fearing, and R. Howe. A planar air levitated electrostatic actuator system. In *Proc. IEEE Workshop on Micro Electro Mechanical Systems (MEMS)*, pages 67–71, Napa Valley, California, Feb. 1990.

[51] R. Prasad, K.-F. Böhringer, and N. C. MacDonald. Design, fabrication, and characterization of SCS latching snap fasteners for micro assembly. In *Proceedings of the ASME International Mechanical Engineering Congress and Exposition (IMECE)*, San Francisco, California, Nov. 1995.

[52] M. H. Raibert, J. K. Hodgins, R. R. Playter, and R. P. Ringrose. Animation of legged maneuvers: jumps, somersaults, and gait transitions. *Journal of the Robotics Society of Japan*, 11(3):333–341, 1993.

[53] J. Reif and H. Wang. Social potential fields: A distributed behavioral control for autonoomous robots. In K. Goldberg, D. Halperin, J.-C. Latombe, and R. Wilson, editors, *International Workshop on Algorithmic Foundations of Robotics (WAFR)*, pages 431–459. A. K. Peters, Wellesley, MA, 1995.

[54] D. Reznik, S. Brown, and J. F. Canny. Dynamic simulation as a design tool for a microactuator array. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, Albuquerque, NM, Apr. 1997.

[55] D. Reznik, J. F. Canny, and K. Y. Goldberg. Analysis of part motion on a longitudinally vibrating plate. In *IEEE/RSJ Int. Workshop on Intelligent Robots & Systems (IROS)*, Grenoble, France, Sept. 1997.

[56] E. Rimon and D. Koditschek. Exact robot navigation using artificial potential functions. *IEEE Transactions on Robotics and Automation*, 8(5), October 1992.

[57] K. A. Shaw and N. C. MacDonald. Integrating SCREAM micromechanical devices with integrated circuits. In *Proc. IEEE Workshop on Micro Electro Mechanical Systems (MEMS)*, San Diego, CA, Feb. 1996.

[58] K. A. Shaw, Z. L. Zhang, and N. C. MacDonald. SCREAM I: A single mask, single-crystal silicon process for microelectromechanical structures. In *Transducers — Digest Int. Conf. on Solid-State Sensors and Actuators*, Pacifico, Yokohama, Japan, June 1993.

[59] J. W. Suh, S. F. Glander, R. B. Darling, C. W. Storment, and G. T. A. Kovacs. Combined organic thermal and electrostatic omnidirectional ciliary microactuator array for object positioning and inspection. In *Proc. Solid State Sensor and Actuator Workshop*, Hilton Head, NC, June 1996.

[60] Z. L. Zhang and N. C. MacDonald. An RIE process for submicron, silicon electromechanical structures. *Journal of Micromechanics and Microengineering*, 2(1):31–38, Mar. 1992.

[61] N. B. Zumel and M. A. Erdmann. Nonprehensile two palm manipulation with non-equilibrium transitions between stable states. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, Minneapolis, MN, Apr. 1996.

Universal Part Manipulation in the Plane with a Single Horizontally-Vibrating Plate

Dan S. Reznik, *UC-Berkeley, Berkeley, CA, USA*
John F. Canny, *UC-Berkeley, Berkeley, CA, USA*

Active surfaces can manipulate (i.e., translate, rotate) one or more parts in the plane by "displaying" arbitrary forces at a number of points. Many designs have been proposed based on a massively-parallel array of microactuators. For such "pixel-oriented" devices, a large density of microactuators per unit area is required for acceptable force field programmability; the large number and small size of moving parts renders these devices difficult to fabricate, test, and maintain. Here we take a minimalist approach and look for a mechanically simpler, sturdier design, which retains the same manipulation dexterity. Surprisingly, we show that a rigid flat plate is just such a device. Namely, a closed, horizontal motion of the plate can be computed which yields arbitrary frictional forces (averaged over the motion) at one or more points, so that one or more parts can be translated and/or rotated simultaneously and independently. Analysis for both part feeding and part manipulation cases, along with dynamic simulation and experimental results with a prototype of the device are included.

1 Introduction

Dextrous manipulation of parts in the plane is of key importance in the assembly, sorting, and feeding of industrial parts. Traditional pick-and-place methods are not suitable for small scale, many-part, and/or parallel manipulation applications due to end-effector clutter, planning/control complexity, and execution latencies. Research on active surfaces has addressed this issue by considering devices which incorporate the actuation mechanism into the surface on which parts rest.

A popular design concept is that of a massively-parallel array of micro-actuators. An arbitrary force can be generated at a specific location by addressing the appropriate "pixel", or group of actuators. Alternatively, their massive parallelism can be used to synthesize "sensorless" fields which automatically locate/orient parts [3, 9]. Existing devices include arrays of thermally-actuated cilia [21], micro-electro-mechanical resonators [3], orientable rollers [13], air-flow nozzles [10], and electric/magnetic elements [15, 12]. The large number and small size of moving elements in array-like devices pose fabrication difficulties and reduce device robustness.

Driven by the concept of minimalism in robotics [5], we search for a mechanically simpler and sturdier device which retains full manipulation dexterity. Our main contribution is to show that a single horizontally-vibrating rigid, flat plate is such a device. Our proposed device concept is shown in Figure 1. A rigid, flat plate is suspended parallel to ground by the shafts of four linear-travel (e.g., voice coil) motors. These can command arbitrary instantaneous accelerations (within bounds), along the plate's three planar degrees of freedom. Parts placed on the plate's surface are manipulated via sliding frictional forces.

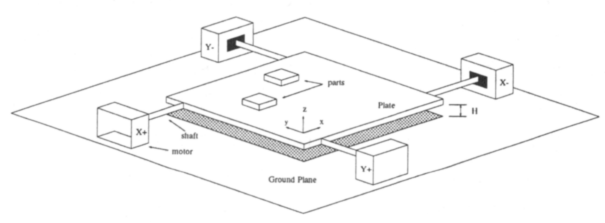


Figure 1: *Model of the proposed device – a rigid, flat horizontal plate and an actuation mechanism (made up of four linear-travel motors). The plate is constrained to vibrate horizontally. Two parts are shown resting on the plate.*

Our main result is to show that given desired forces at one or more part locations, a closed, horizontal motion of the plate – a vibration cycle – can be computed which yields the desired forces (averaged over the entire motion) independently and simultaneously at each point. At the heart of this result is a sequence of five observations: (i) the magnitude of dynamic friction is constant and independent on the relative velocity; (ii) any instantaneous planar rigid motion of the plate corresponds to a rotation about a point; (iii) such a motion gives rise to a frictional force field (assuming dynamic friction at every point) of constant magnitude vectors, oriented tangentially with respect to the center of rotation; (iv) the family of fields of type (iii) is not closed under addition (unlike the family of rigid velocity fields), so that (v) the sum of N frictional force fields is N-dimensional (compare this to a maximum dimension of 3 for rigid velocity fields).

We propose two methods to computing the aforementioned closed motion. In the first method, the plate's motion is composed of a finite sequence of small rotations about a known set of points (e.g., chosen conveniently given the actuation kinematics). Assume the entire sequence is executed fast enough and/or friction at the interface is rather small so that parts lying on the plate's surface move little throughout the sequence. The average force applied to a part during the entire motion is the sum of frictional forces corresponding to each rotation. Owing to (v), the problem of generating desired average frictional forces under each part becomes that of solving for the duration of each rotation (i.e., its "intensity", or "weight"), which under the present method reduces to inverting a linear system.

Switching abruptly between rotations can lead to resonance in the plate's suspension. In the second method, vibration bandwidth is reduced by making the plate describe a smooth, closed motion involving its 3 degrees of freedom simultaneously. We fix the motion along each degree of freedom to be a sinusoid plus a double frequency sinusoid, each with controllable phase and amplitude, for a total of 12 free parameters. The map from parameter space to an average force at a given point is highly convoluted and not-likely to be invertible; Given (v), above, we can expect the image of this map to be high-dimensional (at most 12-dimensional). By using iterative, numeric inversion methods we can compute the parameters which yield the desired forces.

1.1 Related work

The quintessential instance of a vibrations-based part manipulation is the vibratory bowl feeder [4]. In this case, off-plane vibration is used to feed parts along a track internal to the feeder. The APOS feeder [4] uses vibration to aid in massively parallel passive part entrapment applications. Designing feeder/part shapes for optimum entrapment has been the subject of some recent research [11]. More exotic experiments with vibration in the context of part manipulation include resonance-based manipulation [22] and part localization through node-formation in an oscillating surface [2].

Much of the work described here is a continuation of our research in exploiting horizontal vibrations for the purpose of part feeding and manipulation. In [19, 17] we describe a method for part feeding based on a 1d horizontally-vibrating plate. In [18] these ideas are generalized to the 2d manipulation case; the method of sequencing rotations is first introduced there; here this method is further described and contrasted with a second, lower-bandwidth method.

The remainder of this paper is organized as follows: In Section 2 we present manipulation preliminaries including plate kinematics, the concept of force field averaging, and the non-additivity of frictional force fields. In Section 3 we describe the parallel manipulation task along with two proposed methods (sequencing rotations and sinusoidal controls) for computing the required closed plate motions. In Section 4 we present dynamic simulation results for two parallel manipulation tasks. In Section 5 we describe some experiments we have performed with a prototype of the device. Conclusions are presented in Section 6.

2 Manipulation Preliminaries

2.1 Coulomb friction in 2d

Let ν_s and ν_p denote, respectively, the instantaneous surface (i.e., the plate) and part velocities, relative to an inertial frame XY, Figure 2. Assuming the surfaces are sliding (i.e., the relative acceleration is sufficiently high), the frictional force f_{fric} will be of constant magnitude μmg and point opposite to the part's motion relative to the surface, i.e.:

$$f_{fric,2d} = \mu mg \frac{(\nu_s - \nu_p)}{||\nu_s - \nu_p||} \tag{1}$$

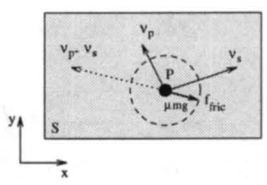


Figure 2: *Coulomb friction in 2d: the frictional force lies on a circle of radius μmg centered on the part (P), and points opposite to P's velocity relative S, i.e., $\nu_p - \nu_s$.*

2.2 Force field averaging

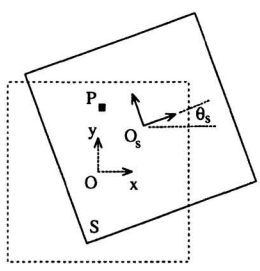


Figure 3: *Dotted square: the plate S at its home position, where its center coincides with the origin O. Solid square: S with its center translated by O_s and rotated by θ_s. A part is shown lying on S at point P.*

The surface S is constrained to move in its own plane. Let $O_s = (x_s, y_s)$, and θ_s denote S's translation and orientation relative to XY, Figure 3. Consider a smooth motion $S(t) = [O_s(t), \theta_s(t)]^t$ of the surface which is *closed*, i.e., $S(0) = S(T)$, for some $T > 0$. The surface's tangential velocity ν_s at a point P is:

$$\nu_s(P,t) = \dot{O}_s(t) + \dot{\theta}_s(t) \times [P - O_s(t)] \tag{2}$$

Let us consider points P much larger in magnitude than the surface's maximum translation, i.e., $||P|| \gg \max_t ||O_s(t)||$, so that $P - O_s(t) \simeq P$. Equation 2 reduces to:

$$\begin{aligned} \nu_s(P,t) &= \dot{O}_s(t) + \dot{\theta}_s(t) \times P \\ &= [u_1(t) - y u_3(t), u_2(t) + x u_3(t)]^t \end{aligned} \tag{3}$$

Where symbols $u_1(t)$, $u_2(t)$, and $u_3(t)$ represent $\dot{x}_s(t)$, $\dot{y}_s(t)$, and $\dot{\theta}_s(t)$, respectively. Note $S(t)$ closed implies $\int_0^T u_i(t)\, dt = 0,\ i = 1 \ldots 3$.

Assumption 1 *P is always sliding on S – surface accelerations at P give rise to inertial forces above the threshold of static friction [14], for most $t \in [0, T)$.*

Note: $S(t)$ closed implies that $\dot{\nu}_s(t)$ will have one or more zero-crossings within $[0, T)$ during which the part will tend to stick to the surface; this is why Assumption 1 can only be true for "most" t.

If the part's velocity $\nu_p(t)$ is negligible compared to $\nu_s(P,t)$ for most $t \in [0, T)$, Equation 1 reduces to:

$$f_{fric}(P,t) = \mu mg \frac{\nu_s(P,t)}{||\nu_s(P,t)||} \tag{4}$$

An upper bound for the part's maximum displacement in T seconds given frictional forces is $||\Delta P||_{max} = \mu g T^2$. Let us assume this quantity negligible with respect to $||P||$ so that the part's position can be considered nearly constant within one cycle. The average frictional force $\bar{f}_{2d}$ applied to the part becomes simply the integral of the instantaneous frictional forces at a P divided by the length of the motion:

$$\bar{f}_{2d}(P) = \frac{1}{T} \int_0^T f_{fric}(P,t)\, dt \tag{5}$$

2.3 Instantaneous force fields

Chasles' theorem [6] in planar kinematics states that any instantaneous rigid velocity $[u_1, u_2, u_3](t)$ corresponds to a rigid rotation about a center $C = (c_x, c_y)$ at angular velocity w where:

$$\begin{aligned} c_x &= -u_2/u_3 \\ c_y &= +u_1/u_3 \\ w &= u_3 \end{aligned}$$

If the $u_i(t)$'s are directly controllable, we can chose them so S executes a rigid rotation about a fixed center C. Alternatively, the actuation kinematics maybe such that rotating the surface about one (or more) fixed points is a simple procedure. Consider an instantaneous rotation $\theta(t)$ of S about a point C, $\theta(0) = \theta(T)$. With C stationary, S's instantaneous tangential velocity $\nu_s(t)$ at point $P = (x, y)^t$ is $\dot{\theta}(t) \times (P - C)$, or:

$$\nu_s(t) = \dot{\theta}(t) \begin{bmatrix} c_y - y \\ x - c_x \end{bmatrix} \tag{6}$$

Define the unit tangential vector $\hat{\tau} = (P - C)/||P - C||$, From Equation 6, the direction of f_{fric} along $\hat{\tau}$ is given by $sgn(\dot{\theta})$, yielding the force field illustrated on Figure 4. So f_{fric} can be expressed succinctly as:

$$f_{fric}(t) = \mu mg \; sgn[\dot{\theta}(t)]\hat{\tau} \tag{7}$$

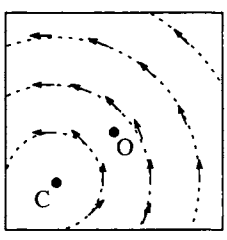


Figure 4: *A rigid motion of S about C yields a frictional force field of tangential vectors of constant magnitude.*

2.4 Non-additivity of force fields

Let $\psi_{k,C}(P)$ and $\phi_{k,C}(P)$ represent, respectively, the value of arbitrary velocity and average force fields at a point P, where k and C denote the fields' "strength" (a linear scaling parameter, explained below) and center of rotation, respectively. Algebraically:

$$\begin{aligned} \psi_{k,C}(P) &= k \; \mu mg \; R_{90}(P - C) \\ \phi_{k,C}(P) &= k \; \mu mg \; R_{90} \frac{P - C}{||P - C||} \\ R_{90} &= \text{rotate-by-90}^o\text{-matrix} \end{aligned} \tag{8}$$

Let Ψ (resp. Φ) denote the 3-dimensional family of ψ's (resp. ϕ's):

$$\begin{aligned} \Psi &= \{\psi_{k,C} | k \in \mathcal{R}, C \in \mathcal{R}^2\} \\ \Phi &= \{\phi_{k,C} | k \in \mathcal{R}, C \in \mathcal{R}^2\} \end{aligned}$$

It can be shown that for $\psi_1 = \psi_{k_1,C_1}$ and $\psi_2 = \psi_{k_2,C_2}$, $\psi_1 + \psi_2 = \psi_{k_3,C_3}$ with:

$$\begin{aligned} k_3 &= k_1 + k_2 \\ C_3 &= \frac{k_1 C_1 + k_2 C_2}{k_1 + k_2} \end{aligned}$$

i.e., Ψ is closed under addition. This property is however not true for Φ, namely:

Observation 1 *Φ is <u>not</u> closed under addition: Let $\phi_1, \phi_2 \in \Phi$. Then $\phi_1 + \phi_2 \notin \Phi$. Namely, $\neg\exists k_3, C_3$ such that $\phi_1 + \phi_2 = \phi_{k_3,C_3}$.*

In other words, the sum of two force fields does not lie in the original space, i.e., the resulting field will be part of a higher-dimensional family of vector fields, as depicted in Figure 5.

3 The Parallel Manipulation Task

Consider M parts distributed over S at known locations P_i, $i = 1 \ldots M$. Let $f_i(k)$ be a desired average

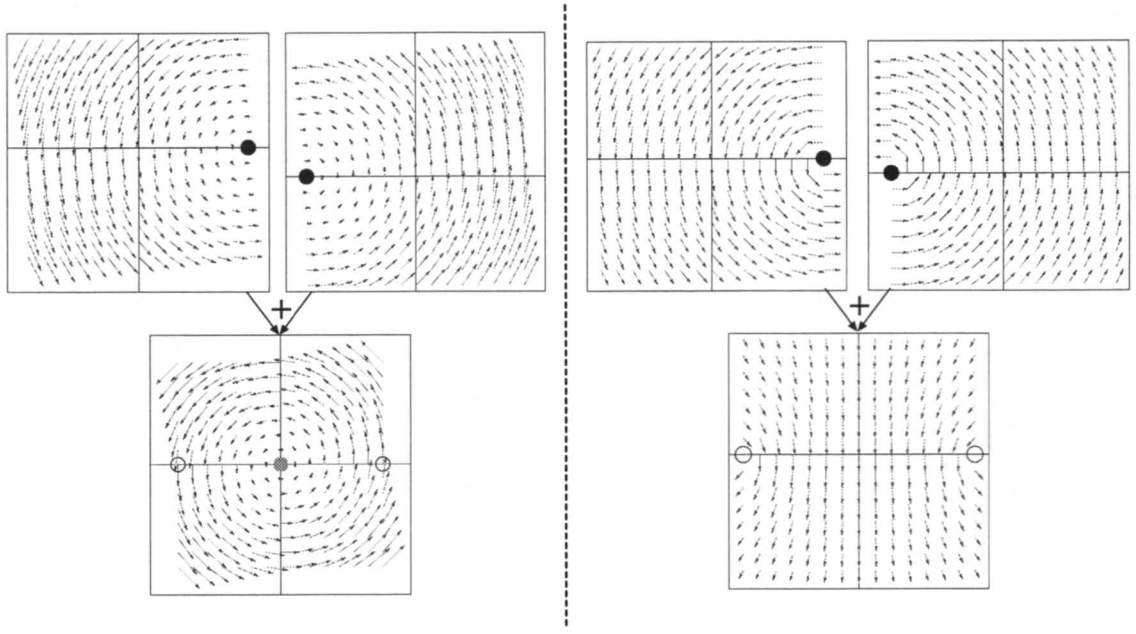


Figure 5: *Left: the sum of two velocity fields is a velocity field. Right: the sum of two frictional force fields is not in the same family.*

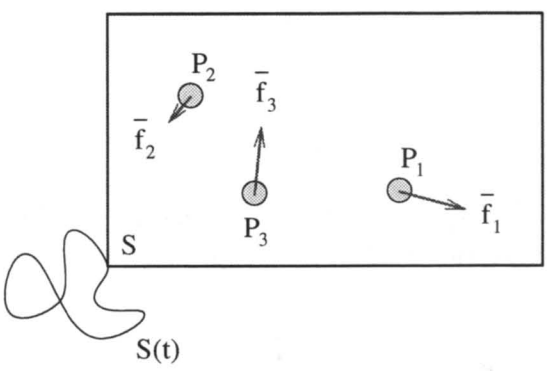


Figure 6: *The parallel manipulation task consists in computing a closed motion $S(t)$ for the surface S which can generate desired frictional forces $\bar{f}_i$ (averaged over the motion) at each of the parts' locations P_i.*

force for part i at a given instant. The goal is to compute a closed motion $S(t)$ which yields $\bar{f}_{2d}(P_i) = f_i$, $\forall i$, as shown in Figure 6.

The setup envisioned is as shown in Figure 7. A snapshot of S is obtained with a camera and sent to the computer. Image processing algorithms are used to determine the P_i's. From the state within a manipulation task, desired forces f_i are computed for each part; this is followed by a solving phase in which a closed motion $S(t)$ is computed for the surface; this motion is then sent out as commands to motors driving the surface, and the loop closes. Feedback helps in compensating for deviations from the desired type of manipulation, caused by modeling errors, unevenness in friction, control inaccuracies, etc.

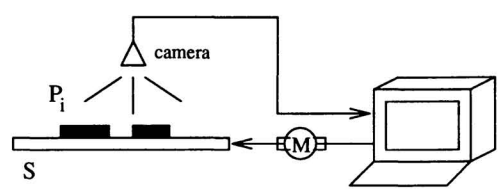


Figure 7: *Closed-loop manipulation: a camera sends images to a computer, which sends commands to a motor which actuates the surface which moves the parts.*

In what follows we present two methods to solving for a closed motion $S(t)$ which yields the $\bar{f}_i$'s.

3.1 Method A: sequencing rotations

Consider N consecutive small-angle rotations of the surface, $N \geq 2M$, each about a distinct point C_j, $j = 1 \ldots N$. Let each such duration's rotation be scaled by an independent k_j, such that $\sum k_j = T$. Let $\phi_j = \mu mg\ \phi_{k_j, C_j}$.

We assume that parts move negligibly during the entire motion, or, equivalently, that the ϕ_j's are constant in a sufficiently large neighborhood of the P_i's. This renders true the following observation.

Observation 2 *Superposition: The net average force $\bar{f}_i$ felt by P_i after N rotations is the weighted sum of the ϕ_j's at the parts' starting positions P_i:*

$$\bar{f}_i = \frac{1}{T} \sum_{j=1}^{N} \phi_j(P_i), i = 1 \ldots M \tag{9}$$

Assume the C_j's are pre-determined (e.g., by the manipulation kinematics), and that the $\bar{f}_i$ are given by the manipulation task. Combining Equations 8 and 9 reduces the parallel part manipulation problem to finding a solution vector $K = (k_1 \ldots k_N)$ which satisfies the linear system $A.K = b$, with:

$$\begin{aligned} A_{ij} &= \begin{cases} -a(i,j) & , \quad i = 1, \cdots M \\ b(i-M, j) & , \quad i = M+1, \cdots 2M \end{cases} \\ & \quad j = 1 \cdots N \qquad (10) \\ b_i &= \frac{N}{\mu mg} \begin{cases} \bar{f}_i^x & , \quad i = 1, \cdots M \\ \bar{f}_{i-M}^y & , \quad i = M+1, \cdots 2M \end{cases} \\ \text{and:} & \\ a(i,j) &= (p_i^y - c_j^y)/r_{ij} \\ b(i,j) &= (p_i^x - c_j^x)/r_{ij} \\ r_{ij} &= ||P_i - C_j|| \end{aligned}$$

3.1.1 Force specification example

Consider a rectangular surface rotatable about its 4 corners $C_1 = (-1, -0.5)^t, C_2 = (1, -.5)^t, C_3 = (1, .5)^t, C_4 = (-1, .5)^t$. Consider two parts $P_1 = (.5, 0)^t$ and $P_2 = (-.75, .25)$. Let $\mu = .25$, $g = 10$, $m = .04$ ($\mu mg = .1$) (note: all in SI units). Let $\bar{f}_1 = (.01, .01)^t$ and $\bar{f}_2 = (-.01, 0)^t$. The goal is to compute a 4-long vector $K = (k_1, k_2, k_3, k4)$ and execute 4 rotations consecutively about each of C_i's (Figure 8 with each such motion "scaled", in duration, by the computed k_j. The components of the 4×4 linear system become:

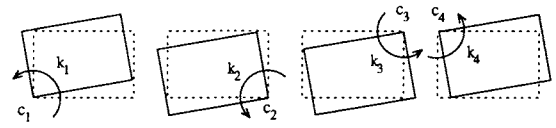


Figure 8: *The surface executes 4 rotations about each of its corners, say, in CCW order. The duration of each rotation is scaled by a computed k_j.*

$$\begin{aligned} A_{4\times4} &= \begin{bmatrix} -.316 & -.707 & .707 & .316 \\ -.949 & -.394 & .141 & .707 \\ .949 & -.707 & -.707 & .949 \\ .316 & -.919 & -.990 & .707 \end{bmatrix} \\ b &= (.4, -.4, .4, 0)^t \end{aligned}$$

Inverting A and compute $k = A^{-1}.b$ to obtain:

$$K = (.58, -.39, .48, -.095)^t$$

As shown in Figure 9, the resulting force field are as desired at the part's locations.

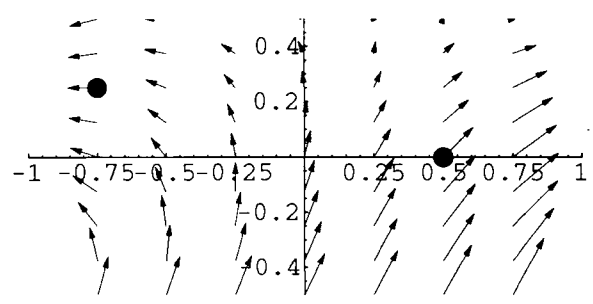


Figure 9: *The average force vectors corresponding to a series of 4 rotations about S's corners. The field points in the desired directions at the location of each part, shown as black disks.*

3.1.2 Rotating parts

The *curl* $\nabla\times f$ of a vector field $f = (f_x, f_y)^t$ is the scalar function $\frac{d}{dx}f_y - \frac{d}{dy}f_x$. It measures the rotation of an infinitesimal area element flowing along the field [20]. A simple expression yields the curl of a rotation-induced force field $\phi_{k,C}$:

$$\nabla\times\phi_{k,C}(P) \quad = \quad k\frac{\mu mg}{||P-C||} \qquad (11)$$

While the curl is measured at a point and the part's footprint will be over an area, one way to induce rotation is to specify a non-zero curl under the part's center of mass, while simultaneously specifying a zero-force for that point. Since the generated fields are continuous, this will create a "whirlpool" centered at the part.

The curl is a linear operator, so for a set of vector fields ϕ_i, $\nabla\times\sum_i \phi_i = \sum_i \nabla\times\phi_i$. So the curl of the net average force field (Equation 9) can be written as:

$$\nabla\times f_i(P_i) \quad = \quad \frac{\mu mg}{N}\sum_{j=1}^{N}\frac{k_j}{||P_i - C_j||} \qquad (12)$$

The above amounts to a single linear constraint, i.e., it becomes a row of the A matrix used in Equation 10. So besides specifying zero-part motion at M locations (amounting to $2M$ constraints), we can specify the curl, simultaneously, at Q locations (Q constraints). To avoid over-constraining the system, $2M + Q \leq N$, the number of rotation centers.

3.1.3 Curl specification example

Consider a single part P_1 placed at location $(.5, 0)$. We want to specify $\bar{f}_1 = (0,0)^t$ (2 constraints) and the curl at P_1 (1 constraint) to be $\delta_1 = -.05$ (negative curl means the field should be turning clockwise around that point). Since only 3 linear constraints are specified, there in no need for a fourth rotation about C_4. The resulting 3×3 linear system becomes:

$$\begin{bmatrix} -s_{11} & -s_{12} & -s_{13} \\ c_{11} & c_{12} & c_{13} \\ r_{11}^{-1} & r_{12}^{-1} & r_{13}^{-1} \end{bmatrix} \begin{bmatrix} k_1 \\ k_2 \\ k_3 \end{bmatrix} = \frac{N}{\mu mg}\begin{bmatrix} f_1^x \\ f_1^y \\ \delta_1 \end{bmatrix}$$

Which yields:

$$\begin{bmatrix} -.316 & -.707 & .707 \\ .949 & -.707 & -.707 \\ .633 & 1.41 & 1.41 \end{bmatrix} \cdot \begin{bmatrix} k_1 \\ k_2 \\ k_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -1.5 \end{bmatrix}$$

Solving the above yields $K = (-.59, -.27, -.53)$. The resulting field, illustrated in Figure 10, contains a CW whirlpool at the part's position, as desired. The integral of the forces under the part generates negative torque, resulting in CW rotation.

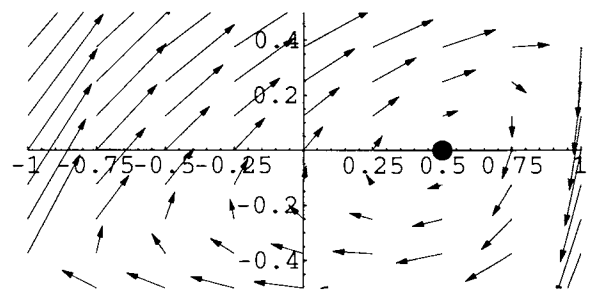


Figure 10: *Average force field obtained after 3 rotations about C_1, C_2, and C_3 in sequence. Zero force and negative curl are desired at the part's location* $(0.5, 0)$. *The result is a CW whirlpool centered on the part.*

3.1.4 Ordering rotations

Superposition assumes the fields are slowly-varying around the parts' neighborhoods. However, field variability increases as one approaches the center of rotation (conversely, at infinity, an average force field looks like a constant field). Furthermore, field variability is proportional to the scaling k_j of a particular rotation. For every center of rotation C_j, define ρ_j, the center's *rotation priority* as follows:

$$\rho_j = \frac{k_j}{\min_i ||P_i - C_j||}$$

A heuristic approach to reduce rotation-ordering effects which violate the superposition assumption is to rotate about the C_j's in *descending* order of rotation priority, ρ_j, above. In Section 3.1.1, $(\rho_1 \cdots \rho_4)^t = (.16, .031, .43, .34)^t$, so the "optimal" rotation ordering is C_3, C_4, C_1, and C_2.

A more rigorous approach to understand order-dependent errors would have to take into account the component of each motion lying on the *lie-bracket* $[\phi_1, \phi_2]$ of two consecutive rotations [16].

3.1.5 Ill-conditioning

The inverse of A (Equation 10) becomes ill-conditioned when at least one the following is true:

- Distinct field values are specified for nearby locations (e.g., for parts approaching each other).
- The field value is specified too far from the centers of rotation.

The first case, specifying distinct field values in a small neighborhood forces high derivatives in the resulting (average) field. Since the latter is the sum of a set of smooth functions ϕ_i, this de-stabilizes the inversion.

For the second case, consider a point P very far from a center of rotation C. The resulting force field around P will be nearly constant and perpendicular to $P - C$ (tangential lines become straight lines). By superposition, the same effect will happen for a sequence of rotations about several distinct centers. Since the family of constant fields has rank 2 and is closed under addition, A's rank will be limited to 2 in that region.

Ill-conditioning yields solution vectors K with unacceptably large entries. This can be addressed by introducing *redundancy*, i.e., additional centers of rotation, so that A becomes rectangular. Singular-value decomposition (SVD) is used to find the smallest-magnitude K, satisfying the constraints [7]. Conveniently, this algorithm allows us to place maximum bounds on the entries of A^{-1} [7].

Another method of dealing with ill-conditioning is to change the location of centers of rotations. If a given set of rotation centers produces an unstable matrix inversion, an alternative set can be attempted. Typically, the set of points the surface can rotate about will be a function of the particular motor/suspension kinematics.

3.2 Method B: sinusoidal controls

The previous manipulation method requires that the surface shift abruptly from one center of rotation to another. From a Fourier standpoint, the motion will contain significant harmonics, possibly causing resonance. One partial solution is to blend (low pass filter) the motion between consecutive rotations. That method requires that N rotations be executed before the correct average forces can be synthesized. Here we address these two issues by proposing an alternative method which (i) uses sinusoidal velocities on the surface's three degrees of freedom simultaneously (low spectral content) and (ii) achieves the desired average forces in a single, closed motion of the surface, involving its 3 degrees of freedom simultaneously.

Recall that Equation 3, 4, and 5 express the instantaneous surface velocity, frictional force, and average force (in one period) at location P, respectively, as a function of the surface's rigid velocities u_1, u_2, and u_3. Here we consider these to be "sinusoids" of the form:

$$u_i(t) = \sum_{j=1}^{N} [a_{ij}\sin(jt) + b_{ij}\cos(jt)], \; i = 1\ldots3 \quad (13)$$

Where the a_{ij} and b_{ij} are $3N$ arbitrary constants. Assume, w.l.o.g., $T = 2\pi$.

3.2.1 The N=1 case

If $N = 1$, Equation 13 reduces to:

$$u_i(t) = a_i\sin(t) + b_i\cos(t), \; i = 1\ldots3 \quad (14)$$

with 6 free parameters $a_1, b_1, a_2, b_2, a_3, b_3$. From trigonometry:

$$a\sin(t) + b\cos(t) = \sqrt{a^2+b^2}\cos(t + \tan^{-1}\frac{a}{b}) \quad (15)$$

i.e., each of the surface's degrees of freedom moves as a simple cosine with selectable amplitude and phase. We shown that in this case, $\bar{f}_{2d} = 0$ for any choice of the 6 free parameters. It is sufficient to show that

$$\int_{-\pi}^{\pi} \frac{\nu_s(P,t)}{||\nu_s(P,t)||} dt = 0$$

From Equation 3, the components of this integrand will be of the form:

$$\begin{aligned} g_1(t) &= \frac{\alpha_1 s_1 + \alpha_2 c_1}{\sqrt{\alpha_3 + \alpha_4 s_2 + \alpha_5 c_2}} \qquad (16) \\ \text{where: } s_i &= \sin(it) \\ c_i &= \cos(it) \end{aligned}$$

where α_i, $i = 1\ldots5$ are polynomials on a_i,b_i, $i = 1\ldots3$, and the components of P. It can be shown that $g_1(t) = -g_1(t-\pi)$, so $g_1(t)$ integrates to zero in the $[-\pi, \pi]$ interval.

3.2.2 The N=2 case

If $N = 2$, Equation 13 reduces to:

$$\begin{aligned} u_i(t) &= a_{i1}\sin(t) + b_{i1}\cos(t) + \\ &\quad a_{i2}\sin(2t) + b_{i2}\cos(2t), \; i = 1\ldots3 \quad (17) \end{aligned}$$

Equation 15 implies that each of the u_i's is the sum of a sine to a double-frequency sine, with amplitudes and phases settable through a total of 12 free parameters.

Observation 3 *With $N = 2$, $\bar{f}_{2d} \neq 0$ in general.*

Consider the special case of $u_2 = u_3 = 0$, and $u_1(t) = \sin(t) + b\cos(2t)$. It can be shown that for some values of b this motion is time-asymmetric [17], i.e., it will produce non-zero forces in x direction.

With $N = 2$, the components of $\bar{f}_{2d}$ will have the form of function $g_2(t)$ shown below:

$$\begin{aligned} g_2(t) &= \frac{\beta_1 s_1 + \beta_2 c_1 + \beta_3 s_2 + \beta_4 c_2}{\sqrt{\beta_5 + r_2 + r_3 + r_4}} \qquad (18) \\ r_2 &= \beta_6 s_2 + \beta_7 c_2 \\ r_3 &= \beta_8 s_3 + \beta_9 c_3 \\ r_4 &= \beta_{10} s_4 + \beta_{11} c_4 \end{aligned}$$

where the β_i's are polynomials on the 12 parameters and on P_x, P_y. The above expression is not integrable in closed-form, furthermore, the integrand has a pole at $a_{ij} = b_{ij} = 0$, making it inappropriate for taylor-series expansion. A clever alternative is to approximate the u_i sinusoids with triangular waves. Namely, consider the piecewise linear approximations $s^*(t) \simeq \sin(t)$ and $c^* \simeq \cos(t)$ defined as:

$$\begin{aligned} s^*(t) &= \begin{cases} -2 - 2t/\pi & , \; -\pi \leq t < -\pi/2 \\ 2t/\pi & , \; -\pi/2 \leq t < \pi/2 \\ 2 - 2t/\pi & , \; \pi/2 \leq t < \pi \end{cases} \\ c^*(t) &= \begin{cases} 1 + 2t/\pi & , \; -\pi \leq t < 0 \\ 1 - 2t/\pi & , \; 0 \leq t < \pi \end{cases} \end{aligned}$$

Scaling the triangular waveform by $12/\pi^2$ yields the best least-squares fit of a sine, reducing the the average absolute error from 15% to 8.5%, as shown in Figure 11. The spectrum of a triangular wave only contains odd harmonics, whose amplitudes are decay as $1/f^2$ [8], rendering it a "near" sine:

$$s^*(t) = s_1 + \frac{s_3}{9} + \frac{s_5}{25} + \cdots$$

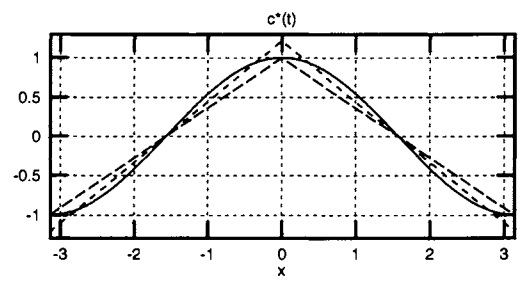


Figure 11: *A plot of* $\cos(t)$ *and its its triangular approximation* $c^*(t)$. *Also shown is* $c^*(t)$ *scaled by the best least-squares fit amplitude* $12/\pi^2$.

By defining similar triangular approximations s_2^*, and c_2^* for the double frequency sines and cosines, $g_2(t)$ above takes the following new form:

$$g_2^*(t) = \frac{k_1 + k_2 t}{\sqrt{k_3 + k_4 t + k_5 t^2}} \qquad (19)$$

with k_i, $i = 1 \ldots 5$ polynomials on the 12 free parameters and on P. From an integral table we obtain:

$$\begin{aligned} \int_0^1 g_2^*(t)dt &= \frac{\left(k_1 k_5 - \frac{k_2 k_4}{2}\right) \log|c_3| + k_2(c_1 - c_2)}{k_5^{3/2}} \\ c_1 &= \sqrt{k_5(k_3 + k_4 + k_5)} \\ c_2 &= \sqrt{k_3 k_5} \end{aligned} \quad (20)$$

There are 8 intervals in $[-\pi, \pi]$ where the linear coefficients for s^*, s_2^*, c^*, and c_2^* are constant, as shown in Figure 12. Separate expressions for the k_i's must be computed for each interval, and Equation 20 must be broken down into a sum of integrals. Let $g_{2,j}^*$ denote g_2^* in interval j, $j = 1 \cdots 8$. The following expression maps the 12 parameters and the coordinates of P to the average frictional force at P:

$$\bar{f}_{2d}(P, a_{ij}, b_{ij}) = \frac{\mu m g}{8} \sum_{j=1}^{8} \int_0^1 g_{2,j}^*(t)dt \quad (21)$$

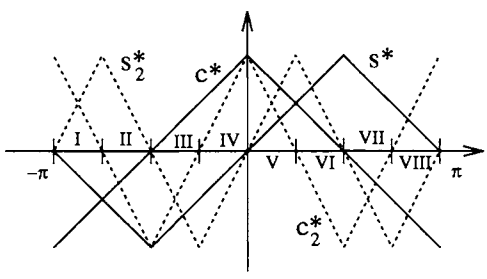


Figure 12: *The s^*, c^*, s_2^*, and c_2^* triangular approximations in $[-\pi, \pi]$. Roman numerals I to VIII label the 8 distinct regions where the functions' linear coefficients are constant.*

The part manipulation problem requires that Equation 21 be inverted, namely, a set of $3N$ motion parameters must be computed which synthesize desired average forces at one or more parts' locations. Let M be the number of parts. Then $\bar{f}_{2d}$ is a $3N$-to-$2M$ map. So long as the map's rank is greater than $2M$, $\bar{f}_{2d}$ can be inverted. The task of analyzing the map's rank algebraically is too complex, especially given the form of Equation 20. Good invertibility however can be achieved with a large enough number of free parameters, i.e., redundancy. For the $N = 2$ case, 12 parameters allow for force to be specified in up to 6 distinct locations, though here we will consider it for just 2 (co-dimension 8). If more parts are involved, higher harmonics such as $\sin(3t)$ and $\cos(3t)$ are needed.

3.2.3 Iterative Inversion

Since a closed-form $\bar{f}_{2d}^{-1}$ is out of the question, we will use a numeric method (gradient descent). We first construct a database of "initial guesses" as follows:

1. Generate a large number of random samples (P, a_{ij}, b_{ij}), each with 14 entries (2 random part coordinates, and motion 12 parameters).
2. Pipe each sample through the forward map (Equation 21), and obtain the corresponding average force $(\bar{f}_x, \bar{f}_y)$.
3. Store the 12-tuple (a_{ij}, b_{ij}) in a search tree, with the 4-long vector $(P, \bar{f}_x, \bar{f}_y)$ as the index.

the above process is illustrated in Figure 13. We use kd-trees [1] as the search/storage data-structure, for both its simplicity and efficiency. Namely, given a query (P', f_1', f_2'), the kd-tree can retrieve the data linked to the closest-matched index in logarithmic time.

For simplicity, we will consider the case of force specification at two locations P_1 and P_2. Let the desired forces be f_1 and f_2, respectively. To find the required 12 parameters a_{ij}, b_{ij} we proceed as follows:

1. Define $\bar{P}$ as the mid-point between P_1 and P_2, and $\bar{f}$ as the average of f_1 and f_2, Figure 14.
2. Query the kd-tree with $(\bar{P}, \bar{f})$, and retrieve the closest "initial guess" vector $Q = (a_{ij}, b_{ij})$.
3. Numerically estimate the Jacobian $J = \partial \bar{f}_{2d} / \partial Q$, evaluated at $(\bar{P}, Q)$.
4. Let Δf represent the 4-long error vector with components $f_i - \bar{f}_{2d}(P_i, Q)$, $i \in \{1, 2\}$. Descend along the error gradient by setting $Q = Q - J^{-1} \Delta f$.

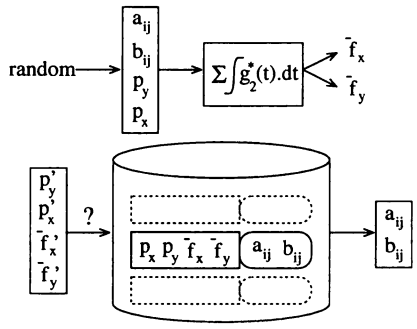


Figure 13: *Top: a random 14-tuple (P, a_{ij}, b_{ij}) is piped through the forward map, yielding the components $\bar{f}_x$ and $\bar{f}_y$ of the average force. A new entry entry indexed by (f_x, f_y, P) and associated data (a_{ij}, b_{ij}) is inserted into a search structure (kd-tree). Bottom: A 4-long query vector is presented to the search structure; the which quickly locates the entry with the closest-matching index and retrieves the associated data.*

5. Repeat from (3) till Δf's magnitude (the error in force) falls below a threshold.

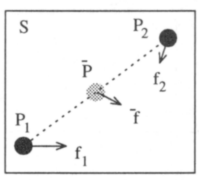


Figure 14: *Given 2 (or more parts), a query vector $(\bar{P}, \bar{f})$ as the average of all part positions and desired forces.*

3.3 Manipulability and scalability

Given one or more part positions, what are the average force directions easiest to synthesize at each part, i.e., requiring parameters with the least magnitude. This notion is akin to that of a velocity ellipsoid at the tip of a serial manipulator. The complex dependency of the forward map on the 12 parameters makes this expression opaque to any analysis. Linearization is also impossible since the Jacobian of $\bar{f}_{2d}$ is not defined at the origin (zero parameter vector). In practice, the onset of non-invertibility is signaled by an increase in the number of iterations taken by the numerical solver per step. The most ill-conditioned force direction can be estimated from the forward map in the following fashion: pipe a large number of randomly-chosen parameter vectors through the forward map. The worst directions are those which yield the highest input/output magnitude ratios (shrinkage of output signals loss of rank in that direction).

The 2nd manipulation method can produce desired forces with a single, sinusoidal motion, featuring a shorter, more band-limited motion with respect to the 1st method. This comes with the price of no good theory for the system's invertibility and difficult scalability. Namely, with the 1st method, incorporating new centers of rotation (adding redundancy) does not affect the structure of the solver. Here, adding more free parameters is obtained with higher order harmonics. For example, adding the $3t$ harmonic would increase the number of free parameters from 12 to 18. New expressions for the k_i's (Equation 19) need to be derived, e.g., using a symbolic engine, for each of the constant coefficient regions in the triangular approximations. This number jumps from 8 to 16, and is in general 2^r, where r is the order of the highest harmonic used.

3.4 Sensorless manipulation

Divergence relates to the shrinkage or expansion of a unit area element flowing through the field [20]. In manipulating parts with force fields, non-zero divergence can be used to automatically reduce the uncertainty about a part's location and/or orientation (see [9] for an example). Unfortunately, the divergence of a frictional force field $\nabla\phi = \phi_{x,x} + \phi_{y,y}$ is zero everywhere. Linearity implies that average force fields obtained by N rotations and/or by a continuous (sinusoidal) motion will be also divergence-free, ruling out sensorless manipulation.

4 Dynamic Simulation

4.1 1st method: parallel sorting of 10 parts

A set of 5 grey and 5 black parts is scattered at random over the surface. The goal is to move all black parts to the left edge and all grey parts to the right

edge of the surface. We use a very simple type of motion planning. At every step, apply a force along $-x$ to the black parts and along $+x$ to grey ones. We will use the method of sequencing rotations, we 24 rotation centers. With a total of 20 constraints per step, this gives us 4 degrees of redundancy. SVD is used to compute a bounded solution vector. As parts are pushed to their respective sides, they come inevitably close to each other. Such situations ill condition the system, causing SVD's solution to produce actual forces which deviate (considerably) from the desired ones. Interestingly, this produces a convenient effect – parts coming close to each other will deviate from their horizontal paths and "dodge" each other, avoiding collisions. Since this is a side effect of SVD, there are no motion planning guarantees here... Snapshots of a dynamic simulation of this example are shown in Figure 15.

4.2 2nd method: trajectory following for 2 parts

Consider two parts P_1 and P_2 which must move with constant speed along distinct pre-specified polygonal trajectories on S. A simple PD controller is used for each part to compute the force needed to reduce the error (deviation in trajectory and desired speed). We use the 2nd method (sinusoidal control) to synthesize the forces computed by the PDs. Snapshots of a dynamic simulation of this example are shown in Figure 16. Numeric inversion typically iterates less than 5 times per step, allowing for real-time visualization.

5 Hardware Experiments

A prototype of the device shown in Figure 17 has been built out of rather inexpensive parts – the motors are recycled disk-drive voice coils; the plate is a thin slab of kitchentop material.

A simple experiment was performed with an earlier version of the the shaker table, for which the surface was an aluminum baking tray. The experiment consisted in driving the surface to a periodic, time-asymmetric [17] rotation about its center, at a few 10s of Hz. We placed a 1 cent coin on the surface, and observed it describe a constant-velocity motion along a circular trajectory, demonstrating part feeding circles around a circular path. Centrifugal effects slowly increase the trajectory's radius causing the coin to eventually touch (or fall off) the surface's boundary. Snapshots of this experiment are shown in Figure 18.

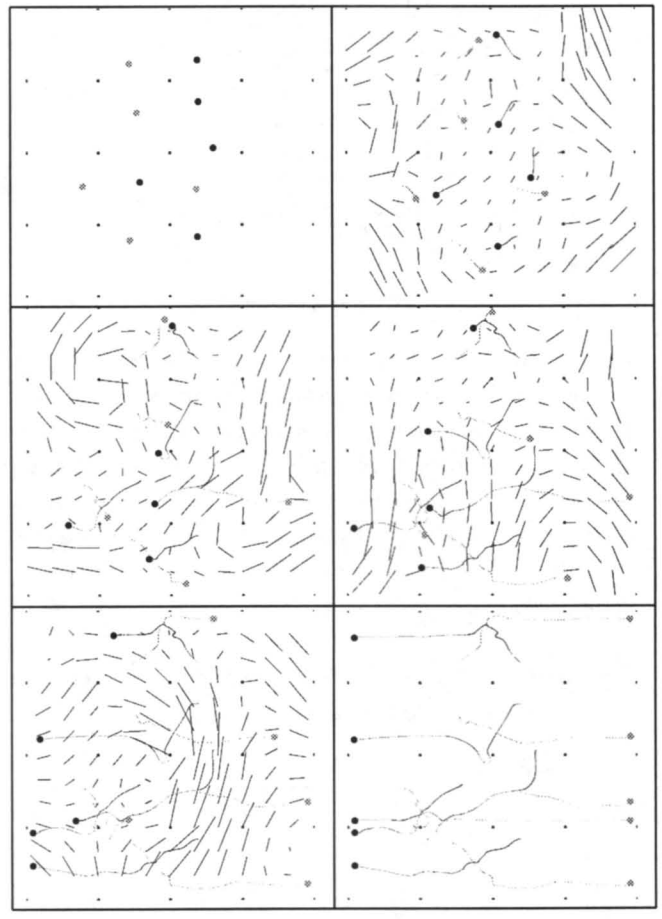


Figure 15: *The part sorting simulation. The sequence should be read left to right, top to bottom. 5 dark and 5 grey disks appear in the first snapshot at their original locations. The goal is manipulate them in parallel so at the end all black go to the left and all grey go to the right. Force field vectors are shown for each snapshot. Also shown are the trails left by the motion of each part. Note that only strictly horizontal force is applied to the parts – the deviations are caused by ill-conditioning in the linear system.*

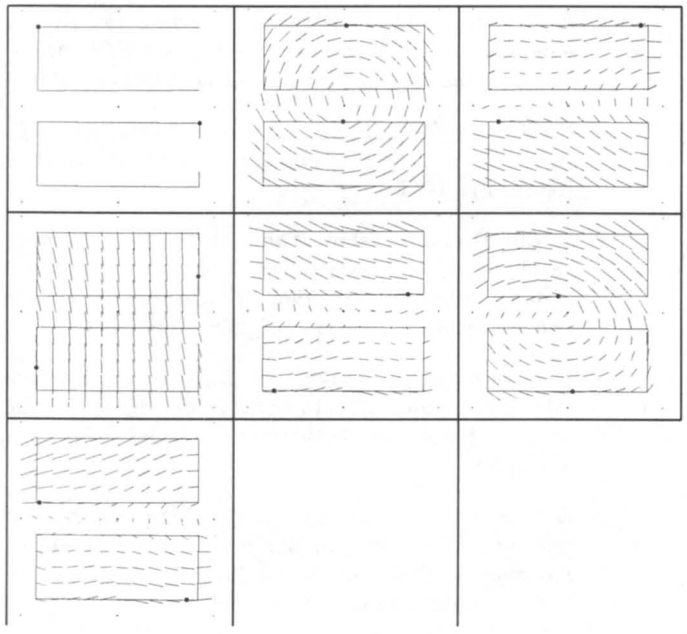


Figure 16: *The trajectory following simulation. Read from left to right, top to bottom. At the top left snapshot, two parts are shown (dark disks) at the beginning of their planned rectangular trajectories. Force field vectors are shown at each snapshot.*

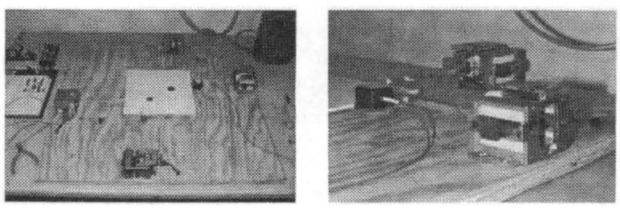


Figure 17: *Left: a prototype of the device built from inexpensive parts. Right: a closeup of two motors; also shown is an accelerometer attached to the plate.*

6 Conclusion

We have presented a novel approach to part manipulation in the plane based on the time-averaging of coulomb friction at one or more points under a rigid, closed motion of the surface. The main feature of the current approach is its mechanical simplicity. Two motion computation methods have been proposed: one involves the execution of a finite number of rotations about a fixed set of points. The other involves computing the parameters for a smooth motion of the surface, where each of its degrees of freedom moves in sinusoidal fashion. The second method has the advantage of much lower bandwidth, being less likely to induce suspension resonance; however its solution requires a rather complex, non-scalable iterative inversion machinery. Dynamic simulation results for two parallel manipulation instances have been presented, illustrating the effectiveness of both methods. Simple experiments with an existing prototype have been presented. The next major step in this research involves developing the motor control and image processing code so as to integrate the device with a supervising PC.

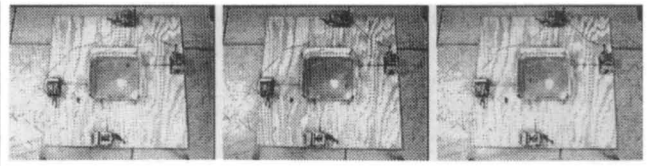


Figure 18: *Snapshots of the coin rotation experiment: an early version of the shaker table is shown, with a baking tray used as the surface. As the tray rotates time-asymmetrically about its center, a 1 cent coin moves with constant speed along a circular trajectory.*

Acknowledgements

This work was supported by NSF grant CDA-9726389: "Challenges in CISE: Planning and Control for Massively Parallel Manipulation". Thanks to Ken Goldberg, Karl Böhringer, Wolfgang Zesch, and Yan Zhuang for useful discussions, suggestions, and comments.

References

[1] J. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18:509–517, 1975.

[2] K. Böhringer, V. Bhatt, and K. Goldberg. Sensorless manipulation using transverse vibrations of a plate. In *IEEE International Conference on Robotics and Automation*, Nagoya, Japan, May 1995.

[3] K. Böhringer, B. Donald, R. Mihailovich, and N. MacDonald. Sensorless manipulation using massively parallel microfabricated actuator arrays. In *IEEE International Conference on Robotics and Automation*, San Diego, CA, May 1994.

[4] G. Boothroyd. *Assembly automation and product design.* Marcel Dekker, Inc., New York, NY, 1991.

[5] J. Canny and K. Goldberg. RISC for industrial robots: Recent results and open problems. In *IEEE International Conference on Robotics and Automation*, San Diego, CA, May 1994.

[6] J. Craig. *Introduction to robotics, mechanics and control.* Addison-Wesley, Reading, MA, 2nd edition, 1989.

[7] W. P. et al. *Numerical recipes in C.* Cambridge University Press, New York, NY, 2nd edition, 1995.

[8] S. Haykin. *An introduction to analog and digital communications.* John Wiley & Sons, Inc., New York, NY, 1989.

[9] L. Kavraki. Part orientation with programmable vector fields: Two stable equilibria for most parts. In *IEEE International Conference on Robotics and Automation*, Albuquerque, NM, Apr. 1997.

[10] S. Konishi and H. Fujita. A conveyance system using air flow based on the concept of distributed micro motion systems. *Journal of Micro-electromechanical Systems*, 3(2):54–58, June 1994.

[11] J. Krishnasamy, M. Jakiela, and D. Whitney. Mechanics of vibration – assisted entrapment with application to design. In *IEEE International Conference on Robotics and Automation*, Minneapolis, MN, May 1995.

[12] C. Liu, T. Tsao, P. Will, Y. Tai, and W. Liu. A micromachined magnetic actuator array for microrobotics assembly systems. In *International Conference on Solid-State Sensors and Actuators*, Stockholm, Sweden, 1995.

[13] J. Luntz, W. Messner, and H. Choset. Parcel manipulation and dynamics with a distributed actuator array: the virtual vehicle. In *IEEE International Conference on Robotics and Automation*, Albuquerque, NM, Apr. 1997.

[14] M. Mason. *Mechanics of Manipulation.* MIT Press, Cambridge, MA, 1997.

[15] F. Moesner and T. Higushi. Devices for particle handling by an ac electric field. In *IEEE Microelectromechanical Systems*, pages 66–71, Jan. 1995.

[16] R. Murray, Z. Li, and S. Sastry. *A mathematical introduction to robotic manipulation.* CRC Press, Boca Raton, FL, 1994.

[17] D. Reznik and J. Canny. The coulomb pump: A novel parts feeding method using a horizontally-vibrating surface. In *IEEE International Conference on Robotics and Automation*, Leuven, Belgium, May 1998.

[18] D. Reznik and J. Canny. A flat rigid plate is a universal planar manipulator. In *IEEE International Conference on Robotics and Automation*, Leuven, Belgium, May 1998.

[19] D. Reznik, J. Canny, and K. Goldberg. Analysis of part motion on a longitudinally vibrating plate. In *International Workshop On Intelligent Robots and Systems*, Grenoble, France, Sept. 1997.

[20] H. Schey. *Div, grad, curl, and all that.* W.W. Norton, New York, NY, 2nd edition, 1992.

[21] J. Suh, S. Glander, R. Darling, C. Storment, and G. Kovacs. Combined organic thermal and electrostatic omnidirectional ciliary microactuator array for object positioning and inspection. In *Proc. Solid State Sensor and Actuator Workshop*, Hilton Head, NC, June 1996.

[22] N. Tran, K. Chan, and V. Hayward. Object behavior using a vibrating plate testbed for part presentation research. EE304-494 project writeup, McGill University, Dec. 1995.

Velocity Field Design for the Modular Distributed Manipulator System (MDMS)

Jonathan Luntz, *Carnegie Mellon University, Pittsburgh, PA, USA*
William Messner, *Carnegie Mellon University, Pittsburgh, PA, USA*
Howie Choset, *Carnegie Mellon University, Pittsburgh, PA, USA*

The Modular Distributed Manipulator System *(MDMS) is a novel materials handling system which is a fixed array of actuated wheels capable of inducing arbitrary motions in the plane. This work develops control laws for transporting and manipulating macroscopic-scale objects (parcels) that rest on the array. The dynamics of parcel transport and manipulation lead to the necessary understanding of such laws. The dynamics are based on an exact discrete representation of the system, unlike other methods where a continuity assumption is made. Given a particular set of wheel speeds, the dynamics of a parcel's motion are derived by examining the distribution of weight among the supporting cells subject to a friction model governing the parcel-wheel contact. The resulting relationship between parcel behavior and wheel speeds can be inverted, therefore wheel speeds can be calculated from desired parcel behavior.*

1 Introduction

The *Modular Distributed Manipulator System* (MDMS) transfers, as well as manipulates, objects in the plane, enhancing applications such as flexible manufacturing and package handling systems. Flexible manufacturing requires that working part paths be arbitrarily specified and that these paths be readily updated by a "factory programmer." Package handling systems, such as airport baggage handling, require parcels to be sorted, oriented, and transported to one of several locations.

Two existing methods of material transfer are conveyor belts and robotic manipulators. The conveyor belts transfer many large heavy objects for long distances, but lack the ability to precisely orient objects.

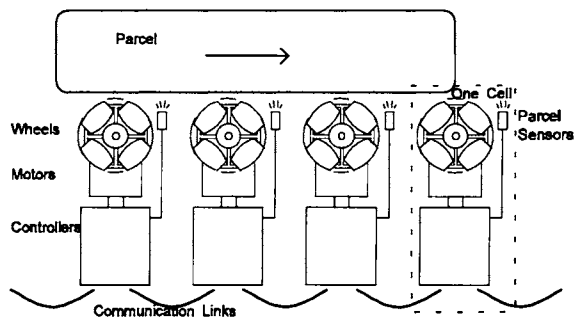


Figure 1: *A few manipulator cells carrying a parcel.*

Furthermore, conveyor belts cannot re-order objects, nor are they well suited to merging objects from different sources. Robots, on the other hand, are able to precisely position and orient objects, but only one at a time, and are limited by strength and reach. The MDMS combines the benefits of conveyor and robotics transport systems while overcoming their limitations.

This alternative method comprises a fixed array of actuators that cooperate to handle objects. In the prototype system, each cell consists of a pair of actuators whose combined action can effect force in any planar direction to a parcel resting on top of the array. Furthermore, the cells contain binary sensors that detect the presence of an object. See Figure 1.

In this system, the parcels are significantly larger than each cell; several cells support a single parcel that can be made to translate and rotate in the plane. Since sensing and actuation are distributed, each of many parcels can be manipulated independently, appearing as if each parcel were carried by a separate vehicle (Figure 2).

Figure 2: *The MDMS: Several parcels can be translated and rotated, independently.*

Implementation of this system requires the development of control laws and policies to provide the desired motions of parcels. These control laws must reflect a physical system, such as the one described in Section 2. These laws build upon an already vast literature of manipulation and actuator arrays, which are described in Section 3.

The control laws are based on the dynamics of a parcel being transported by the MDMS. These dynamics are derived in Section 4 by first calculating the forces which support a parcel resting on an arbitrary set of cells. A friction law is then applied to compute the net forces and moments acting on the parcel.

Section 4 describes the dynamics, given a specific set of wheel velocity. Section 5 describes the inverse problem: given desired parcel dynamics, compute the necessary wheel velocities to attain these dynamics uniformly over the entire array. The resulting field is analyzed for its translational and rotational properties. Under certain assumptions under parcel shape and array symmetry, an object can be positioned and oriented within the resolution of the array to symmetry.

2 Application and Prototype Configuration

For many applications, a dedicated robot or conveyor is the simplest and most appropriate solution. There are cases, however, where additional flexibility, redundancy, modularity and reconfigurability are required, giving the MDMS significant advantages over existing systems.

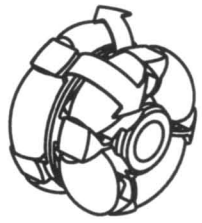


Figure 3: *Roller wheel.*

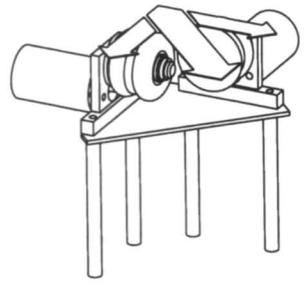


Figure 4: *Prototype cell.*

The MDMS can be used in conjunction with traditional robots and conveyor belts to form hybrid systems. For example, in airport baggage handling, long conveyors can be used to transport parcels over long distances while MDMS arrays can be installed at conveyor junctions to sort and to re-direct parcel traffic. In flexible manufacturing, the MDMS can be used to transport objects between robot workspaces where simple robots are used for object fixturing.

Mechanical Configuration. A prototype system was built consisting of a small array of cells capable of transporting objects about the size of a bread box. Each cell consists of a pair of orthogonally oriented motorized roller wheels (Figures 3 and 4) which are capable of producing a force perpendicular to their axes, while allowing free motion parallel to their axes. Each wheel is driven through a gear reduction by a small DC motor. Each of these cells is attached to a breadboard base (Figure 5) to form a regular array. Currently, the prototype system consists of 18 cells which can be arranged either in a 1-D or a 2-D grid, shown in the photograph in Figure 6.

Electronic Configuration. Each cell is controlled by a MC68HC11-based single-board computer. Distributed control such as this is a relatively underdeveloped and interesting control paradigm and is well suited for the MDMS because it would become impractical or impossible for a single computer to control

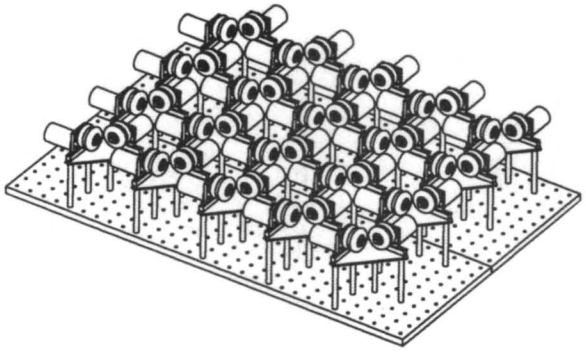


Figure 5: *Two-dimensional array of cells.*

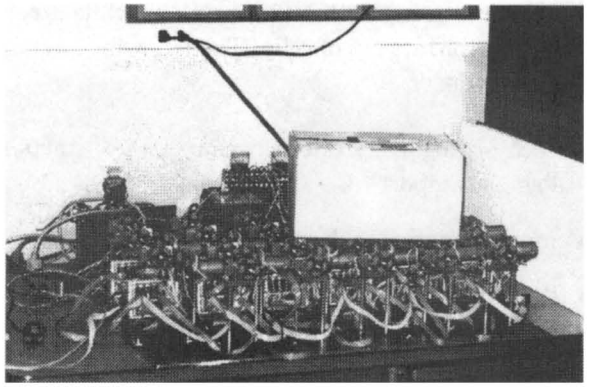


Figure 6: *Current experimental setup.*

hundreds or possibly thousands of cells. The speed of the wheels is measured by a quadrature encoder. The presence of a parcel is sensed by a phototransistor which detects the shadow from the overhead lights. The motor is driven by pulse width modulation using an H-bridge circuit.

Network Configuration. An important aspect of our control scheme is the communication between cells. Each controller has an RS232 communications port which is multiplexed between each of the four neighboring cells, requiring active selection of neighbors. Handshaking lines are used to alert a neighboring cell of a pending message. The cells communicate with their neighbors in a consistent pattern, automatically synchronizing communication.

This communication scheme is more appropriate for our system than a more standard shared-bus local area network. The multiplexed network supports many simultaneous communications, whereas the shared bus only allows one at a time. Our network requires only relies on local information without addressing whereas a standard network requires each cell to be aware of its address and is limited by a maximum address space. Messages can be passed along the array if necessary. Messages are generated using a communication language which involves the construction of short "sentences" made up of labeled pieces of data, or words, punctuated by a checksum.

3 Prior Work

Mason and Erdmann [1, 3] provided a radical alternative to standard robotic manipulation by significantly simplifying the robot manipulator and developing manipulation algorithms for these low-degree-of-freedom sensorless mechanisms. Goldberg [4] developed an algorithm which orients to symmetry a part with a sequence of gripper open, close, and rotation operations without sensor information. This sequence of operations is termed a *squeeze*.

Böhringer, Donald, et. al. [2] applied this type of sensorless manipulation to an array of micromechanical actuators which were used to manipulate very small objects. Whereas Goldberg used a single centralized manipulator to orient objects, Böhringer used distributed actuation, but in both methods the control is centralized. The Böhringer system differs from the MDMS in that (i) it is on a small scale, (ii) the control is centralized, and (iii) the manipulation is entirely sensorless. It is worth noting that although their work has motivated some of the authors' research, their work uses quasi-static analysis, and ignores dynamics because of the small scale application.

Recently, Kavraki [5] supplied further analysis of microactuated systems using elliptical potential fields to orient an object to symmetry without sensors. Potential functions are used to predict stable configurations of objects. Continuous potential functions adequately approximate Kavraki's and Böhringer's systems, because many micro-sized cells provide essentially continuous fields. The MDMS, however, is highly discrete

such potential fields can not be used for motion planning. Only a small number of cells support an object.

In a previous paper by the authors [6] the first step was taken in examining the dynamics of a parcel carried by the MDMS, where the one dimensional motion of the parcel along the array of cells was considered. In that paper, the forces between each cell and the parcel were calculated, and both a coulomb and a viscous-like friction law were considered. The resulting parcel dynamics are that of a simple or damped harmonic oscillator, where the frequency, center of oscillation, and damping constant are parameters which change as the parcel shifts from one set of supports to another. This simple oscillator behavior was also observed in the prototype system. This work was extended into two dimensions in a more recent paper by the authors [7]. Translational forces and rotational torques are calculated as a function of parcel position. Similar mass-spring-damper behavior in the plane was observed.

This paper upgrades the authors' previous work in two dimensions by significantly reducing the amount of calculation necessary to calculate the forces and torques, allowing for the inversion of the dynamics. A wheel velocity field is thus generated which produce desired parcel dynamics with a single equilibrium position regardless of symmetry and cell resolution, and, under certain assumptions, a single orientational position (within symmetry) to the resolution of the cells.

The ultimate goal of this work is to be able to predict and control the overall motion of one or more parcels with two-dimensional translation with rotation in the plane. The work on the MDMS continues a trend in the previous work, where a transition was made from standard, centralized manipulation with sensors to a minimalist, sensorless (but still centralized) approach to a distributed sensorless approach on a microscopic scale. The MDMS again focuses on a macroscopic scale and also includes sensors and feedback. This trend is shown in Figure 7.

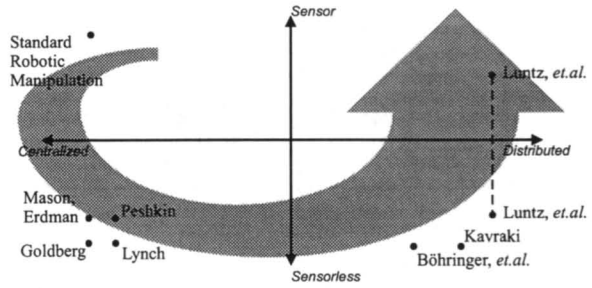


Figure 7: *Trends in Manipulation.*

4 Dynamics of Manipulation

Towards the goal of representing the dynamics of handling a parcel in the plane with an array of actuators, we will consider the dynamics of an array of cells transporting and rotating a parcel in the plane while it rests on a single arbitrary set of cells. The following assumptions are made:

- Each orthogonally oriented pair of wheels acts as a single support.
- Supports act as springs to support the parcel.
- The bottom of the parcel is flat.
- The speed of each wheel is constant.
- Horizontal force between each wheel and the parcel is due to sliding friction.
- Viscous friction (proportional to speed) exists between the wheels and parcel which is also proportional to the normal force.

In order to compute the horizontal translation and rotation dynamics of the parcel, first the equilibrium of the parcel and constitutive relations for the supports are used to compute the normal forces supporting the parcel. Then, a friction law between the parcel and wheels is applied to determine the net horizontal forces and moments The result will be a net force and torque acting on the parcel as a function of the parcel's position. The problem will be considerably simplified by the structure of the mathematics and by several convenient mathematical identities.

Notation: Because of the large number of variables used, this paper will employ a consistent notation. Scalar variables will appear in normal math font (e.g.

s), vectors will appear in normal math font with an arrow (e.g. $\vec{v}$), and matrices will appear in bold font (e.g. $\mathbf{m}$). Subscripts x and y will indicate x and y components, and subscript i will indicate the ith cell. For example, $\mathbf{V}$ is a matrix made up of velocity (column) vectors $\vec{V_i}$ for each cell, with components V_{x_i} and V_{y_i}. $\mathbf{V}$ can also be said to be made up of component (row) vectors $\vec{V_x}$ and $\vec{V_y}$ listing the velocity components for all the cells.

4.1 Normal forces

In order to solve for the n forces supporting the parcel, the equilibrium of the parcel in both the vertical (z) direction and in rotation about the x and y axes must be considered. Since equilibrium only provides three equations, and we have n supports, the system is statically indeterminant, and flexibility in the supports (or equivalently, in the parcel surface, as in the prototype) needs to be considered to solve for the remaining $n-3$ forces. These $n-3$ equations will be referred to as *compatibility* equations. The n equations will be represented in matrix form, and can be used to solve for the vector of n normal forces, $\vec{N}$.

4.1.1 Parcel equilibrium

Consider n cells arranged arbitrarily in the x-y plane having coordinates as entries of the matrix below.

$$\mathbf{X} = \begin{bmatrix} \vec{x} \\ \vec{y} \end{bmatrix} = \begin{bmatrix} x_1 \ldots x_n \\ y_1 \ldots y_n \end{bmatrix} = \begin{bmatrix} \vec{X}_1 \ldots \vec{X}_n \end{bmatrix} \tag{1}$$

A parcel, whose center of mass is located at $\vec{X}_{cm} = \begin{bmatrix} x_{cm} & y_{cm} \end{bmatrix}^{\mathrm{T}}$ resting on n of these cells, is supported by vertical normal forces

$$\vec{N} = \begin{bmatrix} N_1 & N_2 & N_3 & \ldots & N_n \end{bmatrix}. \tag{2}$$

The parcel must be in equilibrium in the vertical direction and in rotation about the x and y axes. The vertical equilibrium of the parcel, with weight W, requires that

$$\sum_{i=1}^{n} N_i = W = \begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \end{bmatrix} \vec{N}^{\mathrm{T}}. \tag{3}$$

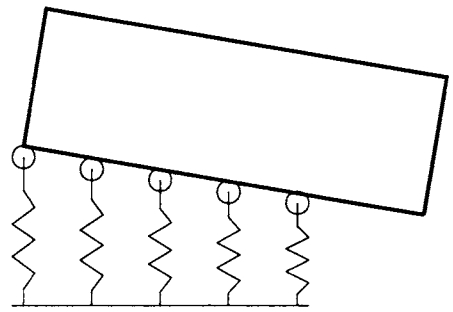


Figure 8: *Flexible supports in many-cell case.*

Rotational equilibrium about the x and y axes require that the moments induced by the normal forces about any point (in this case, the arbitrarily located origin of our coordinate system) sum to the moment about this point induced by the weight of the parcel. The moment equilibrium about the x and y axes, respectively, requires that

$$\sum_{i=1}^{n} N_i y_i = \vec{y}\,\vec{N}^{\mathrm{T}} = W y_{cm}, \text{ and} \tag{4}$$

$$\sum_{i=1}^{n} N_i x_i = \vec{x}\,\vec{N}^{\mathrm{T}} = W x_{cm}. \tag{5}$$

At this point in the development, there are n unknowns (the elements of $\vec{N}$), but only three equations (3,4, and 5) from equilibrium.

4.1.2 Compatibility

To solve for the remaining $n-3$ forces, flexibility in the system must be considered. A simple model of this flexibility is shown in side view in Figure 8. Each support is assumed to be a spring, with Hooke's Law ($N_i = K_s \Delta z_i$) representing the compression of the ith cells under a normal load. Physically, this flexibility can be thought of as a flexible suspension under each wheel, as will be included in the next generation of our hardware array. In the current system, an equivalent flexibility is apparent in the bottom of the parcel, either in the cardboard of a box, or a foam pad on which the box rests.

Assuming the bottom of the box is nominally flat, the flexible cells conform to the bottom of the box to

distribute the weight. All the cells (under the parcel) lie in this plane. The plane of the bottom of the parcel can be represented as

$$z = a'y + b'x + c'. \tag{6}$$

Each cell will obey the following displacement relation.

$$\begin{aligned} \Delta z_i = N_i/K_s &= a'y_i + b'x_i + c' \\ \Rightarrow N_i &+ ay_i + bx_i + c = 0 \end{aligned} \tag{7}$$

4.1.3 Combining Equations

Equations 3, 4, and 5 along with n instances of Equation 7 supply $n+3$ equations and $n+3$ unknowns (n N_i's and 3 plane parameters, a, b, and c). This system of equations can be written in matrix form.

$$\underbrace{\left[\begin{array}{ccc|ccc} 1 & \dots & 0 & 1 & x_1 & y_1 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & 1 & 1 & x_n & y_n \\ \hline 1 & \dots & 1 & 0 & 0 & 0 \\ x_1 & \dots & x_n & 0 & 0 & 0 \\ y_1 & \dots & y_n & 0 & 0 & 0 \end{array}\right]}_{\mathbf{A}} \underbrace{\left[\begin{array}{c} N_1 \\ \vdots \\ N_n \\ \hline c \\ b \\ a \end{array}\right]}_{\vec{N}^{\mathrm{T}}_{abc}} = \underbrace{\left[\begin{array}{c} 0 \\ \vdots \\ 0 \\ \hline W \\ Wx_c \\ Wy_c \end{array}\right]}_{\vec{W}} \tag{8}$$

The top portion of this equation represents the n instances of the compatibility equation (Equation 7). The bottom three rows represent the vertical, x-moment, and y-moment equations (Equations 3, 4, and 5). This equation can be inverted to solve for $\vec{N}_{abc}$ (which contains $\vec{N}$ and a, b, and c.)

$$\vec{N}^{\mathrm{T}}_{abc} = \mathbf{A}^{-1}\vec{W} \tag{9}$$

4.1.4 Solving for Normal Forces

In order to solve for the normal forces in $\vec{N}$, $\mathbf{A}$ must be inverted. Since $\mathbf{A}$ is symmetric and very structured, this can readily be done. Define a matrix $\mathbf{B}$.

$$\mathbf{B} = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ x_1 & x_2 & x_3 & \dots & x_n \\ y_1 & y_2 & y_3 & \dots & y_n \end{bmatrix}. \tag{10}$$

The matrix $\mathbf{A}$ can be written in block matrix form.

$$\mathbf{A} = \left[\begin{array}{c|c} \mathbf{I}_{n\times n} & \mathbf{B}^{\mathrm{T}} \\ \hline \mathbf{B} & \mathbf{0}_{3\times 3} \end{array}\right] \tag{11}$$

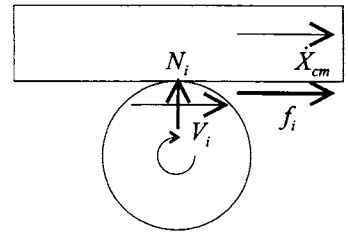


Figure 9: *Interaction between wheel and parcel.*

The inverse of the matrix $\mathbf{A}$ exists if $\mathbf{B}$ has rank 3 (which is true for general cell positions). The expression for the inverse is

$$\mathbf{A}^{-1} = \left[\begin{array}{c|c} \mathbf{I}_{n\times n} - \mathbf{B}^{\mathrm{T}}\left(\mathbf{B}\mathbf{B}^{\mathrm{T}}\right)^{-1}\mathbf{B} & \mathbf{B}^{\mathrm{T}}\left(\mathbf{B}\mathbf{B}^{\mathrm{T}}\right)^{-1} \\ \hline \left(\mathbf{B}\mathbf{B}^{\mathrm{T}}\right)^{-1}\mathbf{B} & -\left(\mathbf{B}\mathbf{B}^{\mathrm{T}}\right)^{-1} \end{array}\right]. \tag{12}$$

The derivation of the inverse is omitted, but the result can easily be verified by multiplying $\mathbf{A}$ by $\mathbf{A}^{-1}$.

Since $\vec{W}$ only multiplies nonzero elements into the right side of $\mathbf{A}^{-1}$, and only the slopes (a, b, and c) result from the lower portion of $\mathbf{A}^{-1}$, only the upper right partition of $\mathbf{A}^{-1}$ is used to calculate $\vec{N}$.

$$\begin{aligned} \vec{N}^{\mathrm{T}} &= \mathbf{B}^{\mathrm{T}}\left(\mathbf{B}\mathbf{B}^{\mathrm{T}}\right)^{-1}\begin{bmatrix} W \\ Wy_{cm} \\ Wy_{cm} \end{bmatrix} \\ &= W\mathbf{B}^{\mathrm{T}}\left(\mathbf{B}\mathbf{B}^{\mathrm{T}}\right)^{-1}\left(\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}\vec{X}_{cm}\right). \end{aligned} \tag{13}$$

4.2 Planar dynamics under viscous friction

The full planar dynamics involve translation and rotation of the parcel. The horizontal forces are derived from the normal forces through the use of a viscous-type friction law. The horizontal force from each cell $\vec{f_i}$ is proportional to a coefficient of friction μ, that cell's normal force N_i, and the vector difference between the velocity of the wheel and the velocity of the parcel at the point of the cell. This velocity difference is a function of both the translational velocity of the parcel $\dot{\vec{X}}_{cm}$, the velocity of the wheel $\vec{V_i}$, the rotation speed of the parcel about its center of mass ω, and the position difference between the cell and the center of

mass $\vec{X}_i - \vec{X}_{cm}$.

$$\vec{f}_i = \mu\left(\vec{V}_i - \dot{\vec{X}}_{cm} + \omega\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}\left(\vec{X}_i - \vec{X}_{cm}\right)\right) N_i \tag{14}$$

4.2.1 Translational Forces

The horizontal force from each cell is summed up over all the cells. This can be done by defining a wheel velocity matrix $\mathbf{V}$ as

$$\mathbf{V} = \begin{bmatrix} V_{1_x} & V_{2_x} & \dots & V_{n_x} \\ V_{1_y} & V_{2_y} & \dots & V_{n_y} \end{bmatrix} \tag{15}$$

$\mathbf{V}$, along with the cell position matrix $\mathbf{X}$, can be used to sum up the horizontal forces.

$$\begin{aligned} \vec{f} &= \sum_{i=1}^{n} \vec{f}_i = \mu\Big(\mathbf{V} - \dot{\vec{X}}_{cm}[1\,1\dots1] \\ &\quad + \omega\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}\left(\mathbf{X} - \vec{X}_{cm}[1\,1\dots1]\right)\Big)\vec{N}^{\mathrm{T}} \\ &= \mu\mathbf{V}\,\vec{N}^{\mathrm{T}} - \mu\dot{\vec{X}}_{cm}[1\,1\dots1]\vec{N}^{\mathrm{T}} \\ &\quad + \omega\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}\left(\mathbf{X}\,\vec{N}^{\mathrm{T}} - [1\,1\dots1]\vec{N}^{\mathrm{T}}\right) \end{aligned} \tag{16}$$

Note that $[1\,1\dots1]\vec{N}^{\mathrm{T}}$ is the sum of the normal forces, which is the parcel weight, W. Therefore, the last two terms in the parentheses are identically zero from Equations 4 and 5. Hence, the net horizontal force is not a function of the parcel's rotation speed and Equation 16 becomes

$$\vec{f} = \mu\mathbf{V}\,\vec{N}^{\mathrm{T}} - \mu\dot{\vec{X}}_{cm}W, \tag{17}$$

The second term in this equation is a linear damping term which is always dissipative since weight is always positive. $\vec{N}$ in the first term can be substituted from Equation 13.

$$\begin{aligned} \vec{f} &= \underbrace{\mu W\mathbf{V}\mathbf{B}^{\mathrm{T}}\left(\mathbf{B}\mathbf{B}^{\mathrm{T}}\right)^{-1}\begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}}_{\mathbf{k_s}}\vec{X}_{cm} \\ &\quad + \underbrace{\mu W\mathbf{V}\mathbf{B}^{\mathrm{T}}\left(\mathbf{B}\mathbf{B}^{\mathrm{T}}\right)^{-1}\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}}_{\vec{f}_o} - \mu\dot{\vec{X}}_{cm}W, \end{aligned} \tag{18}$$

where $\mathbf{k_s}$ is a constant 2×2 matrix and $\vec{f}_o$ is a constant 2×1 vector. The matrix $\mathbf{k_s}$ is essentially a matrix of spring constants, since it specifies force as a linear function of position. The vector $\vec{f}_o$ is an offset force.

4.2.2 Rotational Torque

In two dimensions, the torque each cell applies to the parcel is the scalar cross product of the position vector of the point of application of the force, $\vec{X}_i$, relative to the parcel center of mass, $\vec{X}_{cm}$, and the horizontal force vector from that point, $\vec{f}_i$. With viscous friction, substitute in Equation 14 for $\vec{f}_i$. Applying the matrix definition of scalar cross products, the ith cell applies the following torque.

$$\begin{aligned} \tau_i &= \mu\left(\vec{X}_i - \vec{X}_{cm}\right)\times\left(\vec{V}_i - \dot{\vec{X}}_{cm} + \omega\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}\left(\vec{X}_i - \vec{X}_{cm}\right)\right)N_i \\ &= \mu\vec{X}_i\times\vec{V}_iN_i - \mu\vec{X}_{cm}\times\vec{V}_iN_i - \mu\left(\vec{X}_i - \vec{X}_{cm}\right)\times\dot{\vec{X}}_{cm}N_i \\ &\quad + \omega\mu\left(\vec{X}_i - \vec{X}_{cm}\right)^{\mathrm{T}}\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}\left(\vec{X}_i - \vec{X}_{cm}\right)N_i \end{aligned} \tag{19}$$

This expression can be reduced by defining $R_i = \vec{X}_i\times\vec{V}_i$ (which defines $\vec{R}$) and $\mathcal{X}_i = \vec{X}_i^{\mathrm{T}}\vec{X}_i$ (which defines $\vec{\mathcal{X}}$). Also, multiplying the cross product matrices yields $-\mathbf{I}$. In addition, use the fact that $\vec{X}_i^{\mathrm{T}}\vec{X}_{cm} = \vec{X}_{cm}^{\mathrm{T}}\vec{X}_i$, and that reversing the order of a cross product changes sign.

$$\begin{aligned} \tau_i &= \mu R_iN_i - \mu\vec{X}_{cm}\times\vec{V}_iN_i + \mu\dot{\vec{X}}_{cm}\times\left(\vec{X}_i - \vec{X}_{cm}\right)N_i \\ &\quad - \omega\mu\left(\vec{X}_i - \vec{X}_{cm}\right)^{\mathrm{T}}\left(\vec{X}_i - \vec{X}_{cm}\right)N_i \\ &= \mu R_iN_i - \mu\vec{X}_{cm}\times\vec{V}_iN_i + \mu\dot{\vec{X}}_{cm}\times\left(\vec{X}_i - \vec{X}_{cm}\right)N_i \\ &\quad - \omega\mu\left(\mathcal{X}_iN_i - \vec{X}_{cm}^{\mathrm{T}}\vec{X}_iN_i - \vec{X}_{cm}^{\mathrm{T}}\left(\vec{X}_i - \vec{X}_{cm}\right)N_i\right) \end{aligned} \tag{20}$$

Note that the term multiplying ω is positive since N_i is always positive as is $\left(\vec{X}_i - \vec{X}_{cm}\right)^{\mathrm{T}}\left(\vec{X}_i - \vec{X}_{cm}\right)$.

The torque from each cell in Equation 20 must be summed up over all the cells. This can be done vectorially by taking advantage of the inner product.

$$\begin{aligned}\tau = \sum \tau_i = \mu\vec{R}\vec{N}^{\mathrm{T}} - \mu\vec{X}_{cm}\times\mathbf{V}\vec{N}^{\mathrm{T}} \\ +\mu\dot{\vec{X}}_{cm}\times\left(\mathbf{X}\vec{N}^{\mathrm{T}} - \vec{X}_{cm}[1\,1\ldots1]\vec{N}^{\mathrm{T}}\right) \\ -\omega\mu\left(\vec{\mathcal{X}}\vec{N}^{\mathrm{T}} - \vec{X}_{cm}^{\mathrm{T}}\mathbf{X}\vec{N}^{\mathrm{T}}\right. \\ \left.-\vec{X}_{cm}^{\mathrm{T}}\left(\mathbf{X}\vec{N}^{\mathrm{T}} - \vec{X}_{cm}[1\,1\ldots1]\vec{N}^{\mathrm{T}}\right)\right) \quad (21)\end{aligned}$$

Some terms can be eliminated by examining this equation. $\mathbf{X}\vec{N} = W\vec{X}_{cm}$, and $[1\,1\ldots1]\vec{N} = W$ from parcel equilibrium (Equations 3, 4, and 5). Therefore, $\mathbf{X}\vec{N} - \vec{X}_{cm}[1\,1\ldots1]\vec{N} = 0$ and the torque can be expressed as

$$\tau = \mu\vec{R}\vec{N} - \mu\vec{X}_{cm}\times\mathbf{V}\vec{N} - \omega\mu\left(\vec{\mathcal{X}}\vec{N} - W\vec{X}_{cm}^{\mathrm{T}}\vec{X}_{cm}\right) \quad (22)$$

The last term in this expression is $-\omega$ multiplied by a positive term (from Equation 20) and hence always opposes the parcel's rotation and is dissipative. The expression for the normal forces can be substituted in to give an expression for the moments acting on the parcel as a function of position and rotational speed.

$$\begin{aligned}\tau = \underbrace{\mu W\vec{R}\mathbf{B}^{\mathrm{T}}\left(\mathbf{B}\mathbf{B}^{\mathrm{T}}\right)^{-1}\begin{bmatrix}1\\0\\0\end{bmatrix}}_{\tau_o} \\ +\underbrace{\mu W\vec{R}\mathbf{B}^{\mathrm{T}}\left(\mathbf{B}\mathbf{B}^{\mathrm{T}}\right)^{-1}\begin{bmatrix}0&0\\1&0\\0&1\end{bmatrix}}_{\vec{k}_{s_\tau}}\vec{X}_{cm} \\ +\vec{X}_{cm}\times\left(\vec{f}_o + \mathbf{k_s}\vec{X}_{cm}\right) - \mu\omega\left(\vec{\mathcal{X}}\vec{N} - W\vec{X}_{cm}^{T}\vec{X}_{cm}\right) \quad (23)\end{aligned}$$

where $\mathbf{k_s}$ and $\vec{f}_o$ are the spring and offset constants from the translational dynamics, $\vec{k}_{s_\tau}$ is a 1×2 constant vector relating torque to position, and τ_o is a scalar constant torque. Note that nothing in the previous mathematics involved the orientation of the parcel, and hence, while the parcel rests on a particular set of supports, torque on the parcel is not a function of orientation. This is very important for determining stable orientations.

5 Design of Velocity Fields

The motion of the parcel as it rests on a single set of supports is simply described by a set of 9 constants with mass-spring-damper qualities. A typical problem is to create a velocity field (described by $\mathbf{X}$ and $\mathbf{V}$) to produce mass-spring-damper behavior with uniform desired properties over the entire array. In particular, an equilibrium position and return spring stiffnesses are specified, and rotational equilibrium at the translational equilibrium must be ensured. The following assumptions will be made to perform the analysis:

- The coordinate origin is at the desired equilibrium.
- The cells are arranged with mirror-symmetry around the coordinate axes.
- The parcel also has mirror symmetry.
- When the parcel rotates counterclockwise, more cells under the parcel lie in the first and third quadrants than in the second and fourth. More specifically, $\sum x_i y_i > 0$.

5.1 Translational constants

5.1.1 The Design Problem

With a viscous friction model, the translational motion of the parcel is described by

$$\vec{f} = \mathbf{k_s}\vec{X}_{cm} + \vec{f}_o - \mu W\dot{\vec{X}}_{cm} \quad (24)$$

This expression consists of 6 constants with which we can specify the dynamics of the motion of the parcel: 4 spring constants in $\mathbf{k_s}$ and two constant offset forces in $\vec{f}_o$. Consideration will only be given to the case where $\mathbf{k_s}$ is a diagonal matrix, where x and y motions of the parcel are decoupled. The diagonal terms in $\mathbf{k_s}$ tend to pull the parcel towards a central equilibrium. The off-diagonal terms act as circulatory terms, moving the parcel around the equilibrium, and are not helpful for positioning the parcel (see Figure 10). Eliminating the off-diagonal terms simplifies the design problem and improves the rotational properties of the field.

To move a parcel to a particular position, the equilibrium position $\vec{X}_{cm_e}$ and return spring strengths $k_{s_{xx}}$

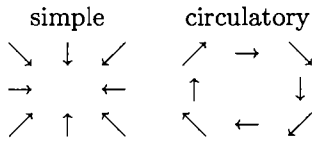


Figure 10: *A simple field (left) with diagonal* $\mathbf{k_s}$ *and a circulatory field (right) with off-diagonal entries of* $\mathbf{k_s}$.

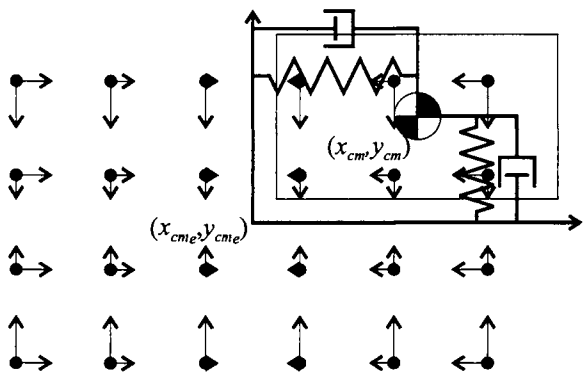


Figure 11: *With* $k_{s_{xy}} = k_{s_{yx}} = 0$ *the parcel behaves like a mass-spring damper system.*

and $k_{s_{yy}}$ will be specified. Physically, this is like the mass-spring-damper system shown in Figure 11. At equilibrium, $\vec{f_o} = -\mathbf{k_s}\vec{X}_{cm_e}$, so, in effect, $\vec{f_o}$ is specified. The design problem then becomes the problem of determining wheel velocities $\mathbf{V}$ given their positions (specified in $\mathbf{X}$ and equivalently in $\mathbf{B}$) and the constants $\mathbf{k_s}$ and $\vec{f_o}$. The section on dynamics defined the functional relationship from $\mathbf{V}$ to $\mathbf{k_s}$ and $\vec{f_o}$. In this section, a method to determine a suitable velocity matrix $\mathbf{V}$ from $\mathbf{k_s}$ and $\vec{f_o}$ is derived.

5.1.2 Restating Dynamics

The elements in $\mathbf{k_s}$ and $\vec{f_o}$ can be expressed directly from Equation 18.

$$\begin{bmatrix} f_{o_x} & k_{s_{xx}} & k_{s_{xy}} \\ f_{o_y} & k_{s_{yx}} & k_{s_{yy}} \end{bmatrix} = \mu W \mathbf{V}\mathbf{B}^{\mathrm{T}} \left(\mathbf{B}\mathbf{B}^{\mathrm{T}}\right)^{-1}. \quad (25)$$

This equation can be rewritten in terms of the vector (of length $2n$) formed by stacking the transposes of the two rows of $\mathbf{V}$ ($\vec{V}_x^{\mathrm{T}}$ and $\vec{V}_y^{\mathrm{T}}$). Taking advantage of the symmetry of $\mathbf{B}\mathbf{B}^{\mathrm{T}}$, the following relation holds.

$$\begin{bmatrix} f_{o_x} \\ k_{s_{xx}} \\ k_{s_{xy}} \\ f_{o_y} \\ k_{s_{yx}} \\ k_{s_{yy}} \end{bmatrix} = \mu W \underbrace{\left[\begin{array}{c|c} \left(\mathbf{B}\mathbf{B}^{\mathrm{T}}\right)^{-1}\mathbf{B} & \mathbf{0} \\ \hline \mathbf{0} & \left(\mathbf{B}\mathbf{B}^{\mathrm{T}}\right)^{-1}\mathbf{B} \end{array}\right]}_{\boldsymbol{\mathcal{B}}} \left[\begin{array}{c} \vec{V}_x^{\mathrm{T}} \\ \hline \vec{V}_y^{\mathrm{T}} \end{array}\right] \quad (26)$$

This produces a set of 6 equations and $2n$ unknowns.

5.1.3 Solving for Wheel Speeds

To solve for the wheel speeds ($\vec{V}_x$ and $\vec{V}_y$) it is necessary to invert the previous set of equations. Since this set of equations is underconstrained, there is some freedom in the solution and further constraints are required. The *Penrose pseudo-inverse* accomplishes this by minimizing the sum of the squares of the wheel speeds. For the non-square matrix, $\boldsymbol{\mathcal{B}}$, the Penrose pseudo-inverse, $\boldsymbol{\mathcal{B}}^{\dagger}$ is defined below, and after applying properties of matrix transposes and inverses, $\boldsymbol{\mathcal{B}}^{\dagger}$ becomes

$$\boldsymbol{\mathcal{B}}^{\dagger} \equiv \boldsymbol{\mathcal{B}}^{\mathrm{T}} \left(\boldsymbol{\mathcal{B}}\boldsymbol{\mathcal{B}}^{\mathrm{T}}\right)^{-1} = \left[\begin{array}{c|c} \mathbf{B}^{\mathrm{T}} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{B}^{\mathrm{T}} \end{array}\right]. \quad (27)$$

Therefore, given a set of constants determined by the desired equilibrium and spring constants (with $k_{s_{xy}} = k_{s_{yx}} = 0$ to decouple the x and y motions of the parcel), it is very easy to solve for the set of wheel speeds which will give a parcel the desired dynamics.

$$\left[\begin{array}{c} \vec{V}_x^{\mathrm{T}} \\ \hline \vec{V}_y^{\mathrm{T}} \end{array}\right] = \frac{1}{\mu W} \left[\begin{array}{c|c} \mathbf{B}^{\mathrm{T}} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{B}^{\mathrm{T}} \end{array}\right] \left[\begin{array}{c} f_{o_x} \\ k_{s_{xx}} \\ 0 \\ \hline f_{o_y} \\ 0 \\ k_{s_{yy}} \end{array}\right] \quad (28)$$

Since each row of $\mathbf{B}^{\mathrm{T}}$ contains the vector $\left[1\ x_i\ y_i\right]$, the wheel speeds for each cell can be computed independently as

$$\begin{aligned} V_{x_i} &= f_{o_x} + k_{s_{xx}} x_i, \text{ and} \\ V_{y_i} &= f_{o_y} + k_{s_{yy}} y_i. \end{aligned} \quad (29)$$

In the diagonal $\mathbf{k_s}$ case, $f_{o_x} = -x_{cm_e} k_{s_{xx}}$, and $f_{o_y} = -y_{cm_e} k_{s_{yy}}$, which can be substituted in the previous equation, yielding

$$V_{x_i} = k_{s_{xx}} (x_i - x_{cm_e}), \text{ and} \quad (30)$$
$$V_{y_i} = k_{s_{yy}} (y_i - y_{cm_e}), \quad (31)$$

which is a field where the cells point towards the equilibrium (for negative $k_{s_{xx}}$ and $k_{s_{yy}}$), with velocities of each component proportional to the perpendicular distance to the corresponding axis. Note that this is a discretized version of an elliptic field as described in the continuous medium case by Kavraki [5].

5.2 Rotational constants

5.2.1 Orienting on a Discrete Field

While a parcel rests on a single set of supports, the torque is not a function of orientation. Therefore, it is not possible to construct a static velocity field which will orient a parcel more precisely than its range of motion which keeps it on a single set of supports. This free range of motion is demonstrated by the black rectangle in Figure 12. All that can be assured locally is that the parcel will be in rotational equilibrium when it is in translational equilibrium. Parcels are oriented as they change support from one cell to the next. There are then three considerations for the parcel's orientation: (*i*) Torque is zero when translational force is zero (at $\vec{X}_{cm} = \vec{X}_{cm_e}$). (*ii*) When the parcel rotates about its equilibrium position, a change in supports induces a restoring torque. (*iii*) Given any starting position and equilibrium, the parcel will eventually reach the desired position and orientation. These considerations will be examined under the field generated in the previous section. Even though the previous field was designed without considering orientation, it can be made to orient the parcel as well as position it.

5.2.2 Torque Equilibrium

Without loss of generality, the origin of the coordinate system will be chosen to be at the desired equilibrium position. This sets the terms of $\vec{f}_o$ to zero and simplifies the problem. When the parcel is in translational equilibrium, $\mathbf{k_s}\vec{X}_{cm} + \vec{f}_o = 0$. Also, with our choice of coordinate system, $\vec{X}_{cm} = 0$ at translational equilibrium. Therefore, when the parcel rotates about its translational equilibrium, Equation 23 reduces to

$$\tau = \tau_o - \omega\mu\vec{\mathcal{X}}\vec{N} \quad (32)$$

Which is a constant applied torque with linear damping. For the parcel to be in complete translational and rotational equilibrium, we must then have $\tau_o = 0$. τ_o is determined from the velocity field which was designed to provide the desired translational dynamics. This constant (from Equation 23) can be written as

$$\tau_o = \mu W \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \left(\mathbf{B}\mathbf{B}^{\mathrm{T}}\right)^{-1} \mathbf{B}\vec{R}^{\mathrm{T}}. \quad (33)$$

The vector $\vec{R}^{\mathrm{T}}$ can be expressed in terms of the stacked velocity vector, giving

$$\tau_o = \mu W \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \left(\mathbf{B}\mathbf{B}^{\mathrm{T}}\right)^{-1} \mathbf{B}* \left[\begin{array}{ccc|ccc} -y_1 & & & x_1 & & \\ & \ddots & & & \ddots & \\ & & -y_n & & & x_n \end{array}\right] \begin{bmatrix} \vec{V}_x^{\mathrm{T}} \\ \vec{V}_y^{\mathrm{T}} \end{bmatrix}. \quad (34)$$

Now, the previous velocity field can be substituted in (with $\vec{f}_o = 0$ due to the choice of coordinate system, and with diagonal $\mathbf{k_s}$).

$$\tau_o = \mu W \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \left(\mathbf{B}\mathbf{B}^{\mathrm{T}}\right)^{-1} \mathbf{B}* \left[\begin{array}{ccc|ccc} -y_1 & & & x_1 & & \\ & \ddots & & & \ddots & \\ & & -y_n & & & x_n \end{array}\right] \left[\begin{array}{c|c} \mathbf{B}^{\mathrm{T}} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{B}^{\mathrm{T}} \end{array}\right] \begin{bmatrix} 0 \\ k_{s_{xx}} \\ 0 \\ \hline 0 \\ 0 \\ k_{s_{yy}} \end{bmatrix} \quad (35)$$

The rows of $\mathbf{B}$ are the ones vector, the vector of x positions, and the vector of y positions. Therefore, the terms produced by multiplying $\mathbf{B}$ with other vectors and matrices form sums of the x and y components of all the cell locations. For example,

$$\mathbf{B}\mathbf{B}^{\mathrm{T}} = \begin{bmatrix} n & \sum x_i & \sum y_i \\ \sum x_i & \sum {x_i}^2 & \sum x_i y_i \\ \sum y_i & \sum x_i y_i & \sum {y_i}^2 \end{bmatrix}. \quad (36)$$

Realizing this, Equation 35 can be rewritten in terms of these sums. After some algebra,

$$\tau_o = \mu W \begin{bmatrix}1 & 0 & 0\end{bmatrix} * \begin{bmatrix} n & \sum x_i & \sum y_i \\ \sum x_i & \sum {x_i}^2 & \sum x_i y_i \\ \sum y_i & \sum x_i y_i & \sum {y_i}^2 \end{bmatrix}^{-1} \begin{bmatrix} \sum x_i y_i \\ \sum {x_i}^2 y_i \\ \sum x_i {y_i}^2 \end{bmatrix} (k_{s_{yy}} - k_{s_{xx}}) \quad (37)$$

In general, the torque resulting from these constants evaluated at the equilibrium position is not zero. However, consider the case where the cells on which a parcel rests are arranged *symmetrically* (mirrored in x and y) about the coordinate axes. Therefore, any term with odd powers of x_i or y_i in a summation will be identically zero. For example, in $\sum x_i y_i$, cells in the first and fourth quadrants cancel cells in the second and third quadrants, making $\sum x_i y_i = 0$. Similarly, $\sum {x_i}^2 y_i = 0$ and $\sum x_i {y_i}^2 = 0$. Therefore, Equation 37 reduces to

$$\tau_o = 0 \quad (38)$$

such that the parcel will be in rotational equilibrium when resting on a mirror-symmetric set of supports at the translational equilibrium.

5.2.3 Rotation About Equilibrium

The symmetric arrangement of cells under the parcel depends on the parcel's shape and orientation. Figure 12 shows a rectangular parcel in three orientations at its equilibrium position with arrows at each cell indicating the magnitudes of the velocities at each cell. In this figure, we can see that the symmetrically oriented parcel (solid black line) has a symmetric set of forces, and does not feel a torque. Similarly, the slightly perturbed parcel (dashed black line) also has a symmetric set of forces and does not feel a torque. However, the parcel which has rotated enough to change supports (dotted black line) has a set of supports which is symmetric about the origin (radially symmetric) rather than the coordinate axes and will feel a torque.

In general, Equation 37 shows that for a given arrangement of cells and a particular orientation of parcel, the torque will be either positive or negative depending on the relative magnitude of $k_{s_{yy}}$ and $k_{s_{xx}}$. Therefore, equilibrium is either stable or unstable (within cell resolution) depending on the values of these constants. If $k_{s_{xx}} = k_{s_{yy}}$ there will be no restoring torque when the parcel rotates enough to shift cells.

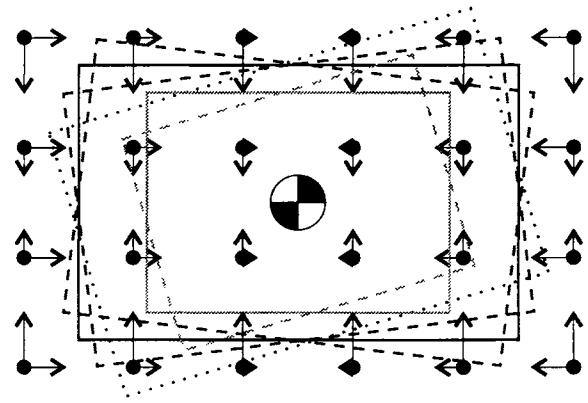


Figure 12: *Rotation of two parcels (black and grey) about their equilibrium positions. Black dashed lines represent the free (unorientable) range of motion of the black parcel. The grey parcel does not satisfy the "positive rotation" property. Note that $k_{s_{xx}}, k_{s_{yy}} < 0$ and $(k_{s_{yy}} - k_{s_{xx}}) < 0$.*

If the parcel is mirror-symmetric itself, a stronger statement can be made about the direction of rotation. The resulting set of supports will be radially symmetric such that for every cell (x_i, y_i) there is a cell $(-x_i, -y_i)$. Thus, many of the terms in Equation 37 become zero.

$$\tau_o = \mu W \begin{bmatrix}1 & 0 & 0\end{bmatrix} * \begin{bmatrix} n & 0 & 0 \\ 0 & \sum {x_i}^2 & \sum x_i y_i \\ 0 & \sum x_i y_i & \sum {y_i}^2 \end{bmatrix}^{-1} \begin{bmatrix} \sum x_i y_i \\ 0 \\ 0 \end{bmatrix} (k_{s_{yy}} - k_{s_{xx}})$$
$$= \frac{1}{n} \sum x_i y_i \left(k_{s_{yy}} - k_{s_{xx}}\right). \quad (39)$$

A final assumption can be made. When the symmetric parcel rotates counterclockwise about the equilibrium position, $\sum x_i y_i > 0$. This is often true, since more of the parcel is in the first and third quadrants, and more first and third quadrant cells (giving $x_i y_i > 0$) are covered. The black rectangle in Figure 12 shows a parcel with this property. However, because of the discreteness of the array, some parcels (for example, the grey rectangle in Figure 12) may have a negative $\sum x_i y_i$ for some counterclockwise rotations. A particular parcel can be checked for "orientability" on the array by analyzing it as it rotates about equilibrium and to see which cells it covers. For parcels with this

"positive rotation" property, there will be a restoring torque for $\left(k_{s_{yy}} - k_{s_{xx}}\right) < 0$ with a stable orientation.

5.2.4 Arbitrary Starting Locations

The previous rotational analysis examined a symmetric parcel whose center of mass is at its equilibrium position, and the cells are arranged symmetrically (such as in a regular array) about this equilibrium. Nothing was said for torques on the parcel when the position is not at the equilibrium. In order to assure proper orientation, the parcel must first reach its equilibrium and then orient itself.

Reaching the equilibrium is guaranteed regardless of orientation and cell distribution. Since the parcel's dynamics are that of a mass-spring-damper centered at the origin independent of which cells are supporting it, the parcel will eventually settle to its positional equilibrium. When the orientation changes, changing the supports, the translational dynamics are unaffected.

Once position equilibrium is reached, a torque will be applied which tends to orient the parcel. For $\left(k_{s_{yy}} - k_{s_{xx}}\right) < 0$, a symmetric parcel will rotate until it is aligned with the coordinate axes. This will happen as long as more first and third quadrant cells are covered when the parcel rotates counterclockwise.

6 Conclusions and Future Work

In this paper, a simplified computation of the dynamics of a parcel was done for the MDMS array. This analysis involved computing the distribution of the weight of a flat parcel over an arbitrary discrete set of flexible supports. The interaction between the constant-speed wheels and the parcel was modeled using a viscous friction model. A matrix formulation was used to sum up the forces on the parcel, and the resulting motion of the parcel is that of a mass-spring-damper, with spring, offset, and damping constant over a particular set of supports.

A standard wheel velocity field was methodically designed to provide uniform, arbitrary spring constants and equilibrium position over the entire array regardless of which cells support the parcel. A pseudo-inverse was used to solve this underconstrained problem which provides the desired behavior while minimizing wheel speed energy. The resulting field, when only the non-circulatory spring constants were used, was an inward-pointing field with each wheel's velocity proportional to its the corresponding component of perpendicular distance to the equilibrium position.

The derived field provides uniform, smooth motion for the parcel even though the supporting forces and points of actuation change discontinuously. This is because the moment equilibrium of the parcel uses the positions of the cells as moment arms, just as the velocities in the field vary linearly with cell position. When the parcel changes supports, the normal forces change in proportion with the wheel velocities providing a continuous horizontal force.

This field was then analyzed for its rotational equilibrium properties. Interestingly, the torque on the parcel is not a function of the orientation because the supporting forces only depend on the position of the center of mass. Therefore, the transitions from cell to cell as the parcel rotates about its equilibrium are used to orient the parcel within the resolution limits of the discrete array. Given the assumption that the set of cells supporting the parcel at equilibrium is mirror-symmetric about the coordinate axes, it was found that the parcel will be in rotational equilibrium at the translational equilibrium. A ramification of this assumption on a regular square-lattice array, such as the MDMS, is that the equilibrium position must be set either exactly on a cell or exactly midway between cells. Also, a parcel can be oriented only to angles of $\frac{n\pi}{4}$, for $n = 0, \ldots, 3$.

The torque developed by the inward-pointing field was analyzed as the parcel rotates about equilibrium. It was determined that, as long as the two spring constants are not equal, when the parcel rotates and the supports change, there will be a restoring torque on a symmetric parcel with the property that $\sum x_i y_i > 0$ for all the cells under the parcel when the parcel rotates counterclockwise. This positive rotation property is an artifact of the discrete array and holds for many parcel sizes and shapes. Furthermore, in the the limit of many cells spaced closely together (e.g., a continuous array), this property always holds for symmetric parcels, as

implied by Kavraki. Finally, a simple analysis can be done to check if a parcel meets this assumption.

Physical intuition can be applied to explain the restoring torque. For example, consider a cell in the first quadrant with coordinates (x, y). This cell will apply a force in the direction $(k_{s_{xx}}x, k_{s_{yy}}y)$ (where $k_{s_{xx}}, k_{s_{yy}} < 0$.) The moment about the origin of this force will be $-k_{s_{xx}}xy + k_{s_{yy}}xy = (k_{s_{yy}} - k_{s_{xx}})xy$. If $(k_{s_{yy}} - k_{s_{xx}}) < 0$, first quadrant cells apply a negative torque, as will third quadrant cells, while second and fourth quadrant cells apply a positive torque. If the parcel rests on a mirror-symmetric set of cells, these torques will cancel. If the parcel rotates, there will be a restoring torque if the first and third quadrant cells outweigh the second and fourth quadrant cells, or, if $\sum x_i y_i > 0$.

When the approach described in this paper is applied to generate a specific vector field that has a single equilibrium position and orientation (to symmetry), this method results in a discretized version of the elliptic potential fields examined by Kavraki. However, the general framework here considers the distribution of weight among supports, while at the same time models the interaction between parcel and actuators, and accounts for the discreteness of the array. This is an extension of the work done by Kavraki and others like Böhringer, who do not need to consider these issues because of the scale of their tasks.

Many of the assumptions, such as symmetry, made in this paper are not necessary and may be relaxed in the future. The positive rotation property will be examined, and a more definite statement about parcel orientability will be made. Another issue for future research is the relationship between the discretized field and the continuous field. Many intuitive ideas operate on both fields, but it is difficult to analyze one in terms of the other. A more unified approach to handle both types of fields is desirable.

In general, there are limitations to a passive (open-loop) policy such as this, particularly on a discrete array where positioning and orientation can only be done within the resolution of the array. An active policy which updates wheel velocities base on feedback from the motions of the parcel would be necessary to more precisely manipulate parcels. Also, the policies in this paper rely on a full-slip contact between the wheels and parcel. A full-stick contact would be better suited to precise manipulation with an active policy in order to have better control over the parcel.

References

[1] S. Akella, W. Huang, K. Lynch, and M. Mason. Sensorless parts feeding with a one joint robot. In *Proceedings. Workshop on the Algorithmic Fundamentals of Robotics*, 1996.

[2] K. Böhringer, B. Donald, R. Mihailovich, and N. MacDonald. A theory of manipulation and control for microfabricated actuator arrays. In *Proceedings. IEEE International Conference on Robotics and Automation*, 1994.

[3] M. Erdmann. An exploration of nonprehensile two-palm manipulation: Planning and execution. In *Proceedings. Seventh International Symposium on Robotics Research*, 1995.

[4] K. Goldberg. Orienting polygonal parts without sensors. *Algorithmica: Special Issue on Computational Robotics*, 10:201–225, August 1993.

[5] L. Kavraki. Part orientation with programmable vector fields: Two stable equilibria for most parts. In *Proceedings. IEEE International Conference on Robotics and Automation*, 1997.

[6] J. Luntz, W. Messner, and H. Choset. Parcel manipulation and dynamics with a distributed actuator array: The virtual vehicle. In *Proceedings. IEEE International Conference on Robotics and Automation*, 1997.

[7] J. Luntz, W. Messner, and H. Choset. Virtual vehicle: parcel manipulation and dynamics with a distributed actuator array. In *Proceedings of SPIE vol. 3201. Sensors and Controls for Advanced Manufacturing. International Symposium on Intelligent Systems and Advanced Maufacturing.*, 1997.

Taylor & Francis
Taylor & Francis Group
http://taylorandfrancis.com

The Complexity of the Two Dimensional Curvature-Constrained Shortest-Path Problem

John Reif, *Duke University, Durham, NC, USA*
Hongyan Wang, *Duke University, Durham, NC, USA*

The curvature-constrained shortest-path problem is to plan a path (from an initial position to a final position, where a position is defined by a location and the orientation angle) in the presence of obstacles, such that the path is the shortest among all paths with a prescribed curvature bound. This paper shows that the above problem in two dimensions is NP-hard, when the obstacles are polygons with a total of N vertices and the vertex positions are given with $N^{O(1)}$ bits. Our reduction is computed by a family of polynomial-size circuits. Previously, there is no known hardness result for the 2D curvature constrained shortest-path problem.

1 Introduction

A *curvature-constrained path* is a path whose curvature at any point along the path is no larger than a prescribed upper bound. Curvature-constrained path planning arises in a variety of robotics problems, e.g. the motions of robot vehicles controlled by steering mechanisms. This paper studies the complexity of planning curvature-constrained paths.

To be more precise, let $P : I \rightarrow R^d$ be a d-dimensional continuous differential path parameterized by arc length $s \in I$. The *average curvature* of P in the interval $[s_1, s_2] \subseteq I$ is defined by $\|P'(s_1) - P'(s_2)\|/|s_1 - s_2|$. A path has an upper-bounded curvature of C if its average curvature is at most C in every interval of the path. A *position* is a tuple (p, θ), where p is a d-dimensional vector specifying the location, and θ specifies the orientation (in d-dimensions, an angle suffices). Given a curvature bound, an initial position, a final position and a set of obstacles, the *curvature-constrained shortest-path problem* is to find a path from the initial position to the final position such that the path is the shortest among all curvature-constrained paths connecting the initial position to the final position.

In the rest of the paper, without further mentioning, when we talk about paths we mean paths that satisfy the curvature constraints.

For the case of a region without obstacles, Dubins [4] characterized the shortest paths in 2D, and Sussmann [9] recently extended the characterization for shortest paths in 3D. For the case of polygonal obstacles in 2D, Fortune and Wilfong [5] gave the first algorithm which computes the exact shortest paths avoiding obstacles. However, their algorithm required super-exponential time. A polynomial-time approximation algorithm was given by Jacobs and Canny [6], whose running time is $O(N^3)$ (N is the number of obstacle vertices) and depends on the obstacle size. Their approximation algorithm was later improved by Wang and Agarwal [8] to reduce the running time to $O(N^2)$. Also the running time of their algorithm does not depend on the obstacle size. Agarwal, Raghavan, and Tamaki [1] also developed efficient approximation algorithms for the 2D shortest paths for a restricted class of obstacles (moderate obstacles). Reif and Wang [7] developed non-uniform discretization approximations for 3D kinodynamic motion planning and 3D curvature-constrained shortest paths, with considerably smaller computational complexity.

Canny and Reif [3] proved that the problem of finding a shortest 3-dimensional path, with no curvature constraints, and avoiding polygonal obstacles, is NP-hard. (Subsequently, Asano, Kirkpatrick and Yap [2] proved using similar techniques the NP-hardness of optimal motion of a rod on a plane with polygonal ob-

stacles.) As a consequence of the above result, the curvature-constrained shortest-path problem in 3D is at least NP-hard. Although the problem in 2D is long conjectured to be a hard problem, its complexity is never stated or proved. This paper provides the first known complexity result for the curvature-constrained shortest-path problem in 2D. We show that if the number of obstacle vertices N is an input parameter and the vertex positions are given with $N^{O(1)}$ bits, this problem is NP-hard even in 2D.

Our approach to the complexity result is similar to the *path-encoding* approach developed in [3] (while the detailed techniques are quite different). This approach is to encode the boolean assignments by paths and then to reduce the 3-SAT problem to finding the shortest path. Recall that a 3-SAT formula in n variables $X_1, \ldots, X_n$ has the form

$$\bigwedge_{i=1,\ldots,m} C_i,$$

where each C_i is a clause of the form $(l_{i1} \vee l_{i2} \vee l_{i3})$. Each l_{ij} in turn is either a variable X_k or a negation of it. The 3-SAT problem is to decide whether there is a boolean assignment to $X_1, \ldots, X_n$ such that the formula is satisfied.

There are basically three steps in the path-encoding approach. First we arrange the obstacles such that there are 2^N shortest paths, where $N = 2m + n$. Each path encodes a different boolean assignment. (In Section 4.2 we will see why it is not enough with 2^n paths.)

For a clause, we can construct the obstacles (to a "clause filter") such that the path whose encoding does not satisfy this clause is stretched a little longer than those paths with satisfying encodings. We can arrange the obstacles such that all the 2^N paths pass through all m "clause filters". The 3-SAT formula is satisfiable if and only if there exists a path whose length is not stretched more than a certain amount (the details will be in later sections).

The third and final step is to have a reverse of the obstacle arrangement so far. This will make all the 2^N paths end at the same location with the same orientation, which is the final position. From the construction, it can be shown that the total number of obstacle vertices used in the above three steps is $N^{O(1)}$.

Our techniques for generating and encoding the paths are quite different from those of [3]. In [3], 3D obstacles are used and a 3D path is represented by the obstacle edges it passes through. In our case, the paths are 2 dimensional and is represented by its orientation at certain common locations. In [3], filtering the 3 literals in a clause can be done in parallel because it is in 3D, while we have to use a more complicated encoding to emulate this in our 2D construction.

The rest of the paper is organized in the following way. In Section 2 we describe the *basic gadget*, an arrangement of some obstacles, which is the building block of our obstacle construction. In Section 3 we show how to use the basic gadgets to construct what we call *splitters* to generate 2^N shortest paths. In Section 4 we show in detail the reduction from the 3-SAT problem to finding the shortest path. In Section 5 we show that the obstacle arrangement does not only exists theoretically but we can compute or at least approximate up to the necessary precision one such arrangement.

In the rest of the paper, we use the following notations. A 3-SAT problem involves m clauses in n boolean variables, $X_1, \ldots, X_n$. N is $2m + n$. $B = N^c$ is the number of bits of precision, where c is some constant. Without loss of generality, we can set the curvature bound to be 1.

2 Basic Gadgets and Their Properties

2.1 A basic gadget

A *basic gadget* $\mathcal{B}$ (see Figure 1) consists of line-segment obstacles. These line segments form the boundary of two squares joined together at point v. There are "holes" at the start point s, the middle point v and the target point t such that the point robot can pass through. The purpose of a basic gadget is to force a robot to pass through point v to get to point t once it enters the first square at point s. For a basic gadget, we also require that $d(s, v) = d(v, t)$ and that the angle $\angle svt = 3\pi/4$. There are two types of basic gadget depending on how the angle $\angle svt$ is counted. If the angle

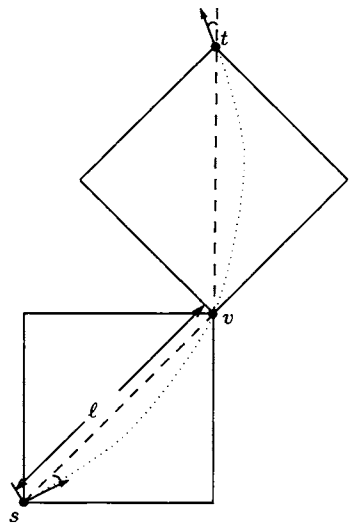


Figure 1: *A basic gadget.*

is counted clockwise from segment mv to segment vt, then it's a type-1 basic gadget. If the angle is counted counter-clockwise, the basic gadget is of type 2. A basic gadget is denoted by $\mathcal{B}(\ell)$ if $d(s,v) = d(v,t) = \ell$. Except in Section 5, all the basic gadgets in discussion are of the same fixed ℓ.

Let $\overrightarrow{sv}$ be the ray that starts at point s, passes through point v and has the direction of going from s to v. If the robot enters the first square at position (s,θ), then the *input ray* is defined to be the ray that starts at s and has an orientation of θ. The *input angle* is defined to be the clockwise angle from $\overrightarrow{sv}$ to the input ray for a type-1 basic gadget, and the counter-clockwise one for a type-2 basic gadget. It is implied that the range of the input angle is $[0,\pi/4]$. Let $\overrightarrow{vt}$ be the ray that starts at point t, aligns with segment vt and has the direction of going from v to t. If the robot exits the second square at position (t,θ'), the *output ray* is defined to be the ray that starts at t and has on orientation angle θ'. The *output angle* is defined to be the counter-clockwise angle from $\overrightarrow{vt}$ to the output ray for a type-1 basic gadget, and the clockwise one for a type-2 basic gadget.

Let $P(\theta)$ be the shortest path from s to t with an input angle of θ, and let $\rho(\theta)$ and $\psi(\theta)$ be the path length and the output angle of $P(\theta)$, respectively. Let $\bar{\theta}$ be such that

$$\rho(\bar{\theta}) = \min_{\theta_i \in [0,\pi/4]} \rho(\theta_i). \tag{1}$$

Therefore $P(\bar{\theta})$ is the shortest path from s to t, among all paths with different input angles. We call $P(\bar{\theta})$ the *canonical path* of the basic gadget. Let $\psi(\bar{\theta}) = \bar{\theta}'$ be the output angle of canonical path $P(\bar{\theta})$.

2.2 A gadget sequence

Let $\mathcal{B}_1$ and $\mathcal{B}_2$ be two basic gadgets. $\mathcal{B}_2$ is a *sequel* of $\mathcal{B}_1$ if the start point of $\mathcal{B}_2$ coincides with the end point of $\mathcal{B}_1$ and $\mathcal{B}_1, \mathcal{B}_2$ are rotationally oriented so that the output angle $\bar{\theta}'$ from $\mathcal{B}_1$ corresponds to input angle $\bar{\theta}$ to $\mathcal{B}_2$ (that is, the output ray of the canonical path of $\mathcal{B}_1$ is the same as the input ray of the canonical path of $\mathcal{B}_2$).

A *gadget sequence* of length ℓ, S_ℓ, is a sequence of gadgets $\mathcal{B}_i$ for $i = 1,\ldots,\ell$, such that $\mathcal{B}_{i+1}$ is a sequel of $\mathcal{B}_i$, for $i = i,\ldots,\ell-1$.

The major part of the lower-bound proof is to show that by using $O(N)$ basic gadgets, we can construct 2^N different paths from the initial state to the final state such that these 2^N paths are all the shortest between the initial state and the final state up to $B = N^c$ bits of precision.

As we will see later in our construction, at some stages, all paths generated so far pass through the target point of the same basic gadget. Thus the paths can not be distinguished by their locations. They have to be distinguished by their orientations. Since all the paths exit through the same gadget, it implies that they can be distinguished by the output angles. Furthermore, since the output ray of canonical path $P(\bar{\theta})$ is fixed, an output angle of a path $P(\theta)$ can be represented by the angle from the output ray of $P(\bar{\theta})$ to the output ray of $P(\theta)$. We call the angle from an output ray of canonical path $P(\bar{\theta})$ to the output ray of a given path $P(\theta)$ the *difference angle.*

Define $f(x) = \psi(\bar{\theta}+x) - \bar{\theta}'$. What this function does is the following. Let θ be the input angle of a given path $P(\theta)$ and $x = \theta - \bar{\theta}$ be the angle from the input ray

of $P(\theta)$ to the input ray of canonical path $P(\bar{\theta})$. Note that $x > 0$ if the angle is clockwise and $x < 0$ if the angle is counter-clockwise. Then $f(x) = \psi(\theta) - \bar{\theta}'$ is the angle from the output ray of $P(\bar{\theta})$ to the output ray of $P(\theta)$. $f(x) > 0$ if the angle is counter-clockwise and $f(x) < 0$ otherwise. Define $g(x) = f(-x)$.

Define

$$F(S, x) = F^{(\ell)}(x) = F(F(\ldots F(x))\ldots), \qquad (2)$$

such that the ith F (from right) is $f(x)$ if $\mathcal{B}_i$ is a type-1 sequel of $\mathcal{B}_{i-1}$ and $g(x)$ if the sequel is type 2, for $i = 2, \ldots, \ell$. Therefore $F(S, x)$ computes the difference angle at the end of a gadget sequence S.

When $|x|$ is close enough to 0, $f(x)$ can be represented by a converging Taylor series as,

$$f(x) = c_0 + \sum_{j=1}^{\lambda} c_j x^j + O(2^{-B}),$$

where $\lambda > 0$ is an integer and that $\sum_{j>\lambda} c_j x^j = O(2^{-B})$, for any x in a predetermined range. Since $f(0) = 0$, $c_0 = 0$. Let k be the integer such that $c_i = 0$, for $i = 1, \ldots, k-1$, and $c_k \neq 0$.

Let θ_1 and θ_2 be two different input angles and are represented with $B = N^c$ bits of precision. Let δ_i be the output difference angle (from the canonical path output angle) for a path with input angle θ_i after a gadget sequence of length ℓ, for $i = 1, 2$. It is easy to see that to distinguish δ_1 and δ_2, it requires $\Omega(Bk^\ell)$ bits of precision. The number of bits required is $\Omega(Bk^N)$ to represent difference angles after a gadget sequence of $O(N)$. If $k > 1$ we are in trouble since we can only afford $poly(N)$ number of bits.

Instead of showing that $k = 1$, we do the following trick to deal with the case when $k > 1$. We rotate a basic gadget around the entering point by an angle of ϵ. By doing this, we change an entering angle of x to $x+\epsilon$. Let h be a map that maps an entering angle before the rotation to an exiting angle (after the rotation). Thus,

$$\begin{aligned}
h(x) &= f(x+\epsilon) \\
&= \sum_{j=1}^{\lambda} c_j (x+\epsilon)^j \\
&= \sum_{j=1}^{\lambda} c_j \epsilon^j + \left(\sum_{j=1}^{\lambda} j c_j \epsilon^{j-1}\right) x \\
&\quad + \left(\sum_{j=1}^{\lambda} j(j-1) c_j \epsilon^{j-2}/2\right) x^2 + \ldots \\
&= d_0(\epsilon) + d_1(\epsilon)x + d_2(\epsilon)x^2 + \ldots \\
&= d_0(\epsilon) + d_1(\epsilon)x + O(x^2).
\end{aligned}$$

We can choose an ϵ, such that the coefficient $d_1(\epsilon) \neq 0$. (In fact, we can choose an ϵ randomly within a range and with high likelihood$d_1(\epsilon) \neq 0$.) Once an ϵ is chosen, d_i are constants, for $i = 0, 1, \ldots$, even though they may be very small.

The difference angle after a 2-gadget sequence is

$$\begin{aligned}
h^{(2)}(x) &= d_0(\epsilon) + d_1(\epsilon)(d_0(\epsilon) + d_1(\epsilon)x + \epsilon) \\
&= d_0^{(2)}(\epsilon) + d_1^{(2)}(\epsilon)x + O(x^2),
\end{aligned}$$

where $h^{(2)}$ means $h(h(x))$,

$$d_0^{(2)}(\epsilon) = d_0(\epsilon) + d_0(\epsilon)d_1(\epsilon) + \epsilon d_1(\epsilon),$$

and

$$d_1^{(2)}(\epsilon) = d_1(\epsilon)d_1(\epsilon),$$

so the coefficient $d_1^{(2)}(\epsilon) \neq 0$.

The difference angle after an ℓ-gadget sequence is

$$h^{(\ell)}(x) = d_0^{(\ell)}(\epsilon) + d_1^{(\ell)}(\epsilon)x + O(x^2), \qquad (3)$$

where again the coefficient $d_1^{(\ell)}(\epsilon) \neq 0$.

$d_0^{(\ell)}$ and $d_1^{(\ell)}$ may be a very complicated functions of ϵ. But ϵ is fixed and we only need to provide up to B bits of precision.

2.3 Canonical and semi-canonical paths of gadget sequences

Recall the *canonical path* of a basic gadget is $P(\bar{\theta})$, where $\bar{\theta}$ is as defined in (1). By the definition of a

gadget sequence, the canonical paths of the gadgets in the sequence form a valid path. We call this path the *canonical path* of a gadget sequence. Furthermore,

Lemma 1 *The canonical path of a gadget sequence S is the shortest path from the start point s of S to the end point t of S, among all paths from (s, θ_1) to (t, θ_2), where $\theta_i \in [0, \pi/4]$. Furthermore, the length of the canonical path is $\ell\bar{\rho}$, where ℓ is the length of S and $\bar{\rho} = \rho(\bar{\theta})$.*

There exists a small real number $\epsilon_1 > 0$, such that if $-\epsilon_1 < \theta - \bar{\theta} < \epsilon_1$, then $\rho(\theta)$ can be represented by a converging Taylor series,

$$\rho(\theta) = \bar{\rho} + \sum_{j=1}^{\mu} d_j(\theta - \bar{\theta})^j + O(2^{-B}),$$

where $\mu > 0$ is some integer and that $\sum_{j>\mu} d_j(\theta - \bar{\theta})^j = O(2^{-B})$ for all $\theta \in [\bar{\theta} - \epsilon_1, \bar{\theta} + \epsilon_1]$, and d_j are reals. Since $\rho(\bar{\theta})$ is a minimum,

$$\frac{d\rho}{d\theta}|_{\theta=\bar{\theta}} = 0.$$

This implies that $d_1 = 0$.

There exists $\epsilon_2 > 0$, such that $\epsilon_2 < \epsilon_1$ and that

$$\sum_{j=2}^{j=\mu} (\epsilon_2)^j < 2^{-B}. \tag{4}$$

This implies that if the input angle θ satisfies $|\theta - \bar{\theta}| < \epsilon_2$, then the length of $P(\theta)$ is the same as $\bar{\rho}$, up to the allowed B bits of precision. Furthermore, there exists $0 < \epsilon_\ell < \epsilon_2$, such that the accumulation of (4) over a gadget sequence of length N is no more than 2^{-B}. In this case, $P(\theta)$ is called a *semi-canonical path.*

2.4 A variant gadget

The purpose of a variant gadget $\mathcal{V}(\delta)$ is to force the output orientation to differ from the output orientation of a basic gadget $\mathcal{B}$ by some amount distinguishable with B bits of precision. It is constructed in the following way. Let $\mathcal{B}$ be a basic gadget and let s, v and t be the start point, the middle point and the target point respectively; see Figure 2. Let ℓ be a line such that the angle between segment vt and ℓ is $\bar{\theta}$. Let s', v' and t' be the start point, the middle point and the target point of $\mathcal{V}(\delta)$ respectively. s' coincides with s, v' lies on the segment sv, t' lies on the line ℓ, $|s'v'| = |v't'|$ and the angle between segment vt and segment $v't'$ is δ.

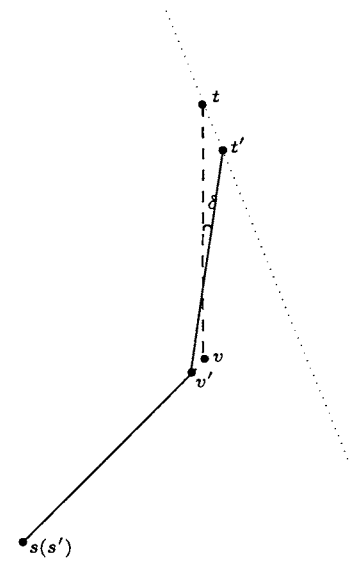


Figure 2: *A variant gadget $\mathcal{V}(\delta)$.*

We can define functions $\rho_\delta(\theta)$ and $\psi_\delta(\theta)$, similar to functions $\rho(\theta)$ and $\psi(\theta)$ of the basic gadget. Let $\bar{\theta}_\delta$ be the input angle that gives the shortest path from s' to t'. Similarly, we have

$$\rho_\delta(\bar{\theta}_\delta + \gamma) = \bar{\rho}_\delta + d' \cdot \gamma^2 + O(\gamma^3), \tag{5}$$

and

$$\psi_\delta(\bar{\theta}_\delta + \gamma) = \bar{\theta}_\delta + c' \cdot \gamma + O(\gamma^2), \tag{6}$$

where d', and c' are some constants.

3 Generating 2^N Paths

3.1 A merger

A *merger* consists of an upper half and a lower half. The upper half is a concatenation of 4 basic gadgets $\mathcal{B}_i$, for $i = 1, \ldots, 4$, as follows (see Figure 3). We place $\mathcal{B}_2$ in a way such that the start point s_2 of $\mathcal{B}_2$ coincides with the target point t_1 of $\mathcal{B}_1$, and the segments v_1t_1

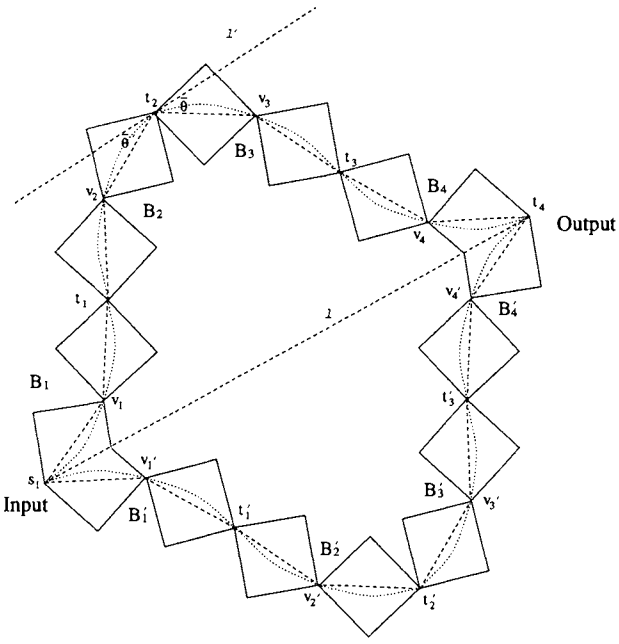


Figure 3: *A merger.*

and t_1v_2 are aligned so that $\angle v_1t_1v_2 = \pi+\bar{\theta}'-\bar{\theta}$. Hence an output angle $\bar{\theta}'$ from gadget $\mathcal{B}_1$ corresponds to input angle $\bar{\theta}$ into gadget $\mathcal{B}_2$, so $\mathcal{B}_2$ is a sequel to $\mathcal{B}_1$.

Let ℓ be the line passing through s_1 and forming an angle of $\bar{\theta}$ with segment s_1v_1. Basic gadgets $\mathcal{B}_3$ and $\mathcal{B}_4$ are placed such that the start point s_3 of $\mathcal{B}_3$ coincides with the target point t_2 of $\mathcal{B}_2$, the start point s_4 of $\mathcal{B}_4$ coincides with the target point t_3 of $\mathcal{B}_3$, and the segments v_3t_3 and t_3v_4 are aligned so that $\angle v_3t_3v_4 = \pi+\bar{\theta}'-\bar{\theta}$, and ℓ forms an angle of $\bar{\theta}'$ with segment v_4t_4. Hence, an output angle $\bar{\theta}'$ from gadget $\mathcal{B}_3$ corresponds to input angle $\bar{\theta}$ into gadget $\mathcal{B}_4$, so $\mathcal{B}_4$ is a sequel to $\mathcal{B}_3$.

The lower half of the merger consists of 4 basic gadgets $\mathcal{B}_i'$, where each $\mathcal{B}_i'$ is symmetric to $\mathcal{B}_i$ with respect to line ℓ.

Since $\angle s_1v_1t_1 = \angle t_1v_2t_2 = 3\pi/4$, and $\angle v_1t_1v_2 = \pi+(\bar{\theta}'-\bar{\theta})$, it follows that segments v_2t_2 and s_1v_1 have angle difference $\bar{\theta}'-\bar{\theta}$. Similarly, segments t_2v_3 and v_4t_4 have angle difference $\bar{\theta}'-\bar{\theta}$. Let ℓ' be the line parallel to ℓ passing through t_2. Thus the angle between ℓ' and segment v_2t_2 is $\bar{\theta}'$ and the angle between ℓ' and segment v_3t_2 is $\bar{\theta}$. Hence an output angle $\bar{\theta}'$ from gadget $\mathcal{B}_2$ corresponds to input angle $\bar{\theta}$ into gadget $\mathcal{B}_3$, so $\mathcal{B}_3$ is a sequel to $\mathcal{B}_2$. Thus we have that each $\mathcal{B}_i$ is a sequel to $\mathcal{B}_{i-1}$, for $i = 2,\ldots,4$. This implies that $\mathcal{B}_i$ (similarly $\mathcal{B}_i'$), for $i = 1,\ldots,4$, form a 4-gadget sequence.

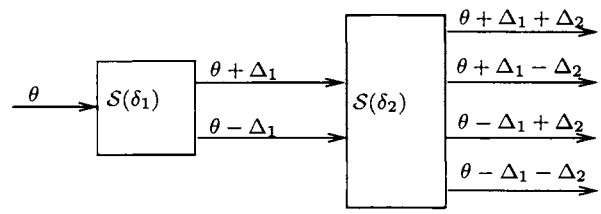


Figure 4: *The scheme of using splitters to generate 2^k orientations and paths.*

Since $\mathcal{B}_1'$ is symmetric to $\mathcal{B}_1$ with respect to the line ℓ, $\angle v_1s_1v_1' = 2\bar{\theta}'$. This implies that an input angle of θ to gadget $\mathcal{B}_1$ is an also a valid input angle of $2\bar{\theta}-\theta$ to gadget $\mathcal{B}_1'$. A path from point s_1 can either go through the upper half or the lower half of the merger to point t_4. However, tracking the changes in orientations, we can see that the two paths merge at t_4, because they have the same orientation at this point. Therefore, a merger by itself does not double the number of paths.

3.2 A splitter

By replacing gadgets $\mathcal{B}_4$ and $\mathcal{B}_4'$ in the merger with variant gadgets $\mathcal{V}(\delta)$, we obtain a *splitter* $\mathcal{S}(\delta)$. A path is doubled when it goes through a splitter. This is because when the sub-path that goes through the upper half of the splitter and the sub-path that goes through the lower half meet at point t_4, their orientations differ due to the variant gadgets. Note that the difference in orientations is controlled by the parameter δ. There exists a set $\{\delta_1,\ldots,\delta_N\}$, such that a path is repeatedly doubled to 2^N paths after going through N splitters $\mathcal{S}(\delta_i)$. The scheme of generating 2^k orientations is shown in Figure 4. Even though we don't know how to compute these δ_i's, we can consider each of them a size B circuit.

4 Reduction from 3-SAT

Recall that a 3-SAT formula in n variables $X_1, \ldots, X_n$ has the form

$$\bigwedge_{i=1,\ldots,m} C_i,$$

where each clause C_i is of the form $(l_{i1} \vee l_{i2} \vee l_{i3})$. Each l_{ij} in turn is either a variable X_k or a negation of it. The 3-SAT problem is to decide whether there is a boolean assignment to $X_1, \ldots, X_n$ such that the formula is satisfied.

This section describes the construction of a curvature-constrained shortest-path problem derived from the input 3-SAT formula. It will have polygonal obstacles defined by a total of $O(N)$ vertices (where $N = 2m + n$), each of which will be specified within 2^{cN} precision, for a constant $c \geq 1$. Thus the resulting curvature-constrained shortest-path problem will be specified by a total of $N^{O(1)}$ bits. This reduction from the input 3−SAT formula to our resulting curvature-constrained shortest-path problem will be computed in $N^{O(1)}$ time, with the exception of certain coefficients which depend on N. Therefor, our reduction can be computed by a family of polynomial-size circuits, where for each $N = 2m + n$, the Nth circuit of the family computes the reduction from any input 3-SAT formula to our resulting curvature-constrained shortest-path problem.

In the previous section we have constructed gadgets to generate 2^N shortest paths. All the paths end at the same locations but with different orientations. We number the paths $0, \ldots, 2^N - 1$, ordered by their orientations. We introduce extra $2m$ boolean variables $Y_1, Z_1, \ldots, Y_m, Z_m$. Path i is considered as boolean assignment to $n + 2m$ variables $Y_1 Z_1 \ldots Y_m Z_m X_1 \ldots X_n$, such that its value is i. The basic idea here is that for each clause we can arrange the obstacles such that if a path's encoding satisfies the clause, it will not be "stretched", i.e. its path length is the same as the length of the canonical path. If the encoding does not satisfy the clause, the path will be "stretched" so it is longer than the canonical path by a distinguishable amount, say γ. Therefore, the 3-SAT formula is satisfiable if and only if there exists a path whose length is the same as the canonical path length, which we know beforehand.

4.1 A shuffler and a filter

As described in [3], in order to filter all the paths whose X_i is set to 0 or 1, we need to shuffle the paths such that X_i becomes the most significant bit (i.e. the bit X_i becomes *exposed*). The way to achieve this is by a construction call a *shuffler*. A shuffler is similar to a merger with some alterations. It lets the first 2^{N-1} paths go through the upper half of the merger and blocks the second 2^{N-1} paths from going through the upper half. It lets the second 2^{N-1} paths go through the lower half of the merger and blocks the first 2^{N-1} paths from going through the lower half. We can vary the parameters for the gadgets in the upper and lower halves of the merger such that at the location of merge, the order of the paths (ordered by their orientations) is $0, 2^{N-1}, 1, 2^{N-1} + 1, \ldots$. In this way, the shuffler has made the second bit to be the most significant bit. We can let the paths go through a sequence of shufflers to expose any bit.

If, instead of varying the parameters of the basic gadgets in the merger, we put a tiny obstacle in either the upper or the lower half of the merger, we get a construction called the *filter*. The effect of the tiny obstacle is to stretch the paths so that they are longer than the canonical path by a distinguishable amount. If we put the tiny obstacle in the upper half of the merger, the filter filters out paths whose most significant bit is 1, i.e. paths whose most significant bit is 1 will have lengths the same as the canonical path length, while the paths whose most significant bit is 0 (going through the upper half of the merger) will be stretched by a distinguishable amount.

4.2 The reduction

A naive way to "emulate" a clause, say $X_i \overline{X_j} X_k$, is as follows. First we let the paths to go through a sequence of shufflers so that the bit X_i becomes the most significant bit. We let the paths then go through a filter that stretches all the paths whose X_i is set to 0 by an amount of γ. Second we shuffle the paths again so that

X_j becomes the most significant bit. We then let the paths go through a filter that stretches all the paths whose X_j is set to 1 by an amount of γ. Finally we do the same for bit X_k as we did for bit X_i. Therefore, the encoding of a path satisfies clause $X_i\overline{X_j}X_k$ if and only if its length is stretched by no more than 2γ.

The above method does not work for the following reason. Take an example of path A and path B and a 3-SAT formula with two clauses. Path A is stretched by 2γ for the first clause and then by 2γ for the second clause, where as path B is stretch by 3γ for the first clause and then γ for the second clause. This means path A satisfies the formula while path B does not. However we can not tell the difference in path length of A and B. This is where the extra variables Y_i, Z_i, for $i = 1, \ldots, m$, are used. For the ith clause, the bits Y_iZ_i specify which of the 3 clause literals the encoding will satisfy. Let $y_1z_1 \ldots y_mz_mx_1 \ldots x_n$ be the encoding of a path (i.e. y_i, z_i and x_i are the boolean assignments for variables Y_i, Z_i and X_i respectively). The encoding indicates that the $(2y_i+z_i+1)$th literal of the ith clause should be satisfied by $x_1 \ldots x_n$.

Lemma 2 *For a 3-SAT formula, there exists a satisfying encoding, if and only if there exists an encoding $y_1z_1 \ldots y_mz_mx_1 \ldots x_n$, such that $1 \leq 2y_i + z_i + 1 \leq 3$ and $x_1 \ldots x_n$ satisfies the $(2y_i + z_i + 1)$th literal of the ith clause, for $i = 1, \ldots, m$.*

We shuffle the 2^N paths so that Y_iZ_i become the two most significant bits. Using these two bits, we divide the paths into 4 groups. A group is labeled j, if $2Y_i + Z_i + 1 = j$. Let X_{ij} be the variable for the jth literal of the ith clause. For the paths in group j, for $j = 1, 2, 3$, we send them into a subconstruction that first exposes the bit X_{ij} (i.e. makes X_{ij} the most significant bit) and then filters the most significant bit to be either 0 or 1 as appropriate. In the meantime, we block all the paths in group 4. Finally we reverse this construction so that all the paths come to a common locations with different orientations. Let $y_1z_1 \ldots y_mz_mx_1 \ldots x_n$ be the encoding of a path. At this moment, this path is not stretched by an extra of γ, if and only if $1 \leq 2y_i + z_i + 1 \leq 3$ and $x_1 \ldots x_n$ satisfies the $(2y_i + z_i + 1)$th literal of the ith clause.

5 Computing the Initial Orientation

We have shown that we can arrange the obstacles so that there exist 2^N shortest paths from an initial position. The initial location is easily found. To complete the construction, we need to show how to find the initial orientation.

Recall that a basic gadget is parameterized by length ℓ, which is the distance between point s and point m. $\bar{\theta}$ (as defined in Section 2.1) depends on ℓ. Let $\ell = x$ and let $\theta = \bar{\theta}$. Then,

Lemma 3 *θ and x satisfy*

$$\begin{aligned} x^2 \sin^2\theta + \quad & 2x\left(\sin\tfrac{\pi}{8} - \sin\theta + \cos\theta \sin\left(\tfrac{\pi}{8} + \theta\right)\right) \\ & + \left(1 - \cos\left(\tfrac{\pi}{8} + \theta\right)\right)^2 = 0. \end{aligned} \tag{7}$$

Proof: Because of the symmetry of the shortest path from s to t (see Figure 1), $\bar{\rho} = 2\rho_1$, where ρ_1 is the length of the path $P(\theta)$ between point s and point m.

¿From s to t, the total change in orientation is $2\theta + \pi/4$. Since the path is symmetric, the change in orientation at point m is $\theta + \pi/8$. This also lets us know the orientation of the path at point m. We call the ray starting at m and with this orientation the *middle ray.*

Let r_1 be the center of the unit circle tangent to the input ray and the input location, and let r_2 be the center of the unit circle tangent to the the middle ray at location m. Let L be the distance between r_1 and r_2. It is easy to see that $\rho_1 = \theta + \pi/8 + L$.

Without loss of generality, we can place the basic gadget in the coordinate system such that the coordinates of point s is $(0, 0)$, and the coordinates of point m is $(x, 0)$ (i.e. the line segment sm lies on the x-axis). It is easy to see that the coordinates of r_1 are

$$\left(\cos\left(\frac{3\pi}{4} - \theta\right), \sin\left(\frac{3\pi}{4} - \theta\right)\right),$$

and the coordinates of r_2 are

$$\left(\frac{\sqrt{2}}{2}x + \cos\frac{7\pi}{8}, \frac{\sqrt{2}}{2}x + \sin\frac{7\pi}{8}\right).$$

We can compute the square of distance L as

$$L^2 = x^2 - 2x\left(\sin\frac{\pi}{8} + \sin\theta\right) + \left(2 - 2\cos\left(\frac{\pi}{8} + \theta\right)\right).$$

Recall that θ is the angle that gives the shortest path from point s to point t among all input angles. Therefore,

$$\frac{d}{d\theta}\rho_1 = 0.$$

Plugging in the expression of ρ_1 in x and θ, eventually we obtain that θ and x will satisfy Equation (7). □

Now for a given θ, we can calculate x, such that $\theta = \bar{\theta}$ for this size x, by solving (7), which involves solving a quadratic equation.

6 Error Analysis

What we have to show is that the 2^N paths have the same length as the canonical path up to the polynomial $B = N^c$ number of bits of precision, i.e. the accumulated error in path length over all the gadgets is no more than 2^{-B}. Each path has to pass through $\alpha(N)$ gadgets, where α is a low degree polynomial in N. Therefore it is sufficient to show that for each gadget, the difference in path length is no more than $2^{-B}/\alpha(N)$, or better, no more than $2^{-4B/3}$.

Let θ be an input angle and we require that $|\theta - \bar{\theta}| < 2^{-2B/3}$. By the discussion in Section 2.3, we know that $\rho(\theta)$ can be approximated as a polynomial in $\theta - \bar{\theta}$ and its first term coefficient is 0. This implies that $|\rho(\theta) - \bar{\rho}| = O((\theta - \bar{\theta})^2) = O((2^{-2B/3})^2) = O(2^{-4B/3})$. On the other hand, in the interval $[2^{-2B/3}, 2^{-B}]$, we have enough different angles to make 2^N paths, and this suffices to show the NP-hardness of the 2D curvature-constrained shortest-path problem.

Acknowledgments

This research was supported by grants NSF CCR-9725021 and CCR-96-33567 and Army Research Office contract DAAH-04-96-1-0448.

References

[1] P.K. Agarwal, P. Raghavan, and H. Tamaki. Motion planning for a steering-constrained robot through moderate obstacles. In *Proc. 27th Symp. on Theory of Computing*, pages 343–352, 1995.

[2] T. Asano, D. Kirkpatrick and C. Yap. $D1$ optimal motion of a rod, In *Proc. Symp. on Computational Geometry*, 1997.

[3] J. Canny and J. Reif. New lower bound techniques for robot motion planning problems. In *Proc. 28th Symp. on Foundations of Computer Science*, pages 49–60, 1987.

[4] L.E. Dubins. On curves of minimal length with a constraint on average curvature and with prescribed initial and terminal positions and tangents. *Amer. J. Math.*, 79:497–516, 1957.

[5] S. Fortune and G. Wilfong. Planning constrained motion. *Annals of Mathematics and Artificial Intelligence*, 3:21–82, 1991.

[6] P. Jacobs and J. Canny. Planning smooth paths for mobile robots. In Z. Li and J. Canny, editors, *Nonholonomic Motion Planning*, pages 271–342. Kluwer Academic Publishers, 1992.

[7] J. Reif and H. Wang. Non-uniform discretization approximations for Kinodynamic motion planning and its applications. In *Proc. 2nd Workshop on Algorithmic Foundations of Robotics*, 1996.

[8] H. Wang and P. K. Agarwal. Approximation algorithms for curvature-constrained shortest paths. In *Proceedings of the 7th ACM-SIAM Symposium on Discrete Algorithms*, pages 409–418, 1996.

[9] H.J. Sussmann. Shortest 3-dimensional paths with a prescribed curvature bound. In *Proc. 34th Conf. on Decision and Control*, pages 3306–3312, 1995.

Taylor & Francis
Taylor & Francis Group
http://taylorandfrancis.com

Approximation and Decision Algorithms for Curvature-Constrained Path Planning: A State-Space Approach

Jürgen Sellen, *ESG Elektroniksystem- und Logistik-GmbH, Munich, Germany*

We consider the problem of planning curvature-constrained paths amidst polygonal obstacles, connecting given start and target configurations. This problem has been extensively studied in the automatic control and in the computational geometry communities, and has applications e.g. in flight path planning and in motion planning for car-like robots.
In this paper, we present an approximation algorithm which is able to find an admissible path with curvature R^{-1} which is not longer than $(1+\varepsilon)$ times the length of the shortest admissible path with curvature $(R+\varepsilon)^{-1}$ and clearance ε from the obstacles. Assuming that the planning space is the unit cube $[0,1]^2$, the algorithm has running time $O(\varepsilon^{-3})$. Unlike previous approaches [8, 18], the presented algorithm discretizes state-space and can also be used for motion planning in weighted regions.
Based on this algorithm, we describe a procedure to determine if there exists an admissible path that satisfies a given curvature-constraint. This algorithm has the ability to solve intuitively easy problem instances quickly, i.e., in polynomial time, but does not terminate for one critical value of the curvature-constraint. In contrast, the best known complete decision scheme [19] is exponential.

1 Introduction

1.1 Motivation

One of the most basic problems in non-holonomic motion planning is the problem of computing smooth motions of a point-sized mobile robot with a given, minimal turning radius.

This problem is often considered as a first step towards modelling the actual kinematics of a car. In this case, the curvature-constraint is imposed by the steering mechanism. The midpoint between the two rear wheels can be shown to move along a curvature-constrained path. The environment is generally modelled by polygonal obstacles.

Another application arises in mission planning for cruise missiles. Here, bounded curvature can provide a reasonable idealization for the flight dynamics. The optimization task is to find a path which maximizes the survival probability of the missile. The probabilities are imposed by threats like radar, and can vary continuously over the terrain. Hence, the environment is a digital map with probability values. The situation can thus be compared to 'weighted region problems' instead of problems with 'hard obstacles'.

We note that, although our results are formulated for the case of polygonal obstacles, the presented approximation algorithm works for the latter planning problem as well.

1.2 Problem statement

In curvature-constrained path planning, configurations correspond to tuples specifying position and orientation of the moving object. The orientation can be represented by a unit vector or by an angle.

The basic *path planning problem* can be formulated as follows: given initial and final configurations S, T, a polygonal environment $E \subseteq \mathbf{R}^2$, and a curvature-bound $R^{-1} > 0$, find a C^1-path $w : [0,l] \to \mathbf{R}^2$ satisfying

(1) $w(t) \in E \quad \forall\, t \in [0,l]$,

(2) $||\dot{w}(t)|| = 1$,
$||\dot{w}(t_2) - \dot{w}(t_1)|| \leq R^{-1}|t_2 - t_1| \quad \forall\, t, t_1, t_2 \in [0,l]$,

(3) $(w(0), \dot{w}(0)) = S, (w(l), \dot{w}(l)) = T$.

We assume that the environment E is a subset of the unit square $[0,1]^2$, and that $[0,1]^2 - E$ consists of convex polygonal obstacles with constant complexity. Constant complexity means that we shall omit running time dependencies on the number of obstacle edges. This number only comes into play for checking whether grid points or grid edges lie in free space. The convexity assumption is used for the decision algorithm (lemma 10) but plays no central role. In addition to convexity, we assume that the obstacles have a pairwise distance greater than Δ, $\Delta < R$, and that the radius R is smaller than 1. Also, we assume that S and T have distance $\geq \Delta$ from the obstacles. Our focus is to obtain asymptotic results for small ε.

Condition (2) restricts the set of possible paths (resp., motions) to those with unit speed and bounded curvature. The parameter t denotes time, but also equals path length in this setting. The parameter R is called the *minimal turning radius* of the robot, and R^{-1} the *maximal curvature* of the unit speed path w. We denote a path w that satisfies conditions (1)-(3) as *R-constrained.*

The *shortest path problem* in this context is to find an R-constrained path with minimal Euclidean length l. The *decision problem* is to determine whether an R-constrained path exists. The intrinsic complexity of the decision task can be illustrated by examining the *critical radius*:

Definition 1 For given E,S and T, the critical radius R_c is the supremum of all radii $R \in [0, \infty]$ for which an R-constrained path w exists. The value R_c^{-1} is called critical curvature of (E, S, T).

While, for any fixed query radius R, the shortest R-constrained path has finite length, this length may increase to infinity for a series of query radii that approaches R_c. In particular, there may be no finite-length path that satisfies the constraint R_c. An example for this situation can be found in [14].

1.3 Previous work

The mathematical foundation for the algorithmic treatment of curvature-constrained path planning has been laid in the 60's, by a paper of Dubins [6]. This paper characterizes the shortest R-constrained trajectories between given start and target configurations in the unrestricted plane. The main result consists in the reduction of the set of possible solutions to a discrete set of *canonical trajectories*, trajectories which consist of arcs of radius R and straight line segments. This characterization has been extended by Reeds and Shepp [11] to motions that include reversals ('backing up'). Some recent work [3, 17, 5, 16] deals with control-theoretic aspects of the Dubins and the Reeds/Shepp problem, to generate optimal trajectories in an obstacle-free setting.

The first exact decision algorithm under the existence of obstacles is due to Fortune and Wilfong [7], and is based on a reduction of the problem to deciding the validity of some Tarski formula. The running time of this algorithm is exponential in the number of the vertices of the obstacles and in the bitsize of their coordinates.

Jacobs and Canny [8] describe an approximation algorithm to compute shortest R-constrained paths. Their algorithm finds an approximate shortest path if this path is sufficiently *robust* to allow for small changes in orientation (resp., position) along the path without causing new collisions. The algorithm uses a discretization of the space of contact configurations between the moving object and the obstacles. The worst case running time is $O(\frac{n^3}{\varepsilon} \log n + (\frac{n}{\varepsilon})^2 \log(\frac{n}{\varepsilon}))$, where ε is a measure for both the closeness of the approximation and the robustness of the path, and n denotes the number of obstacle edges. Recently, Wang and Agarwal [18] improved the algorithm of Jacobs and Canny, achieving a running time of $O((\frac{n}{\varepsilon})^2 \log(\varepsilon))$. Also, their running time does not depend on the edge-lengths of the obstacles, a dependency which is inherent in the original approach.

In [12], Reif and Wang describe an approximation algorithm for kinodynamic motion planning which is based on a non-uniform discretization of state-space.

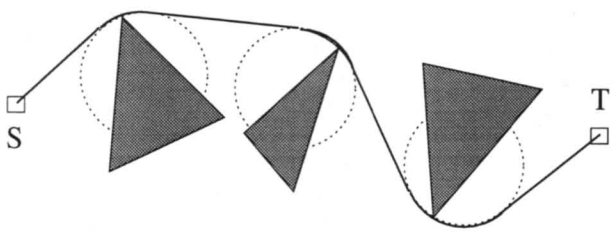


Figure 1: *Canonical (Dubins) path.*

They also apply their technique on shortest curvature-constrained path planning in any dimension d. However, the worst case running time of this algorithm has a very high dependency on the desired approximation error ε.

In recent years, there have been several results on simpler subclasses of the general setting [19, 1, 4], including approximation and decision algorithms for the case in which the environment is given by moderate obstacles (i.e., convex obstacles whose boundary is itself curvature-constrained).

The first lower bound for curvature-constrained path planning in 2d is given by Reif and Wang [13]. By using techniques similar to the NP-hardness proof of the 3-dimensional Euclidean shortest path problem, they show that shortest curvature-constrained path planning is NP-hard.

However, it is still open whether the decision problem is NP-hard. Even the known approximation algorithms for the shortest path problem use settings in which the approximation error ε is attached not only to the length of the path but also to other qualities of the problem (e.g. position, orientation, the curvature-constraint or robustness as introduced by Jacobs and Canny).

The notion of critical curvature as well as an algorithm to approximate R_c have been introduced by the author in [14]. The results of this paper are a follow-up and improvement of [14, 15].

1.4 Outline and results

In this paper, we present new algorithms for the approximation problem and the decision problem.

The approximation algorithm is based on a setting of the shortest path problem in which we relate the approximation error to an uncertainty in the turning radius and in the location of the moving object:

Definition 2 A path w is called an ε-approximate solution of the shortest curvature-constrained path problem if w is R-constrained with length $\leq l(1+\varepsilon)$, where l is the length of the shortest $(R+\varepsilon)$-constrained path from S to T in E with ε-clearance from the obstacles.

Clearance ε means that the path keeps distance ε to the obstacles (we also denote the path as ε-free). Based on this setting, we shall prove

Theorem 3 *An ε-approximate solution of the shortest curvature-constrained path problem can be computed in time $O(\varepsilon^{-3})$.*

The algorithm is based on a discretization of the 3-dimensional set of possible configurations of the robot. In particular, we use a uniform grid of spacing ε for the robot's location, and a special grid of spacing approximately $\sqrt{\varepsilon}$ for its orientation. Hence, we obtain a state set of size $O(\varepsilon^{-5/2})$. The possible transitions through the state set are given by motions along Dubins canonical trajectories: they correspond to motions along circles of radius R and straight line connections between such motions. For each state, we need $O(\varepsilon^{-1/2})$ transitions. Finally, the desired approximate shortest path can be computed from the transition graph by a simple shortest path algorithm. The correctness of the algorithm can be proven by several deformation ('taut string') arguments.

Our approach to the decision problem exploits a notion of intuitive complexity. Similar approaches have been successfully applied to traditional motion planning. E.g., Alt et. al. [2] show that the problem to find a collision-free motion of a rectangular box in a planar polygonal environment is easy if the environment admits a solution for a bigger box. In analogy to this paper, we denote our measure for the intuitive complexity as *width*:

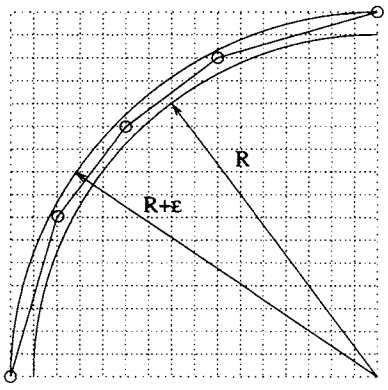


Figure 2: *Grid circle.*

Definition 4 Let R_c be the critical radius of (E, S, T). Then the width of the decision problem with query radius R is $W = |R - R_c|$.

With this notation, we shall prove

Theorem 5 *The existence of an R-constrained path can be decided in time* $O(W^{-3}(\frac{\Delta}{R})^{-6})$.

Hence, for $W \neq 0$, our approach provides a decision procedure which is more practical than the inherently exponential solution of Fortune and Wilfong [7]. The decision procedure is polynomial in W^{-1}, and gets exponential only for exponentially small values of W.

The decision procedure exploits the approximation algorithm which is executed for a series of different values of R and ε. Additionally, we use a central lemma which shows that clearance can be traded versus curvature: if there exists an admissible $(R + \Omega(\varepsilon(\frac{\Delta}{R})^{-2}))$-constrained path, then there also exists an $(R + \varepsilon)$-constrained path with clearance ε.

2 The Approximation Scheme

2.1 Preliminaries

In this subsection we review some normal form properties of curvature-constrained paths. These properties play a central role in our state-space approach to approximation.

A *canonical trajectory* with respect to the curvature-constraint R^{-1} is a path which first follows an arc of radius R, then either a straight line or another arc of radius R, and finally again an arc of radius R. This includes degenerate cases, in which the *initial arc*, the *line segment*, or the *final arc* are missing.

For a given configuration (p, u), there are two circles of radius R passing through p and tangential to u, corresponding to a left or to a right hand turn. Given start and target configurations S and T, there are thus at most 6 different canonical trajectories from S to T, corresponding to the *turn types* $\{LSL, LSR, RSL, RSR, LRL, RLR\}$.

Dubins [6] shows that the shortest path between S and T is a canonical trajectory provided $E = \mathbf{R}^2$. Jacobs and Canny [8] extend this result to polygonal environments (see figure 1):

Lemma 6 *Among all R-constrained paths w from S to T in E, there exists a path w of minimal length which touches* ∂E *only finitely many times and which is canonical between two contact configurations.*

More precisely, the path w touches ∂E during finitely many time intervals $[t_1, t_2]$. The contact configurations may be defined as those configurations $(w(t_i), \dot{w}(t_i))$ at which w enters or leaves ∂E.

2.2 Discretization

The simplest approach to path planning is to discretize the set of possible configurations (or states) and to search for a feasible path inside an adjacency graph of the configurations. What makes this approach interesting in our case is the non-holonomic constraint together with the desire to obtain the smallest possible number of states and transitions.

Our approximation setting allows to attach the approximation error ε to both qualities of state: location and (indirectly) orientation. To achieve the desired error limit in terms of location, we discretize E by a uniform grid of spacing $\Theta(\varepsilon)$. To discretize the set of orientations, we shall exploit normal form properties of curvature-constrained paths. Our set of states and the corresponding transitions will be constructed in a way that allows to follow a given Dubins path within a 'tube' of width $O(\varepsilon)$ by a feasible grid path. Here,

we make use of the observation that circles of radius 1 can be approximated with error ε by $O(\varepsilon^{-1/2})$ grid segments of length approximately $\sqrt{\varepsilon}$.

Lemma 7 *There exists a planar grid G with uniform grid spacing $\Theta(\varepsilon)$ and a closed polygonal path (i.e., a grid circle) $C = (s_1, \ldots, s_k)$ on G such that the following properties hold:*

(1) C consists of $O(\varepsilon^{-1/2})$ segments $s_1, \ldots, s_k$.

(2) C is contained inside the annulus between the cocentric circles with radii $R + \frac{2}{3}\varepsilon$ and $R + \varepsilon$.

(3) The polygon with boundary C is convex and point-symmetric.

In the following, we shall identify each segment $s_i = \overline{pq}$ of C by the vector $q - p$. The set $D = \{s_1, \ldots, s_k\}$ of these direction vectors will be used to discretize orientation.

We note that, by the symmetry of C, D contains for each vector v also its opposite vector $-v$, and that D is thus independent of the orientation of C. Also, as an addendum to lemma 7, the grid spacing $\Theta(\varepsilon)$ can be chosen such that the angle between any two neighboring direction vectors s_i, s_{i+1} is bounded above by $c2\pi\sqrt{\varepsilon}$ for any given constant $c < 1$.

Now, let $E^\varepsilon = \{\ p \in [0,1]^2\ ;\ d(p, [0,1]^2 - E) \geq \frac{\varepsilon}{3}\ \}$ be the environment shrunk by $\frac{\varepsilon}{3}$, and let G_E be the set of all grid points of G inside E^ε.

With these definitions, we define a state set by

$$\mathcal{S} = G_E \times D \times \{L, R\}.$$

The set $\{L, R\}$ serves to specify whether the robot follows a left-hand or a right-hand turn. We shall use the notation $\overline{L} = R$ and $\overline{R} = L$.

For each vector s, we define the right-turn successor $succ_R(s)$ as the vector that follows after s during a clockwise traversal of the grid circle, and analogously the left-turn successor $succ_L(s)$ as the successor during a counter-clockwise traversal.

According to Dubins' result, the set of transitions is chosen to allow motions along circular arcs of radius R and straight line 'jumps' between such motions.

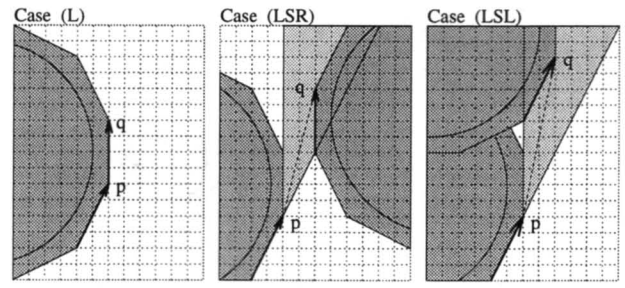


Figure 3: *Transitions from a state (p, s, L).*

For $F \in \{L, R\}$, let

$$\begin{aligned} \mathcal{C}(p, s, F) \quad = \quad & \{\ q \in G\ ;\ \exists\ a, b \in [0,1] \\ & \text{s.t. } q = a \cdot s + b \cdot succ_F(s)\ \} \end{aligned}$$

be the grid parallelogram spanned by s and $succ_F(s)$. We note that this parallelogram can be viewed as a 'vector basis' for the infinite cone spanned by s and $succ_F(s)$: any point q inside this cone can be reached by a vector sum $q = p + \sum_i (p_i - p)$, with $p_i \in \mathcal{C}(p, s, F)$. Now, let

$$\mathcal{C}_E(p, s, F) = \{\ q \in \mathcal{C}(p, s, F)\ ;\ \overline{pq} \in E^\varepsilon\ \}$$

be the part of $\mathcal{C}(p, s, F)$ that is visible from p.

Then we define $\mathcal{T}(p, s, F)$ as the following set of transitions (see figure 3):

$$\begin{array}{ll} (F) & (p, s, F) \rightarrow (p + succ_F(s), succ_F(s), F) \\ (FS\overline{F}) & (p, s, F) \rightarrow (q, succ_F(s), \overline{F})\ \forall q \in \mathcal{C}_E(p, s, F) \\ (FSF) & (p, s, F) \rightarrow (q, s, F)\ \forall q \in \mathcal{C}_E(p, s, F). \end{array}$$

Finally, we define a set of transitions $\mathcal{T}$ as the union of all transitions $\mathcal{T}(p, s, F)$ from states $(p, s, F) \in \mathcal{S}$.

2.3 Correctness

Let $\mathcal{G} = (\mathcal{S}, \mathcal{T})$ be the directed graph that is given by the states $\mathcal{S}$ and the transitions $\mathcal{T}$. For simplicity, let us assume that the start and target points $S, T \in E^\varepsilon$ coincide with graph nodes.

In this subsection, we shall prove that a shortest path search from S to T in $\mathcal{G}$ provides an ε-approximate solution of the curvature-constrained shortest path problem.

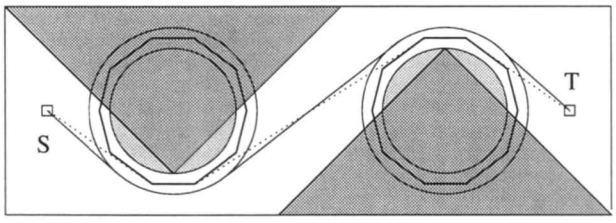


Figure 4: *Deformation of a Dubins path.*

Lemma 8 *If E contains an $(R+\varepsilon)$-constrained path w from S to T with ε-clearance and length l, then $\mathcal{G}$ contains a path w' from S to T with length $\leq l(1+\varepsilon)$.*

Proof: W.l.o.g. we may assume that w is a Dubins path. For each circular arc of this path, we introduce a new obstacle: a grid circle inside the circle that corresponds to the arc. Now, let w^* be the taut string path with respect to the new obstacles (see figure 4). Due to the ε-clearance of w, w^* is $\frac{2}{3}\varepsilon$-free. Further, w^* is a grid path that follows either edges of grid circles or that connects grid circles by straight line segments. The circular parts correspond to transitions of type L or R. Now consider a segment that connects two circular parts and that starts at a state (p, s, F). As this segment must be tangential to the grid circles, it lies inside the cone spanned by s and $succ_F(s)$, and corresponds to (successive application of) transitions of type FSF,$FS\overline{F}$. As w^* has $\frac{\varepsilon}{3}$-clearance from $[0,1]^2 - E^\varepsilon$, and by use of the addendum to lemma 7, there is a choice of transitions that lies inside E^ε and hence inside $\mathcal{G}$. Finally, the corresponding path w' has length $l(w') \leq (1+\varepsilon)l(w^*) \leq (1+\varepsilon)l(w)$. □

Lemma 9 *If $\mathcal{G}$ contains a path w from S to T with length l, then E contains an R-constrained path w' from S to T with length $\leq l$.*

Proof: Consider a segment u of w that corresponds to a transition of type L or R. The segment u is part of a grid circle to the left and of a grid circle to the right of u. Depending on the transition type, we introduce a new obstacle which is a circle of radius $R+\frac{2}{3}\varepsilon$ inside the grid circle to the left or to the right of u. The idea now is that the desired path w' can be chosen as the taut string path with respect to the new obstacles (see figure 4). However, special care again has to be taken about the transitions of type FSF, $FS\overline{F}$. Here, one can show that the corresponding part of the path moves inside a channel of successive parallelograms $\mathcal{C}(p, s, F)$. The concave corners of this channel can be made 'smooth' by circles of radius $R+\frac{2}{3}\varepsilon$, and the taut string path inside the channel gets R-constrained. The so constructed path w' keeps distance $\leq O(\varepsilon)$ from w. By the addendum to lemma 7, we can achieve a distance $\leq \frac{2}{3}\varepsilon$, and the path w' hence stays inside E. □

Lemma 9 is not trivial. It ensures that a path in $\mathcal{G}$ can be 'globally' deformed to a feasible path with radius $\geq R$. This is remarkable as this does not follow if we consider the path segments 'locally': it may be impossible to connect the midpoints of two adjacent path segments by a circular arc with radius $\geq R$.

The correctness of the approximation scheme directly follows from the two lemmata above.

2.4 Time analysis

The state set $\mathcal{S}$ has size $O(\varepsilon^{-5/2})$. Further, there are $O(\varepsilon^{-1/2})$ transitions from each state (the number of grid points inside a parallelogram $\mathcal{C}_E(p, s, F)$). The shortest path search can thus be performed in time $O(\varepsilon^{-3})$. Omitting the complexity of the construction of $\mathcal{G}$ (resp., the dependency on the number of obstacle edges), we obtain theorem 3.

If we assume that the obstacles are given by n edges, then the graph $\mathcal{G}$ can trivially be constructed in time $O(n/\varepsilon^3)$, by checking any transition for intersection with the obstacle edges. However, it is also possible to use one of the known output-sensitive algorithm for reporting line segment intersections. (In order to control the intersections between transitions, one may exploit that the transitions can be divided into $O(1/\varepsilon)$ sets with $O(1/\varepsilon^2)$ transitions of the same direction.)

3 The Decision Procedure

3.1 Trading curvature vs. clearance

As a central lemma in our approach to the decision problem, we show in this subsection that we can

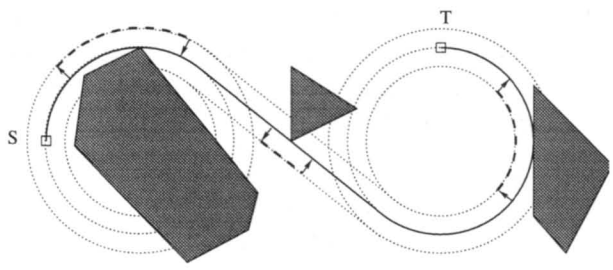


Figure 5: *Construction of ε-free $(1-\delta)$-constrained paths.*

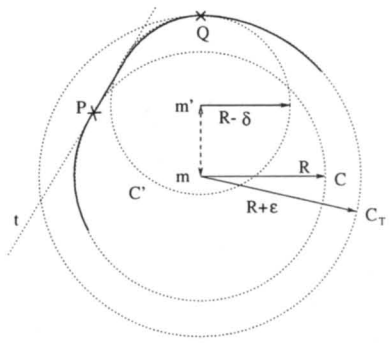


Figure 6: *Jump between circles of different radii.*

achieve clearance of the moving object by allowing it to travel along a path with higher curvature:

Lemma 10 *If there exists an R-constrained path w from S to T inside E then there exists an ε-free, $(R-\delta)$-constrained path w', with $\delta = c\varepsilon(\frac{\Delta}{R})^{-2}$ for a suitable constant c.*

Proof: Let us consider the tube T of all points with distance $\leq \varepsilon$ from $w([0,l])$. We shall show that we can bend w inside T such that its clearance from the obstacles gets ε.

First, we break w into parts which can be bended 'away' from the obstacles independently from each other, see figure 5. We say that w touches some obstacle O at time t if the distance between $w(t)$ and O is less than ε. If w touches an obstacle O_1 at time t_1, then w can touch a different obstacle O_2 at time t_2 only for $|t_1 - t_2| > \Delta - 2\varepsilon$. We say that an obstacle O lies to the left (resp., right) side of w at t, if the line perpendicular to $\dot{w}(t)$ and passing through $w(t)$ intersects the

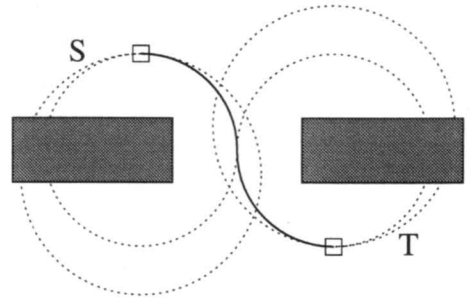


Figure 7: *An ε-free path without radius tolerance.*

obstacle O to the left (resp., right) of $w(t)$. We note that a path can touch the same obstacle at different sides for different t, however, this is only possible if the path makes a turn inbetween. It is now easy to see that $[0,l]$ can be broken into intervals, such that w touches at most one obstacle O inside each interval, such that this obstacle lies only at one side of w during this interval, and such that different intervals have distance $> \Delta - 2\varepsilon$.

Now, consider an interval $[t_1, t_2]$ during which w touches some obstacle O. Then we bend w by moving to the side of the tube T opposite to O. The necessary 'jump' from the path to one side of the tube (resp., back again) has to be completed inside $[t_1 - O(\Delta), t_1]$ (resp., $[t_2, t_2 + O(\Delta)]$). It is easy to see that the jumps can be realized by motions along 2 circles with radius $R - \delta$ for some $\delta > \varepsilon$.

The most critical jumps are those between 2 circular arcs with a radius that differs by ε. In the following, we discuss one of the possible cases, given by figure 6. The task is to jump from a point P on circle C with radius R to a point Q on circle C_T with radius $R+\varepsilon$. The jump is realized by a circle C' with radius $R-\delta$ and by a straight line segment t, the tangent of C and C' (it is possible to use a second circle with radius $R-\delta$ instead of the line segment t). If the circles C and C' have midpoints $m = (0,0)$ and $m' = (0, \varepsilon+\delta)$, then the intersection P of t with C has coordinates $P_x = \pm R\frac{\sqrt{\varepsilon(\varepsilon+2\delta)}}{\varepsilon+\delta}$ and $P_y = R\frac{\delta}{\varepsilon+\delta}$. Finally, in order to obtain a distance between P and $Q = (0,R)$ of at most $O(\Delta)$, it is sufficient to choose δ as in the lemma. □

It is interesting to note that the opposite direction of lemma 10 is not true. There exist R-constrained paths with clearance ε which can not be made $(R+\delta)$-constrained for any $\delta > 0$, see figure 7.

3.2 Width-dependent decision

Let $A(R,\varepsilon)$ denote our approximation algorithm, started for a fixed input instance (E,S,T), with grid spacing chosen to achieve approximation error $\varepsilon_G = \varepsilon c^{-1}(\frac{\Delta}{R})^2$ (c as in lemma 10) and curvature-constraint R^{-1}.

Then $A(R,\varepsilon)$ has the following properties:

- If the algorithm finds a path, then there exists an admissible R-constrained path (lemma 9).
- If the algorithm finds no path, then there exists no admissible $(R+\varepsilon)$-constrained path (lemma 8 and lemma 10).

The desired decision can be made if either $A(R,\varepsilon)$ finds an admissible path, or if $A(R-\varepsilon,\varepsilon)$ finds no admissible path. Now, we run $A(R,\varepsilon)$ and $A(R-\varepsilon,\varepsilon)$ in an infinite loop with decreasing values $\varepsilon_i = 2^{-i}$, until a decision can be made. This will happen as soon as $\varepsilon_i < |R-R_c|/2$, or, $2\varepsilon_i < W$. Hence, the number of necessary loops equals $i = O(log(W^{-1}))$. In each loop, the running time of the queries to the approximation algorithm is bounded by $O(\varepsilon_i^{-3}(\frac{\Delta}{R})^{-6})$. The total running time is dominated by the running time of the last step, which is $O(W^{-3}(\frac{\Delta}{R})^{-6})$. This finally proves theorem 5.

4 Discussion

In this paper, we developed algorithmic approaches to curvature-constrained path planning that rely on a clever discretization of state-space. In particular, we introduced a shortest path approximation algorithm that relates the approximation uncertainty to location (i.e., clearance) and curvature. By trading curvature vs. clearance, the approximation algorithm can be used as the central part of a simple decision procedure which exploits a notion of intuitive complexity.

We argue that the concepts behind our results have wider applicability, and that similar approaches can be used in various multicriteria and non-holonomic path planning problems.

Though the proposed approximation and decision schemes are easy to implement, the given analysis is complicated and requires big constants. With respect to practical applications, it would be desirable to derive better constants, resp., to improve the presented techniques.

References

[1] P.K. Agarwal, P. Raghavan, H. Tamaki, Motion planning for a steering-constrained robot through moderate obstacles, *Proc. 27th ACM Symp. Theory Comp.*, 1995, 343-352.

[2] H. Alt, R. Fleischer, M. Kaufmann, K. Mehlhorn, S. Näher, S. Schirra, C. Uhrig, Approximate Motion Planning and the Complexity of the Boundary of the Union of Simple Geometric Figures, *6th ACM Symp. Comp. Geom.*, 1990, 281-289.

[3] J. Boissonnat, A. Cérézo, J. Leblond, Shortest paths of bounded curvature in the plane, *Proc. IEEE Int. Conf. on Robotics and Automation*, 1992, 2315-2320.

[4] J. Boissonnat, S. Lazard, A polynomial-time algorithm for computing a shortest path of bounded curvature amidst moderate obstacles, *12th ACM Symp. Comp. Geom.*, 1996, 242-251.

[5] X.-N. Bui, P. Souères, J. Boissonnat, J. Laumond, The Shortest Path Synthesis for Nonholonomic Robots Moving Forwards, *IEEE Int. Conf. on Robotics and Automation*, 1994, 2-7.

[6] L.E. Dubins, 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*, 1957, 497-516.

[7] S. Fortune, G. Wilfong, Planning Constrained Motion, *Annals of Math. and AI 3*, 1991, 21-82.

[8] P. Jacobs, J. Canny, Planning Smooth Paths for Mobile Robots, *Proc. IEEE Int. Conf. on Robotics and Automation*, 1989, 2-7. Also in: *Nonholonomic Motion Planning*, ed. Z. Li, J. Canny, Kluwer Academic Press, 1993, 271-342.

[9] J.-C. Latombe, *Robot Motion Planning*, Kluwer Academic Publishers, 1991.

[10] C.H. Papadimitriou, An Algorithm for Shortest-Path Motion in Three Dimensions, *Inform. Proc. Letters*, Vol. 20, 1985, 259-263.

[11] J.A. Reeds, L.A. Shepp, Optimal paths for a car that goes both forwards and backwards, *Pacific Journal of Mathematics*, 145(2), 1990, 367-393.

[12] J. Reif, H. Wang, Non-Uniform Discretization Approximations for Kinodynamic Motion Planning and its Applications, *Proc. 2nd Int. Workshop on the Algorithmic Foundations of Robotics*, AK Peters, 1996.

[13] J. Reif, H. Wang, The Complexity of the Two Dimensional Curvature-Constrained Shortest-Path Problem, *these proceedings*.

[14] J. Sellen, Planning Paths of Minimal Curvature, *Proc. IEEE Int. Conf. on Robotics and Automation*, 1995.

[15] J. Sellen, Curvature-Constrained Path Planning, *Modelling and Planning for Sensor-Based Intelligent Robot Systems*, World Scientific, to be published.

[16] P. Soueres, J.-P. Laumond, Shortest Paths Synthesis for a Car-Like Robot, *IEEE Transactions on Automatic Control*, Vol.41 No.5, May 1996, 672-688.

[17] H.J. Sussmann, G. Tang, Shortest paths for the Reeds-Shepp car: a worked out example of the use of geometric techniques in nonlinear optimal control, *Report SYCON-91-10, Rutgers University*, 1991.

[18] H. Wang, P.K. Agarwal, Approximation Algorithms for Curvature-Constrained Shortest Paths, *Proc. 7th ACM-SIAM Symp. Discrete Algorithms*, 1996.

[19] G. Wilfong, Motion planning for an autonomous vehicle, *Proc. IEEE Int. Conf. on Robotics and Automation*, 1988, 529-533.

Taylor & Francis
Taylor & Francis Group
http://taylorandfrancis.com

Elastic Bands for Nonholonomic Car-Like Robots: Algorithms and Combinatorial Issues

Hazem Jaouni, *LAAS-CNRS, Toulouse, France*
Maher Khatib, *LAAS-CNRS, Toulouse, France*
Jean-Paul Laumond, *LAAS-CNRS, Toulouse, France*

This paper presents a dynamic structure allowing to enrich any nonholonomic motion planner for car-like robots with the capacity of reactiveness to environment changes. It is based on a pavement of paths by star shaped domains computed in the singular metric of the car-like robot. The algorithms to build such pavements are presented and their complexity is analyzed. We then introduce an optimization routine consisting in following the negative gradient of an energy function, the energy being the result of a combination of repulsive forces (due to obstacles) and attractive forces (tending to shorten the path). The key point of the approach is to consider the configuration space as a metric space equipped with the nonholonomic metric.

Applications to nonholonomic path planning including reactivity to moving obstacles are presented.

1 Elastic Bands and Nonholonomic Metrics

About five years ago, Quinlan and Khatib [9] proposed a dynamic trajectory modification concept called the elastic band allowing to maintain a permanent flexible and deformable path between an initial and a final robot configuration. The key point of the concept lies on a pavement of the path with collision-free bubbles, whose size and shape present a simplified and practical image of the free space around the path. The sequence of bubbles that cover the path is called a bubble band. Applying contraction forces between bubbles and repulsion forces due to the obstacles transforms the bubble band into a elastic band. Among the possible applications of this concept let us mention the reactive control (allowing to account for environment changes during execution phase) and path optimization.

Elastic bands are wholly built on distance functions. They were first developed for holonomic robots where the metric space associated to shortest paths is simply the Euclidean space. In order to extend the bubble band to a nonholonomic robot, the problem of shortest paths should be resolved and the associated distances, between configurations and between a configuration and an obstacle, should be found.

The problem is solved for pointwise car-like robots. Reeds and Shepp [10] established a sufficient family that contains the shortest paths (from a configuration to a configuration) for such a robot. This family was refound and refined using control theory simultaneously by Sussmann and Tang [13] and by Boissonnat, Cerezo and Leblond [3]. The metric space associated to this set of shortest paths was established by Soures and Laumond [12] through the partition of the configuration space into classes, each containing a single type of paths. Finally, the distances to points and segments in the plane were resolved, respectively, by Soures, Fourquet and Laumond [11] and by Vendittelli and Laumond [14].

Consequently, a bubble band for the car-like robot has been developed in [5] where the bubbles are balls in the corresponding metric space. This lead to a more general definition of the bubble band concept [6].

The interest of the bubble band (and the associated elastic band) lies in a simple combinatorial way to represent a given path among the obstacles surrounding it: the optimization routines are all the more efficient as the length of the sequence of bubbles is short.

The contribution of this paper is to replace the balls by "stars" and to show the added value (in terms of combinatorial complexity) of this new approach. A

configuration being given, a star is the set of configurations which are reachable by a shortest path that does not intersect the obstacles. After introducing necessary definitions, we present an algorithm to compute the pavement of a path with stars and we compare its complexity with respect to the pavement with balls. It appears that, in the worst case of the parking task, both pavements have the same (optimal) complexity, while in the average cases, the pavement with stars is much less expensive than the pavement with balls. Finally, we show how to transform such a pavement with stars into an elastic band and we present experimental results dealing with nonholonomic path planning including reactivity to moving obstacles.

2 Bubble Bands: Definitions

The basic idea of the bubble band method is the pavement of an arbitrary path by a finite sequence of subsets in the configuration space. In order to guarantee the feasibility of the path, these subsets should verify several properties [8, 6]. The following definitions sum up these properties. The configuration space of the robot is a manifold $\mathcal{M}$. $\mathcal{O} \subset \mathcal{M}$ represents the union of obstacles.

Definition 1 Let $\mathcal{O}(\mathcal{M})$ be a family of open subsets in $\mathcal{M}$ such that for all open subset $\mathcal{A}$ of $\mathcal{M}$, for all configuration P in $\mathcal{A}$, there is an element $\mathcal{B}$ in $\mathcal{O}(\mathcal{M})$ that verifies:

- $\mathcal{B} \subset \mathcal{A}$ (*non-collision condition*),
- $P \in \mathcal{B}$ (*cover condition*),
- $\forall\, Q \in \mathcal{B}\ \exists$ an optimal path $\gamma(P, Q) \subset \mathcal{B}$ joining P to Q (*accessibility condition*).

$\mathcal{O}(\mathcal{M})$ is called a set of *bubbles*. P is said to be a center of the bubble $\mathcal{B}$.

Though we do not require a bubble to have a unique center, the examples used in this paper (*i.e.* balls and stars) do have unique centers.

Definition 2 A *bubble band* joining an initial configuration $P_0 \in \mathcal{M}$ to a final configuration $P_1 \in \mathcal{M}$ is a finite sequence of bubbles $\{\mathcal{B}_i, i = 1, \ldots, n\}$ such that:

- All the bubbles $\mathcal{B}_i$ belong to the *free space*: $\mathcal{M} - \mathcal{O}$.
- $\mathcal{B}_i \cap \mathcal{B}_{i+1} \neq \emptyset\ \forall i = 1, \ldots, n-1$.
- P_0 is a center of $\mathcal{B}_1$ and P_1 is a center of $\mathcal{B}_n$.

Since we are interested in car-like robots, we will restrict our discussion to the manifold $\mathbf{R}^2 \times \mathcal{S}^\infty$ which represents the configuration space of such robots. This manifold has a metric structure compatible with the shortest paths feasible by these robots [12]. This metric is denoted by d_{RS}; it has been proved to be equivalent to the Euclidean one. The balls issued from it (see Figure 1) form a set of bubbles that verifies the conditions of Definition 2. In fact we are only interested in maximal balls relative to the inclusion operation. Let us then recall the definition of such a ball.

Definition 3 Let C be a configuration of the free space and let d_C be the length of the shortest feasible path to obstacles. The *ball* centered at C is the set of configurations that can be reached from C by optimal paths shorter than d_C.

Such a bubble is then defined by a center C and a radius d_C corresponding to the length of the shortest path leading the car-like robot to a contact with an obstacle; they are denoted by $\mathcal{B}(C, d_C)$ or $\mathcal{B}(C)$. Distance computation in this framework is addressed in [14]. A bubble band was developed that uses these bubbles for pointwise and circular car-like robots [5].

In this paper we present another set of bubbles. These bubbles are also define by a center (which is also unique). Each bubble is defined to be the set of configurations reachable from this center by shortest paths. This set incorporates thus more information about obstacles than the previous one, which will lead to a better representation of the free space and thus to a more efficient bubble band.

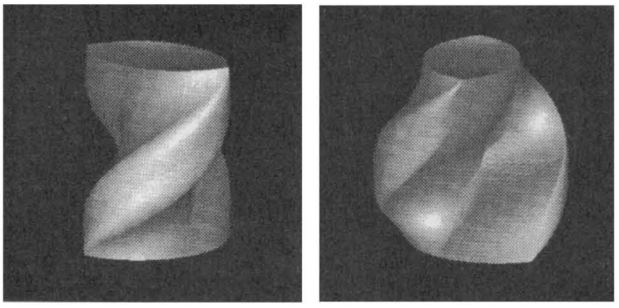


Figure 1: *Balls in the car-like robot metric; the outer surfaces represent the set of points in $\mathbf{R}^2 \times \mathcal{S}^\infty$ reachable by shortest paths of a given length (the θ-axis is almost vertical).*

Definition 4 Let C be a configuration of the free space. The *star* centered at C is the set of configurations that can be reached from C by shortest paths that does not intersect $\mathcal{O}$. It is denoted by $\mathcal{E}(C)$.

Remark: A star of center C contains always the ball with same center. Thus if we have a set of balls that cover a path, and we replace each ball by the star with same center, the resulting set of stars will also cover the path.

Figure 2 from [4] shows an example of the boundary of a star: in this example the obstacle is reduced to a straight line segment. This segment appears as a rectangular domain in the configuration space. The center of the star is the configuration $(0,0,0)$. The star is an unbounded domain whose (infinite) borders are truncated at some given distance from the origin for display purpose. The color of the various patches corresponds to the type of the shortest paths reaching the boundary.

The stars clearly verify the hypothesis of Definition 2: they are bubbles. Moreover, any collision-free shortest path starting at some some configuration C ends at a configuration belonging to $\mathcal{E}(C)$. Therefore, for any C, the star $\mathcal{E}(C)$ contains the ball $\mathcal{B}(C, d_C)$. As a consequence, the minimal number of stars necessary to cover an arbitrary path in a arbitrary environment is always lesser than the minimal number of necessary balls.

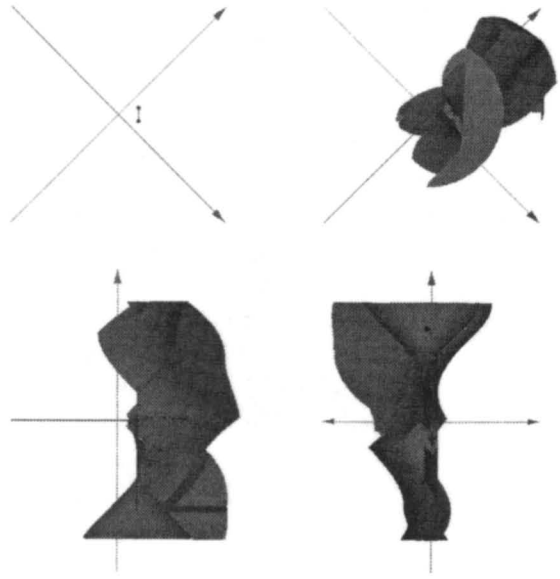


Figure 2: *Boundary of a star for a segment located near the origin.*

In fact we will show later that the maximal complexity of both bubble bands which corresponds to the number of bubbles that compose them and thus to the number of primitive operations necessary to join both ends is the same. Nevertheless, the mean complexity is lesser in the case of star bubbles because of the above remark.

Throughout this paper, *ball band* designs the bubble band where the bubbles are balls of the metric space and *star band* the bubble band that uses the stars instead.

3 Pavements: Algorithms and Complexity

The starting point of the bubble band is an arbitrary collision-free path that joins a starting configuration to a goal configuration in $\mathbf{R}^2 \times \mathcal{S}^\infty$. This path is not necessarily feasible by the car-like robot. It may be computed by any (holonomic) motion planner.

Let $\sigma : [0, \ell] \longrightarrow \mathbf{R}^2$ be such a path parameterized by arc length. ℓ is then the length of the path. We suppose

that the obstacles are polygons in the plane and we note n the number of line segments that compose the boundaries of these obstacles. The robot is assumed to be a polygon.

Pavement by balls: In order to cover the path σ by a sequence of balls, two operations are necessary. The first one computes the distance from a given configuration to obstacles, noted $d_{\mathcal{O}}$ and used to determine the size of the balls. This distance is taken to be the minimum of the n distances to the line segments that compose the set of obstacles. Computing the shortest path distance for a polygonal robot from a configuration to a straight line segment is still an open problem. The procedure we use in practice considers obstacles grown by the radius of the smallest circle enclosing the robot. It is then based on the results appearing in [14] and giving the shape of the shortest path for a pointwise car-like robot to either a point or segment; the distance to arcs of circles will be developed in an appendix in the final version of the paper.

The second operation is the distance d_{RS} between configurations associated to shortest paths. It is used to check the overlapping of consequent balls and thus to maintain the connectivity of the band. This second distance requires a constant time and is independent of both ℓ and n.

The pavement of the path σ by balls has been introduced in [5]. We start with the path σ. This path is firstly sampled into a sequence of configurations. The number of the resulting configurations, noted l, is proportional to the length of the both, ℓ, and the resolution of the sampling. Then the first bubble is defined to be the greatest ball centered on the first configuration of the sequence. The sequence is then spanned until the last configuration still belonging to the last constructed bubble is reached. This configuration is thus taken to be the center of the following bubble which also has the maximum size. The previous step is repeated till the goal (or last) configuration is found to be in a bubble. A final ball is created around it. Figure 4 shows the pavement covering the path of Figure 3; the drawings correspond to the projection of the (3-dimensional) balls onto the plane.

Let b be the number of balls that compose the created band. The distance $d_{\mathcal{O}}$ was used b times to determine the size of these balls while the distance d_{RS} was used $l-1$ times for the inclusion test. The complexity of this operation is then $O(l+nb)$.

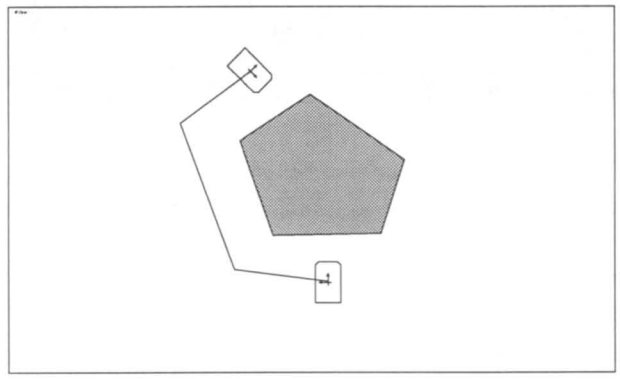


Figure 3: *Initial path.*

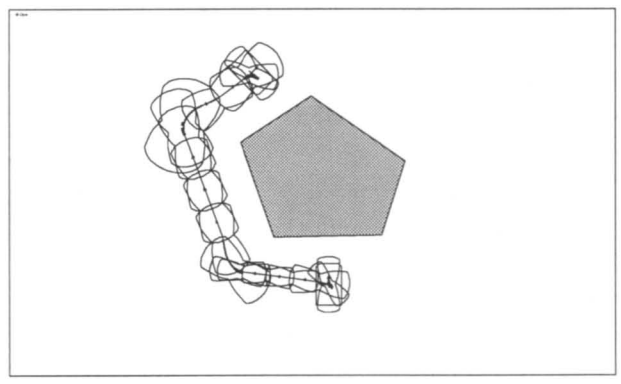


Figure 4: *Pavement with balls*

Pavement by stars: This pavement requires the answer to the question: 'is a given configuration "visible" from a starting one ?', i.e. 'is the shortest path between them collision-free ?' To answer this question we use the collision-checker described in [7]. It works for polygonal robot and its complexity is $O(n)$.

A pavement by stars can be computed with the same algorithm as the pavement by balls. Starting from the initial configuration, we search for the first configuration on σ which is not visible; then the configuration

before becomes the center of a new star. The complexity of the algorithm is then $O(nl)$. Usually, $l \gg b$ and the complexity of this algorithm is much worse than the first one.

A better algorithm may be devised by using the inclusion of $\mathcal{B}(C, d_C)$ in $\mathcal{E}(C)$. It is composed of two steps. In the first one, we construct a ball band. Then the second step consists in spanning this band and in replacing the balls by stars. Whenever two stars are connected (i.e. the center of the first is in the second) all the intermediate bubbles are removed. The first step has the same complexity as the pavement by balls. The second step has a complexity equal to $O(nb)$. Finally, the total complexity remains $O(l + nb)$.

Figure 5 shows the result of the algorithm: four stars are sufficient to cover the initial path; the first four configurations appearing in the figure are the centers of the four stars.

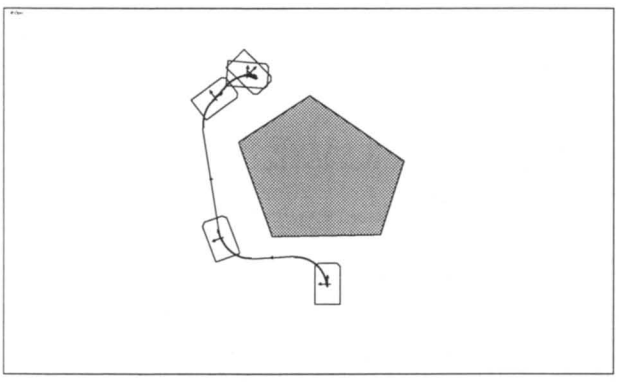


Figure 5: *Pavement with stars.*

From pavements to collision-free feasible paths: By definition of the bubbles, in both pavements the shortest path between the centers of two adjacent bubbles (balls or stars) is collision-free. Then the sequence of the shortest paths linking iteratively the centers of adjacent bubbles constitutes a collision-free feasible path, i.e. a solution to the nonholonomic path planning problem.

Complexity of the pavements: We define the complexity of a pavement by its number of bubbles. This notion is closely related to the complexity of the nonholonomic motion planning problem.

While in classical (holonomic) path planning, the complexity of the problem only depends on the complexity of both the robot and the environment (e.g., number of polygons edges), it is known that nonholonomy requires to account for the size of the free space. For instance, it has been proved in that the number of maneuvers necessary to park a car between two other ones is in $\Omega(\epsilon^{-2})$ where ϵ is the clearing distance: distance between both fixed cars (in black) minus the length of the moving car (in white) (see Figure 6). This result is a consequence of the shape of the nonholonomic metric in the neighborhood of a configuration. It is derived from a complexity model consisting in paving a path by nonholonomic balls [1, 7], exactly as provided by the first pavement above. Nevertheless this model of complexity accounts only for the worst cases (i.e., when the size of the free space is very small); in such cases the pavement by balls is optimal.

The pavement by stars suggests another model of complexity for nonholonomic path planning: the complexity of a path is the number of stars necessary to cover it. Figure 7 illustrate the interest of this model: in a corridor the number of balls necessary to cover the straight line segment depends linearly on the length of the corridor, while the solution has clearly a constant cost. The pavement by stars accounts for this constant cost: only one star is necessary.

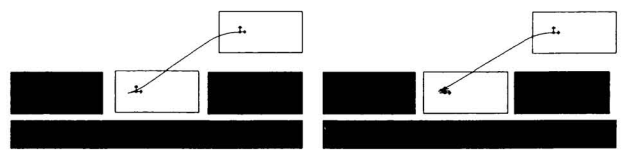


Figure 6: *The number of maneuvers varies as the inverse of the square of the clearing distance.*

In most cases the number of stars used to cover a path is considerably smaller that the necessary number of balls (compare Figures 4 and 5). Due to the inclusion of $\mathcal{B}(C, d_C)$ in $\mathcal{E}(C)$, it is always smaller and it is equivalent for the worst cases.

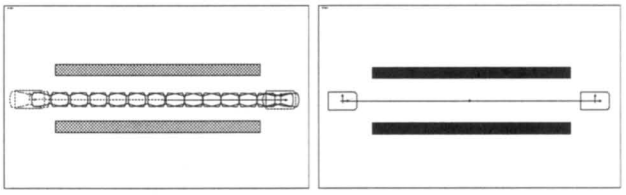


Figure 7: *A ball band and a star band in a corridor.*

Note: The algorithms above assume that the initial path has been firstly sampled into a sequence of l configurations. Then their complexity depends on l. It seems that another way to build both pavements would consist in directly computing the intersection of σ with the balls and the stars respectively. The shape of the balls and the shape of the stars can be analytically computed (see Figures 1 and 2). It remains that their frontier is too complex to hope for exact intersection algorithms even with simple curves such as lines.

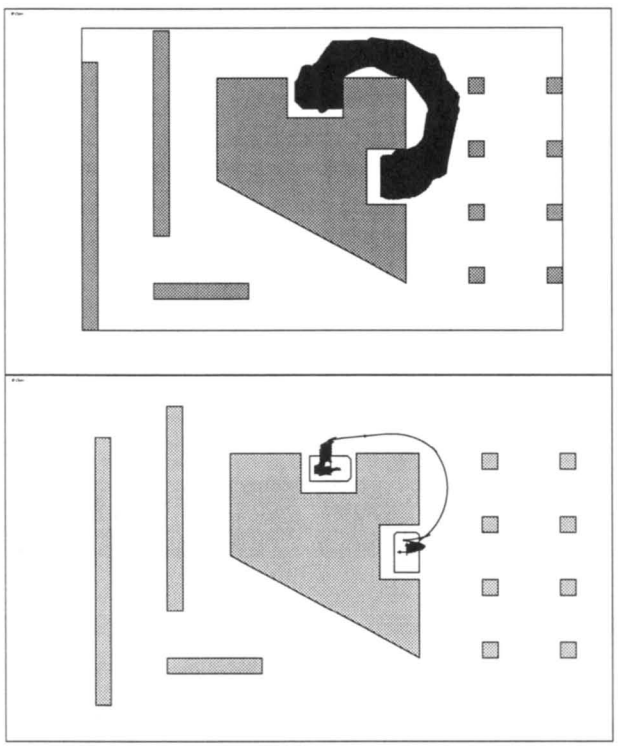


Figure 8: *A holonomic path (top) and the nonholonomic path provided by the star band (bottom).*

An alternative way to build a pavement by stars can be done by using a dichotomy procedure as it appears in the approximation step of the nonholonomic path planner described in [7]. We first try to reach the goal directly from the initial configuration by the shortest feasible path. If this path intersects an obstacle, the configuration halfway along σ is used as a subgoal. The problem is thus broken into two subproblems. If, at any point, the minimal length path between subgoals is not collision-free, then we recursively subdivide σ. When all of the necessary subdivisions are completed, the stars centered at the intermediate points constitute a pavement of σ. The advantage of this method is to not require any sampling of σ. Nevertheless, experiments shows that the convergence of the algorithm is in general slower than the algorithm above. For instance the planner by dichotomy of the holonomic path of Figure 8 (top) requires 10981 subdivisions (computed in more than 1 minute[1]) while our algorithm requires only 500 initial balls which are reduced to 66 stars constituting the band of Figure 8 (bottom). The computation time was 0.8 seconds.

4 Bubble Band Optimization: Elastic Band

Once a first pavement created, we consider the associated collision-free feasible path (constituted by a sequence of shortest paths). The path together with its pavement constitutes the so-called bubble band. This section presents a machinery tending to optimize the form and length of the band and allowing to integrate environment changes dynamically.

The basic operations deal with the bubbles: bubbles may be modified individually (by moving their center), new bubbles may be added and redundant bubbles may be removed. In order to guarantee a feasible path, the bubble band must always verify the three following conditions:

[1]The holonomic paths presented in this paper have been computed by using the holomonic planner described in [2]. All the algorithms have been implemented and run on a `SPARCstation20`.

- The bubbles should belong to the free space (*non-collision condition*),
- Successive bubbles should be connected (*feasibility condition*),
- The bubble band should converge in a static environment (*stability condition*).

In [5], the reactivity of the ball band is governed through the balls' centers. This is possible because the balls are close together. This is not true for the star band where stars may be far from each other. Nevertheless, the creation strategy will make it possible to define forces on the star centers and still respect the above conditions. We begin by presenting an energy function that the star band has to optimize. We will show that all the operations applied to the band tend to minimize this energy in order to insure the convergence of the band.

Two configurations C_1 and C_2 being given, we denote by $\gamma(C_1, C_2)$ the shortest path between them. The length of $\gamma(C_1, C_2)$ is $d_{RS}(C_1, C_2)$. $\mathsf{clear}(\gamma)$ is the (Euclidean) distance between a path γ and the obstacles. Let $\mathcal{EB} = \{\mathcal{E}_i(C_i), i = 1, \ldots, n\}$ be a star band.

The energy is composed of two terms:

$$\mathcal{P}(\mathcal{EB}) = \mathcal{P}_{\rangle\backslash\sqcup}(\mathcal{EB}) + \mathcal{P}_{\rceil\S\sqcup}(\mathcal{EB}). \qquad (1)$$

The first, called the internal energy, is proportional to the length of the band (i.e., the length of the associated path):

$$\mathcal{P}_{\rangle\backslash\sqcup}(\mathcal{EB}) = \mathcal{A} \sum_{\rangle=\infty}^{\backslash-\infty} \lceil_{\mathcal{RS}}(\mathcal{C}_\rangle, \mathcal{C}_{\rangle+\infty}). \qquad (2)$$

The second term is proportional to the distance to obstacles and, consequently, is called the external energy:

$$\mathcal{P}_{\rceil\S\sqcup}(\mathcal{EB}) = \mathcal{B} \min_{\rangle=\infty,\ldots,\backslash-\infty} \sqcap(\mathsf{clear}(\gamma(\mathcal{C}_\rangle, \mathcal{C}_{\rangle+\infty}))), \qquad (3)$$

where u is a positive convex derivable function whose domain (i.e. the interval on which its value is not equal to zero) is some interval $[0, d_{min})$; d_{min} represents the limit of the obstacles influence. For example,

$$u(d) = \begin{cases} (d - d_{min})^2 & \text{if } d_{\mathcal{O}}(P, \mathcal{O}) < d_{min}, \\ 0 & \text{otherwise.} \end{cases}$$

We begin by presenting the operations of creating new bubbles and removing unnecessary ones. Then we will study the forces that are applied to the bubble centers.

4.1 Connectivity

In order to respect the accessibility condition, a star should be created between two successive stars before they become disconnected.

Let $\mathcal{EB} = \{\mathcal{E}(C_i), i = 1, \ldots, n\}$ be a connected star band. If $\mathsf{clear}(\gamma(C_i, C_{i+1}))$ becomes lesser than some threshold ϵ, we suppose that the two stars tend to get disconnected from one another. A configuration $C_{i+\frac{1}{2}}$ is then chosen on the path $\gamma(C_i, C_{i+1})$ to be the center of a new intermediate bubble.

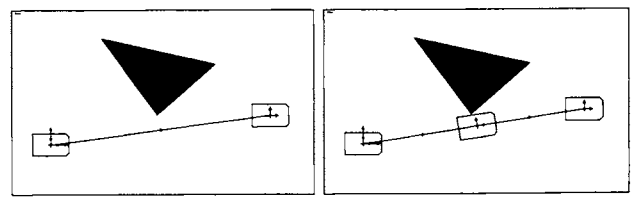


Figure 9: *Creation of a new star.*

Any configuration can be chosen on the path without modifying the band energy since the resulting path is not modified. If $C_{i+\frac{1}{2}}$ is not the closest configuration to the obstacles, either $\gamma(C_i, C_{i+\frac{1}{2}})$ or $\gamma(C_{i+\frac{1}{2}}, C_{i+1})$ will be as close to the obstacles as $\gamma(C_i, C_{i+1})$ and another bubble would then be created. In order to avoid this loop, we choose the new center to be the closest configuration on the original path to the obstacles and we forbid the creation of a new center if its distance to one of the surrounding ones is lesser than its distance to obstacles.

To do this, the collision checker of the polygonal robot along the shortest path (described in [7]) has been revisited to provide the configuration closest to obstacles (in case of non-collision).

4.2 Redundancy

A star is no longer necessary whenever its two neighbors become connected. It should then be eliminated.

Of course this elimination should not increase the energy of the band.

The elimination of a star reduces the length of the band and the internal energy decreases. On the other hand the resulting path may get closer to obstacles with a consequent increase in the external and maybe in the total energy of the band. In order to avoid this, a star is eliminated if and only if the adjacent bubbles become connected and if the center of the redundant star is close to the path between them.

Not only the creation and elimination operations does not increase the total energy, but also they does not produce a discontinuity in the behavior of the resulting path.

Note that the complexity of both operations is $O(nb)$ where n is the number of line segments that compose the obstacles and b is the number of bubbles in the band. In fact in the connectivity test the distance between configurations is computed $b-1$ times whereas the distance to obstacles is computed $b-1$ times in the redundancy elimination.

4.3 Potentials and forces

The internal structure of the bubble band is always the same whatever shape the bubbles take. It can be thought of as a sequence of configurations successively joined together by shortest paths which belong to the free space. The union of these paths form the global path that connect the start configuration to the goal.

The contraction force between two consecutive bubbles is taken to be the negative gradient of the internal energy.

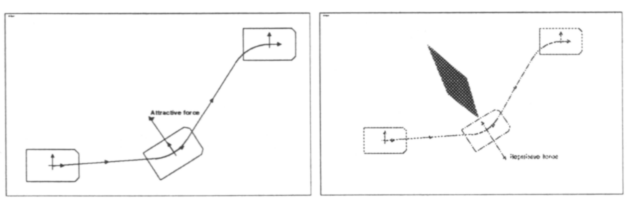


Figure 10: *Internal and external forces.*

To avoid obstacles, a force should be exerted on the stars for the global path to move away from them. Since the closest configurations to the obstacles on local paths are centers of stars (or else they are far from obstacles and not affected by them), it is sufficient to apply a repulsive force on these centers to obtain the desired result.

The sum of both forces makes the band move in the direction of energy decreasing. At each sampling step bubbles are created (to guarantee the connectivity) or removed. We have seen that these operations do not modify the total energy of the band. Then the energy of the band is always decreasing (it cannot oscillate). The three conditions (non collision, feasibility, and stability) are satisfied.

Finally, the complexity of the force computation and application is the same as that of bubble creation and elimination. While $b-1$ operations are necessary to compute the gradient of the $b-1$ local distance between centers, nb operations are necessary to compute the distance to obstacles. The total complexity is thus $O(nb)$.

4.4 Comments

Both ball and star bands optimization routines have been implemented. Two comments may be illustrated by Figures 11 and 12.

- The path associated to the minimum energy does not correspond to a minimal length shortest path in the presence of obstacles. This is only a "good" path making a compromise between minimal length and optimal clearance. Indeed the optimization routine above depends on the value of the constant real numbers A and B appearing in equations (2) and (3). The value of B for the solutions appearing in Figures 11 and 12 on the left is greater than the value used to compute the solutions appearing on the right.

- The same figures illustrate the interest of the stars with respect to the balls. In the case of balls, for the path length to decrease, the band is sometimes forced to get close to obstacles. The balls become then smaller and their number increases (from 8 to 11 in Figure 11). This incompatibility between

length and number of bubbles is less important in the case of stars, mainly because the stars are not bounded in diameter as are the balls (the number of stars remains constant in the examples shown in Figure 12).

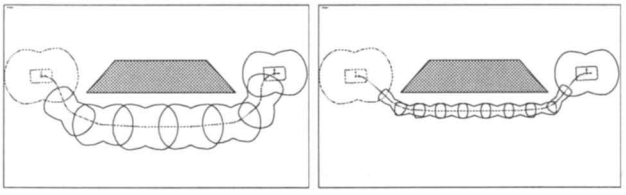


Figure 11: *Number of balls versus length of paths.*

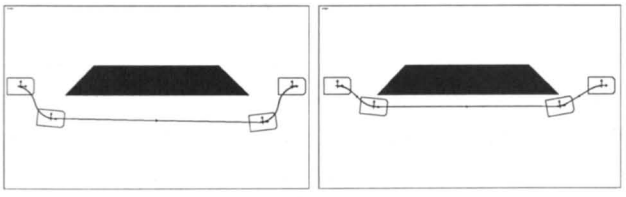


Figure 12: *Number of stars versus length of paths.*

5 Applications

Two main applications of the star bands are possible.

Holonomic path approximation: It is well known that car-like robots are small-time controllable (for any bounded domain of the free space, from any starting configuration lying in this domain, there exists a neighborhood of the starting configuration whose points are reachable without leaving the domain); as a consequence, any holonomic collision-free path can be approximated by a sequence of collision-free feasible paths. This idea is at the origin of the planner described in [7]. The elastic band appears as a good method for devising approximation routines. For instance we have seen that we can control the clearance to the obstacles just by tuning two parameters. Moreover the method is fast: the solution computed for the example in Figure 13 (bottom) takes 1.09 seconds to approximate and smooth the initial holomonic

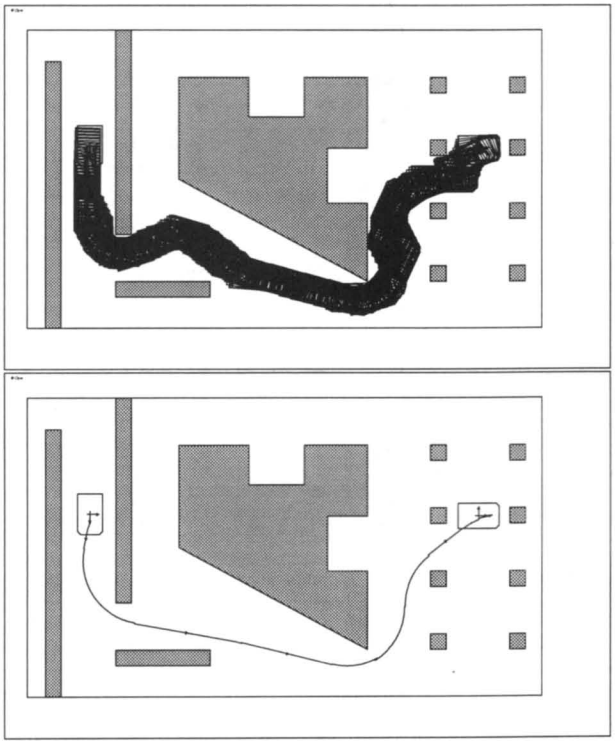


Figure 13: *Smoothing a holonomic path with the elastic band.*

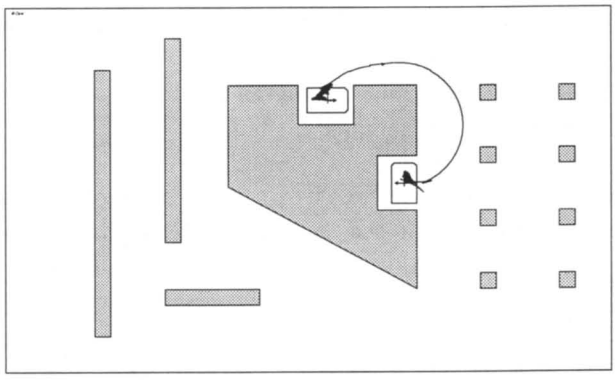


Figure 14: *Smoothing of the path in Figure 8 with the elastic band.*

path (Figure 13, top); it contains just one cusp. Figure 14 shows the smoothing of the first nonholonomic path appearing in Figure 8. Finally Figure 15 shows the smoothing of a nonholonomic path who reaches in

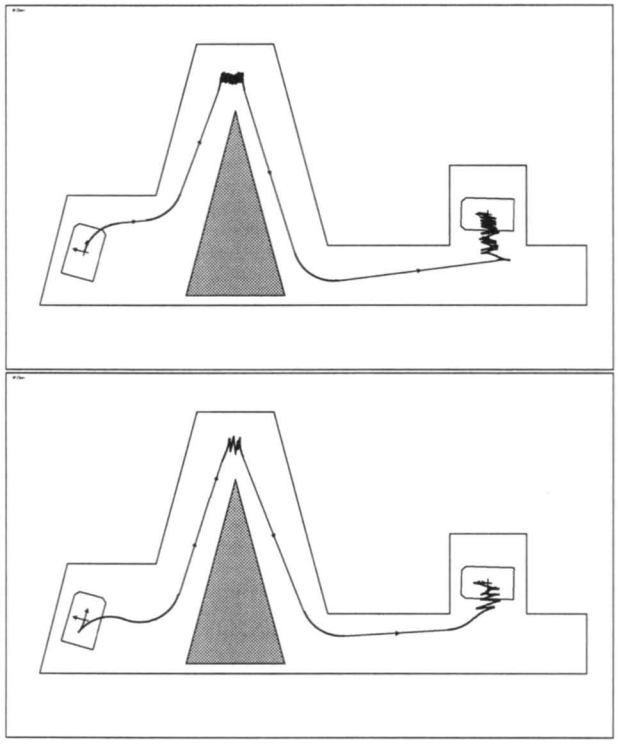


Figure 15: *Smoothing of a nonholonomic path with the elastic band.*

two places the maximal complexity. This is a typically difficult case: in fact we have tried to apply the method combining dichotomy and random smoothing described in [7] with constant failure.

Reactiveness to environment changes: The main advantage of the star elastic band is to allow better reactivity than the ball band. The method is sufficiently fast to allow reactiveness to environment changes. Figure 16 (top left) shows a first path provided by the application of the star band (the intermediate positions correspond to the star centers). When we add an unexpected obstacle (black square) the elastic band moves providing a new admissible and collision-free path. The reactiveness of the optimization routine above is sufficient to move the black square with the mouse: the new path is computed in real time. Figure 17 shows another example illustrating the fact that the star band can work in very cluttered environment and can generate automatically maneuvers when needed. The reactiveness works well as long as a path exists that is topologically equivalent. When this is no longer the case, a new path should be planed. If the black square would continue to move in the same direction, there would be no more solution: this event is detected by the elastic band method when the creation of a new star becomes impossible.

6 Conclusion

We have seen that the dynamic property of the star bands presented in this paper allows to improve the capacity of any existing path planner by accounting for the clearance to the obstacles and environment changes. This property may be also exploited during the motion execution phase.

Such exploitation is currently under study on the mobile robot Hilare 2 developed at LAAS-CNRS. It should replace the application of the balls band in this context, as it has been described in [6]. The main interest of the application of the bubble bands in execution phase is twofold: it allows to absorb the uncertainties on both the real environment (w.r.t. to its geometric model) and the real motion; moreover it allows to take into account in real time unexpected moving obstacles (whose dynamics should be of course at the same scale as those of the band).

The algorithmic issue that remains to be explored to improve the method concerns the computation of the shortest paths to the obstacles for a *polygonal* robot.

From a formal point of view, the pavement of paths by (nonholonomic) stars opens new instances of classical problems, such as the art gallery problem: how many car-like robots are necessary to cover a given space ?

Finally we have seen that the number of stars covering a given path gives a better complexity model of the nonholonomic paths than the balls computed in the singular metrics. How this model can be extended to more general nonholonomic systems ?

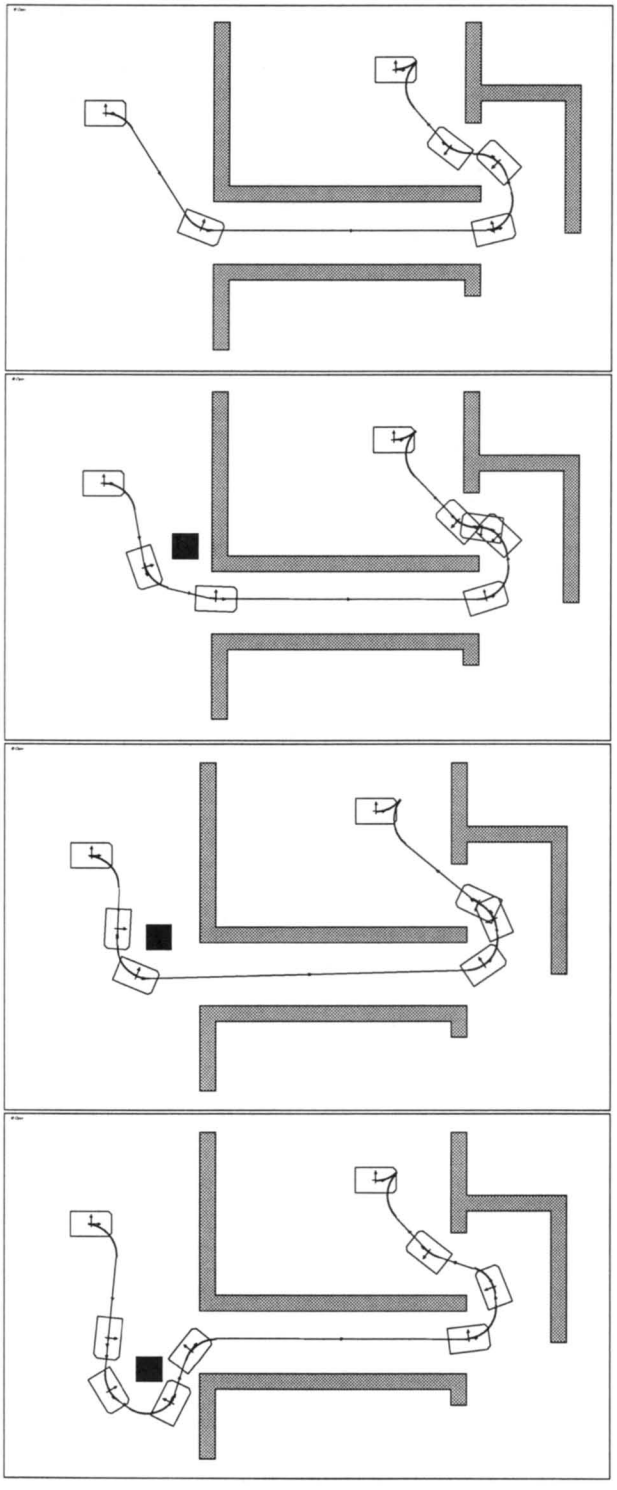


Figure 16: *Reactiveness to unexpected obstacles.*

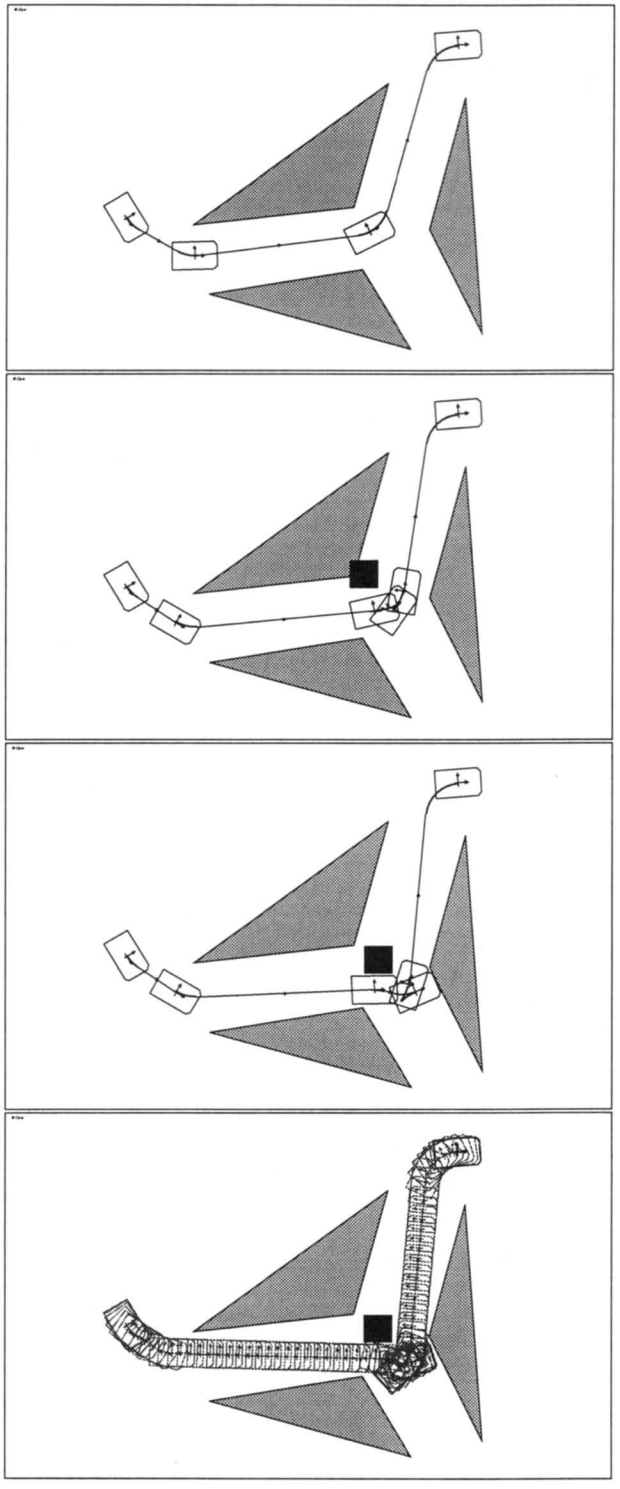


Figure 17: *Reactiveness to unexpected obstacles.*

References

[1] A. Bellache, J.P. Laumond, and P. Jacobs. Controllability of car-like robots and complexity of the motion planning problem. *International Symposium on Intelligent Robotics*, pages 322–337, 1991.

[2] J. Barraquand, and J.C. Latombe Robot motion planning: a distributed representation approach. *International Journal of Robotics Research*, 1991.

[3] J.D. Boissonnat, A. Cerezo, and J. Leblond. Shortest paths of bounded curvature in the plane. In *IEEE, International Conference on Robotics and Automation, Nice (France)*, pages 2315–2320, May 1992.

[4] H. Jaouni. *tude d'une mtrique non-holonome et applications aux robots mobiles.* PhD thesis, Universit Paul Sabatier, Toulouse, December 1997.

[5] M. Khatib, H. Jaouni, R. Chatila, and J.P. Laumond. Dynamic path modification for car-like non-holonomic mobile robots. In *IEEE, International Conference on Robotics and Automation, Albuquerque (USA)*, pages 2920–2925, April 1997.

[6] M. Khatib, H. Jaouni, R. Chatila, and J.P. Laumond. How to implement dynamic paths. In *5th International Symposium on Experimental Robotics, Barcelone (Espagne).*, pages 225–236, June 1997.

[7] J.P. Laumond, P. Jacobs, M. Taix, and R. Murray. A motion planner for non-holonomic mobile robot. *IEEE, Transactions on Automatic Control.*, 10, 1994.

[8] S. Quinlan. *Real-Time Collision-Free Path Modification.* PhD thesis, Stanford University, CS Departement, January 1995.

[9] S. Quinlan and O. Khatib. Elastic bands: connecting path planning and control. In *IEEE, International Conference on Robotics and Automation, Atlanta (USA)*, 1993.

[10] J.A. Reeds and L.A. Shepp. Optimal paths for a car that goes both forwards and backwards. *Pacific Journal Mathematics*, 145(2):367–393, 1990.

[11] P. Souères, J.Y. Fourquet, and J.P. Laumond. Set of reachable positions for a car. *IEEE, Transactions on Automatic Control.*, 39(8), August 1994.

[12] P. Souères and J.P. Laumond. Shortest paths synthesis for a car-like robot. *IEEE, Transactions on Automatic Control.*, 41(5), May 1996.

[13] H.J. Sussmann and W. Tang. Shortest paths for the reeds and shepp car: a worked out example of the use of geometric techniques in nonlinear optimal control. Technical Report SYSCON-91-10, Rutgers Center for systems and Control, 1991.

[14] M. Vendittelli and J.P. Laumond. Visible position for car-like robot amidst obstacles. In *Algorithms for Robotic Motion and Manipulation, 1996 WAFR*, J.P. Laumond and M. Overmars Eds, A.K. Peters, 1997.

Trajectory Generation for Mechanical Systems with Application to Robotic Locomotion

Richard M. Murray, Joel W. Burdick, Scott D. Kelly, James Radford
California Institute of Technology, Pasadena, CA, USA

This paper gives a survey of current and emerging techniques for motion control of nonlinear mechanical systems, motivated by applications in robotic locomotion. For this class of systems, internal changes of shape and/or application of body fixed forces are the mechanism by which the robot moves in its environment and the geometric mechanical properties of the system are crucial in understanding how specific gaits for generating motion can be obtained.

1 Introduction

We define a *robotic locomotion system* as a robotic device that is capable of propelling itself in its environment. A typical example of a locomotion system is a wheeled mobile robot, such as the Hilare mobile robot depicted in Figure 1. This system interacts with its environment through the constraints imposed by the wheels touching the ground. Under normal operation, the wheels roll without slipping, allowing the robot to apply forces against its environment and propel itself.

The interaction of a mobile robot with its environment is clearly essential in order for the robot to locomote. It turns out that in a variety of instances, this interaction can be modelled as a Pfaffian constraint of the form

$$\omega_i(q)\dot{q} = 0 \qquad i = 1, \ldots, k \tag{1}$$

where q is the configuration of the robot and k is the number of constraints between the robot and its environment. This is precisely the type of constraint that arises in grasping problems, but here one is not concerned with manipulating an object, but rather "manipulating" the robot itself. As a consequence, the

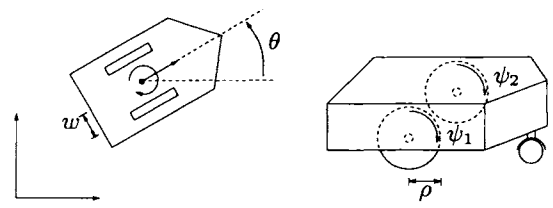


Figure 1: *The Hilare mobile robot.*

types of questions that one poses in locomotion sytems are different than the common questions in grasping.

The constraints in equation (1) are constraints on the velocity of the robot. Each ω_i indicates a configuration dependent direction in which the robot cannot move. We are primarily interested in the case where the constraints imposed on the system are *nonholonomic*, meaning that they do not constrain the actual configuration of the robot. This is clearly the case with the Hilare robot, which can move to an arbitrary position and orientation in the plane.[1] Asking that the constraints be nonholonomic is equivalent to asking that the system be *controllable*: given an initial and final configuration for the system, there exists a trajectory that connects them.

The problem of actually finding a path that connects two configurations is sometimes referred to as *constructive controllability*. Like the robot path planning problem, one is often required to find trajectories that avoid a set of obstacles in the configuration space. The additional limitation imposed by the constraints (1) makes this problem harder, since one can no longer choose any path in configuration space that connects the initial and final points.

[1]This is shown in detail in Example 1.

An alternative to studying the constructive controllability problem is to try to construct *gaits* for the system. Roughly speaking, a gait corresponds to some cyclic change in the control variables that generates motion in the environment in some direction. We shall see in the sequel that gait behavior for robotic locomotion systems is often closely tied to the controllability structure of the system, allowing additional insights between the geometric properties of the constraints and the achievable motion of the system.

The goal of this paper is to describe a framework for understanding robotic locomotion problems and studying controllability, trajectory generation, and gaits. In order to do so, we will first give a very brief description of geometric mechanics, focusing on the use of "connections" to describe the constraints between a robot and its environment. This point of view allows a unified view of a wide variety of locomotion systems, at the expense of learning a bit of abstract mathematics (connections on principal bundles). More details on the specific mathematical results illustrated in this paper can be found in [3], [9] and the references therein.

Whenever possible, we follow the notation of [10]. In particular, we will make use of the Lie group $G = SE(3)$ to represent motion of a robot in its environment and $\mathfrak{g} = se(3)$ to represent the space of (body) velocities of a locomotion system. Readers unfamiliar with rigid body kinematics in this setting are referred to Chapter 2 of [10].

2 Connections on Principal Bundles

Consider a mechanical system with velocity constraints of the form given in equation (1). Let Q represent the configuration space of the system and suppose we can find coordinates on Q of the form $q = (r, s)$ such that the constraints become

$$\dot{s} + a_i(r)\dot{r} = 0 \qquad r = 0, \ldots, k. \tag{2}$$

The special property of this system is that the constraints are independent of s (this is not true in general). We call the r the *base* coordinates and s the *fiber* coordinates for the system (this terminology will be explained later).

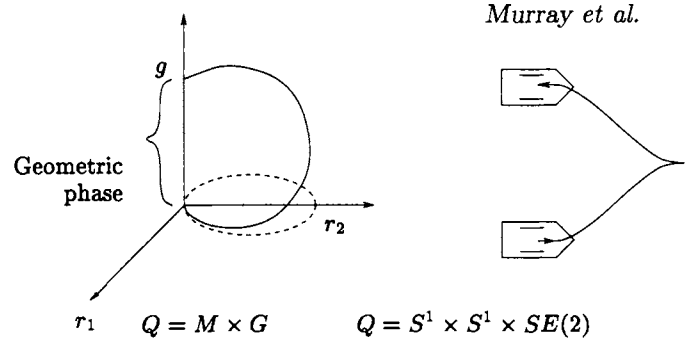


Figure 2: *Holonomy associated with a closed path in base space.*

For many locomotion systems, it will be the case that we can directly control the motion in the base space, parameterized by r. Suppose now that we want to move from an initial configuration $q_0 = (r_0, s_0)$ to a final configuration $q_f = (r_f, s_f)$. This can be done by separating the problem into two steps. First, we choose a path in the base space that steers the system from (r_0, s_0) to a new configuration (r_f, s_i), where s_i is the intermediate position that one obtains by integrating equation (2) using the chosen base space path (any path in the base space will do; a straight line is perhaps the simplest).

The second step is to try to find a closed path in the r coordinates such that

$$s(T) = \int_0^T a(r(t))\dot{r}(t)\,dt = s_f,$$

where $r(0) = r(T) = r_f$. This integral is called the *holonomy* associated with the base space path and it gives the motion in the "fiber" variables, s, associated with the closed path in base space. If we can find such a path, then we will have succeeded in moving the system from its initial configuration to the desired final configuration. This second step is depicted graphically in Figure 2.

The situation above is a special case of a connection on a principal bundle. We now show how to extend this basic idea to a more geometric context that is applicable to locomotion sytems. A much more complete development can be found in [3].

Consider a locomotion system with configuration space $Q = R \times G$ where G is either $SE(3)$ or one of

its proper subgroups ($SO(3)$, $SE(2)$, etc) and R is the remaining variables for the system. We call R the *base space* or *shape space* for the system and G the *group* or *fiber* variables. We assume the constraints for the system are in the form given by equation (1). If the constraints are constant in the reference frame of the vehicle, then it turns out that they can be written in the form

$$\Gamma(q)\dot{q} = \mathrm{Ad}_g(g^{-1}\dot{g} + A(r)\dot{r}) = 0 \tag{3}$$

where $g \in G$ is the position and orientation of the robot and r is the shape. The operator Ad_g is the adjoint operation, an invertible linear transformation (see [10] for a description and explicit formulas). Although the adjoint could clearly be ommited without affecting the constraints, we included it to obtain the proper mathematical description of the constraints (see [3]).

The operator Γ is called a *connection*: it describes how trajectories in the base space R are related (connected) to the motion in the group G. It is the equivalent of the constraints (2) that allowed paths to be lifted from the base space to the fiber. Here, the fiber corresponds to the position and orientation of the vehicle, and hence the connection describes how changes in the shape of the robot affect the position of the robot within that environment. The term $A(r)$ is called the *local connection one form*. Formally, it is a map between the velocity in the shape space and the body velocity of the system.

Although this form of the constraints is nothing more than a rewriting of the constraints in a particular way, it uncovers the essential geometry of the constraints and decouples the group variables from the shape variables. This is illustrated in the following example.

Example 1 Consider the Hilare mobile robot shown in Figure 1. Let (x, y, θ) represent the position and orientation of the vehicle and let ψ_1 and ψ_2 represent the angles of the left and right wheels. The constraints on this system are that the individual wheels cannot

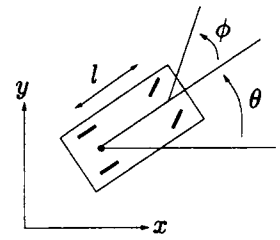


Figure 3: *A simple model of an automobile.*

slide sideways, yielding constraints of the form:

$$\begin{aligned}
\omega^1(q)\dot{q} &= \dot{x}\cos\theta + \dot{y}\sin\theta - \frac{\rho}{2}(\dot{\psi}_1 + \dot{\psi}_2) = 0 \\
\omega^2(q)\dot{q} &= -\dot{x}\sin\theta + \dot{y}\cos\theta = 0 \\
\omega^3(q)\dot{q} &= \dot{\theta} - \frac{\rho}{2w}(\dot{\psi}_1 - \dot{\psi}_2) = 0.
\end{aligned} \tag{4}$$

By rearranging the equations above, we can regroup the constraints in the form

$$\Gamma(q)\dot{q} = \begin{bmatrix} \dot{x}\cos\theta + \dot{y}\sin\theta \\ -\dot{x}\sin\theta + \dot{y}\cos\theta \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} -a & -a \\ 0 & 0 \\ -b & +b \end{bmatrix} \begin{bmatrix} \dot{\psi}_1 & \dot{\psi}_2 \end{bmatrix} = 0.$$

The first term in the sum represents the body velocity of the robot ($g^{-1}\dot{g}$) and the second term is the local connection one form.

Exercise 1 Figure 3 depicts a kinematic model of an automobile on the plane. For simplicity's sake, we treat the front and rear pairs of wheels as single wheels at the midpoints of their respective axles. We parametrize the shape space $R \subset S^1 \times S^1$ by the steering angle ϕ and the rotation angle ψ of the rear wheel. We control the system by specifying these angles for all time; we will not need the rotation angle of the front wheel. We define ψ to increase as the car moves forward. We denote the radius of the wheels by ρ and the distance between their centers by l.

Motion of the car is restricted at any point by the specification that the wheels can roll but cannot slide on the plane. Show that the connection for this system is given by

$$A(r) = \begin{bmatrix} -\rho & 0 \\ 0 & 0 \\ 0 & -(\rho/l)\tan\phi \end{bmatrix}$$

where $r = (\phi, \psi)$.

3 Controllability of Locomotion Systems

We now consider the problem of determining if a locomotion system is controllable, in the sense that we can move from one configuration to another by appropriate choice of path for the base variables (which we will assume we control in a direct manner). The material in this section is derived from [3].

Let $Q = R \times G$ represent the configuration space for our system and assume the base dynamics are fully actuated, so that we can follow any path in r. We assume that the constraints on the system are of the form

$$\dot{g} = g(-A(r)\dot{r}) \tag{5}$$

where the local connection form A describes how the shape of the robot interacts with its environment. Our goal is to study the controllability of the system described by equation (5) in terms of the properties of the local connection form $A(r)$. This has the advantage of properly taking into account the role of the Lie group G and thus avoids the use of local coordinates, which can lead to extremely messy formulas.

We now make precise the notion of controllability for a locomotion system. There are two cases to consider: controllability on the entire space $Q = R \times G$ and controllability only on the fibers.

Definition 1 A locomotion system is said to be *totally controllable* if, for any $q_0 = (r_0, g_0)$ and $q_f = (r_f, g_f)$, there exists a time $T > 0$ and a curve $r(\cdot)$ connecting r_0 and r_f in the base space such that the solution curve for (5) passing through q_0 satisfies $q(0) = q_0$ and $q(T) = q_f$.

Definition 2 A locomotion system is said to be *fiber controllable* if, for any initial position $g_0 \in G$, final position $g_f \in G$, and initial shape $r_0 \in R$, there exists a time $T > 0$ and a base space curve $r(\cdot)$ satisfying $r(0) = r_0$ such that the solution curve for (5) passing through (r_0, g_0) satisfies $q(0) = q_0$ and $q(T) = (r(T), g_f)$ for some $r(T) \in R$.

In order to determine controllability, we make use of the *curvature* of the connection form $A : \dot{r} \mapsto \mathfrak{g} = se(3)$:

Definition 3 Given a local connection form A on Q, we define the corresponding local *curvature form* to be the set of $\mathfrak{g}$-valued vectors satisfying

$$F_{ij} = \frac{\partial A_i}{\partial r_j} - \frac{\partial A_j}{\partial r_i} + [A_i, A_j] \qquad i, j = 1, \ldots, m \tag{6}$$

where $[\cdot, \cdot]$ denotes the Lie bracket on $\mathfrak{g}$ (see [10]) and m is the dimension the shape space M.[2]

The curvature form measures the integrability of the kinematic constraints. In particular, if the constraints are holonomic then the curvature is identically zero.

To determine controllability for a locomotion system, we use a standard theorem in nonlinear control theory known as Chow's theorem (see [10]). Using the left-invariance properties of the connection, the structure of the equations of motion, and various identities from differential geometry it is possible to rewrite the conditions of Chow's theorem in terms of the curvature. Define the following sequence of subspaces of the Lie algebra $\mathfrak{g}$ at a fixed point $r \in M$:

$$\begin{aligned}
\mathfrak{h}_1 &= \operatorname{span}\{A_i : i = 1, \ldots, m\} \\
\mathfrak{h}_2 &= \operatorname{span}\{F_{ij} : i, j = 1, \ldots, m\} \\
\mathfrak{h}_3 &= \operatorname{span}\{\frac{\partial F_{ij}}{\partial r_k} + [F_{ij}, A_k],\ [F_{ij}, F_{kl}], \\
&\qquad\qquad i, j, k, l = 1, \ldots, m\} \\
&\ \ \vdots \\
\mathfrak{h}_k &= \operatorname{span}\{\frac{\partial \xi}{\partial r_k} + [\xi, A_k],\ [\eta, \xi] : \\
&\qquad\qquad k = 1, \ldots, m,\ \eta \in \mathfrak{h}_2 \oplus \cdots \oplus \mathfrak{h}_{k-1}\}.
\end{aligned} \tag{7}$$

Theorem 1 *[3] A system defined on a trivial principal bundle Q over R with structure group G and local connection $A(r)$ is* locally fiber controllable *near $q \in Q$ if and only if*

$$\mathfrak{g} = \mathfrak{h}_1 \oplus \mathfrak{h}_2 \oplus \mathfrak{h}_3 \oplus \cdots .$$

[2]The standard definition of the curvature form, written DA in [3], actually corresponds $-F_{ij}$, but the sign convention here is better tuned to the series expansion in the next section.

The system is locally totally controllable *if and only if*

$$\mathfrak{g} = \mathfrak{h}_2 \oplus \mathfrak{h}_3 \oplus \cdots .$$

This proposition again highlights the mechanical nature of the system and exploits the extra structure provided by the Lie group. More details and additional examples, including snakelike and legged locomotion systems, are provided in [3]. Control of locomotion systems that rely on the interaction between constraints and symmetries are described in [12].

Example 2 Consider again the mobile robot introduced in Example 1. If we redefine the base coordinates as $\phi_1 = \psi_1 + \psi_2$ and $\phi_2 = \psi_1 - \psi_2$, then the local connection form for the system is

$$A = \begin{bmatrix} -a & 0 \\ 0 & 0 \\ 0 & -b \end{bmatrix},$$

and the local curvature form is

$$F_{11} = F_{22} = 0$$

$$F_{12} = -F_{21} = \begin{bmatrix} 0 \\ -ab \\ 0 \end{bmatrix}.$$

The vectors A_1 and A_2 span $\mathfrak{h}_1$, and the vector F_{12} spans $\mathfrak{h}_2$. Since these three vectors are linearly independent, the system is fiber controllable. Since these are all constant vectors and the third component of the Lie bracket of any two elements of $se(2)$ is zero, no element of $\mathfrak{h}_k$, $k = 3, 4, \ldots$, will have a nonzero third component. The system cannot, therefore, be totally controllable. The reason for this is clear: (locally) the angle of the robot directly determines the difference between the wheel angles and hence we always have $\theta = \rho\phi_2$. Of course, since we usually are not interested in specifying the internal angles of the wheels, the fact that the system is not totally controllable is of no practical consequence.

Exercise 2 Consider the kinematic car in Exercise 1. Show that

$$\mathfrak{h}_1 = \operatorname{span} \begin{bmatrix} \rho \\ 0 \\ (\rho/l)\tan\phi \end{bmatrix},$$

$$\mathfrak{h}_2 = \operatorname{span} \begin{bmatrix} 0 \\ 0 \\ (\rho/l)\sec^2\phi \end{bmatrix},$$

$$\mathfrak{h}_3 = \operatorname{span} \left(\begin{bmatrix} 0 \\ (\rho^2/l)\sec^2\phi \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 2(\rho/l)\sec^2\phi\tan\phi \end{bmatrix} \right),$$

and

$$\mathfrak{h}_4 = \operatorname{span} \left(\begin{bmatrix} 0 \\ 0 \\ \frac{2\rho}{l}(\sec^4\phi + 2\sec^2\phi\tan\phi) \end{bmatrix}, \begin{bmatrix} 0 \\ \frac{2\rho^2}{l}\sec^2\phi\tan\phi \\ 0 \end{bmatrix}, \begin{bmatrix} \frac{\rho^3}{l^2}\sec^4\phi \\ 0 \\ 0 \end{bmatrix} \right).$$

Hence the system is locally strongly controllable because $\mathfrak{h}_4$ contains three linearly independent vectors away from $\phi = \pi/2$.

4 Gait Behavior

One of the common characteristics of locomotion systems is the use of *gaits* to generate motion. For the purpose of this paper, we shall define a gait as any closed path in shape space. Figure 4 shows three gaits for a mobile robot with fixed orientation, modeled by the connection

$$A(r) = \begin{bmatrix} 0 & -r\cos\theta \\ 0 & -r\sin\theta \end{bmatrix},$$

(see [3] for more information).

The properties of the connection one form can be used to understand the gait behavior of a locomotion system. Indeed, the holonomy associated with a closed path precisely describes the motion that one obtains by executing a given gait. This relationship can be demonstrated using an expansion for the motion in the group variables as a function of path followed in the

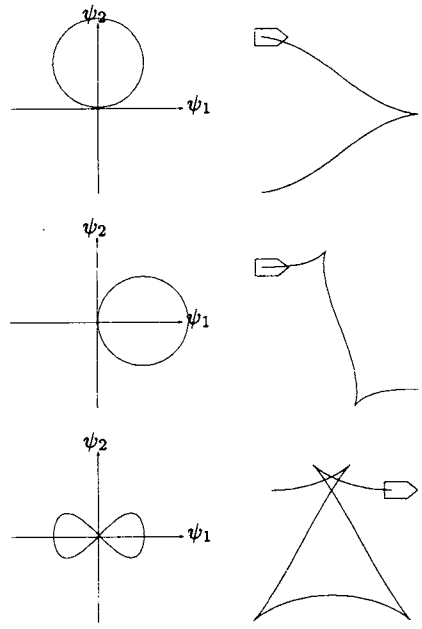


Figure 4: *Gait behavior for a mobile robot with fixed orientation.*

base space. We summarize here the work of Radford and Burdick [13].

Let $g \in SE(3)$ represent a position and orientation. Using the exponential map, it is possible to write this rigid motion in "exponential coordinates", $z \in se(3)$, defined as

$$g = e^z.$$

Roughly speaking, the exponential coordinates can be thought of as a constant body velocity which, when followed for one unit of time, results in the desired motion.

In terms of exponential coordinates, Radford derived a formula for the holonomy associated with a closed path. Let $r(t)$, $0 \leq t \leq T$ be a closed path of duration T and let $A(r)$ be the local conection form associated with the locomotion system. The motion in the group variables is given by

$$\begin{aligned} z(\alpha) = \frac{1}{2}F_{ij}\int_\alpha dr^i dr^j \\ - \frac{1}{3}(F_{ij,k} + [F_{ij}, A_k])\int_\alpha dr^i dr^j dr^k \\ + \text{higher order terms} \end{aligned} \tag{8}$$

Note that the first term is comprised of the elements that span $\mathfrak{h}_2$, the second term spans $\mathfrak{h}_3$, etc.

This formula provides a tremendous amount of geometric insight into the gait structure for locomotion sytsems. Given any path in the base space, we can compute the integrals indicated in the formula and then multiply by the coefficient, which is precisely the curvature and other terms that were derived in the controllability analysis. Although this formula only holds locally, it is often the case that very large changes of shape give qualitatively similar motions.

To illustrate the use of this formula, Table 1 shows the integrals associated with using integrally related sinusoids as inputs for a two input system. The elements in the right hand section of this table hold for *any* locomotion system, while the elements in the left hand section show the computations for two specific systems (the amoeba example will be described in the next section). To determine the motion due to a certain gait, add the contributions for each row in the given column (not that higher order terms are omitted, thus R^1 and R^2 should be sufficiently small).

Exercise 3 Use Table 1 to determine the required motions in the shape space of the kinematic car to produce a pure rotation and a sideways translation. Use these computations to propose an algorithm for parallel parking a car.

5 Locomotion in Fluids

To illustrate the generality of some of the ideas presented above, we now give a brief description of how the basic concepts can be used to study certain problems in underwater locomotion. For more information on the material introduced in this section, see [4, 2, 5].

We concentrate on a deformable body immersed in an inviscid, irrotational, incompressible fluid. To illustrate how locomotion can be achieved for such a system, consider the underwater vehicles shown in Figure 5, based on the work of Saffman [14]. The vehicle consists of a deformable body that can change shape along a one parameter family of ellipsoids and shift an internal mass.

Inviscid amoeba	Hilare robot	Direction of motion	$R^1 t$ 0	0 $R^2 t$	$R^1 \sin nt$ $R^2 \cos nt$	$R^1 \cos nt$ $R^2 \sin nt$	$R^1 \cos nt$ $R^2 \sin 2nt$
$\epsilon^2 k_2$	$[\frac{\rho}{2}, 0, \frac{-\rho}{w}]$	$-A_1$	$2R^1\pi$	0	0	0	0
$-\epsilon^2 k_1$	$[\frac{\rho}{2}, 0, \frac{\rho}{w}]$	$-A_2$	0	$2R^2\pi$	0	0	0
$-\epsilon^2$	$[0, \frac{-\rho^2}{2w}, 0]$	$\frac{1}{2}F_{12}$	0	0	$-R^1R^2n\pi$	$R^1R^2n\pi$	0
ϵ^2	$[0, \frac{\rho^2}{2w}, 0]$	$\frac{1}{2}F_{21}$	0	0	$R^2R^1n\pi$	$-R^2R^1n\pi$	0
0	$[\frac{\rho^3}{3w^2}, 0, 0]$	$-\frac{1}{3}(F_{12,1} + [F_{12}, A_1])$	0	0	0	$2R^1R^2R^1n\pi$	$-R^1R^2R^1n\pi$
0	$[\frac{-\rho^3}{3w^2}, 0, 0]$	$-\frac{1}{3}(F_{12,2} + [F_{12}, A_2])$	0	0	$-R^1R^2R^2n\pi$	0	0
0	$[\frac{-\rho^3}{3w^2}, 0, 0]$	$-\frac{1}{3}(F_{21,1} + [F_{21}, A_1])$	0	0	0	$-R^2R^1R^1n\pi$	0
0	$[\frac{\rho^3}{3w^2}, 0, 0]$	$-\frac{1}{3}(F_{21,2} + [F_{21}, A_2])$	0	0	$2R^2R^1R^2n\pi$	0	0

Table 1: *Gait table for a two input locomotion system with integrally related sinusoids as inputs and $T = 2\pi$.*

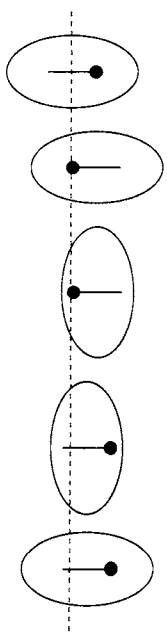


Figure 5: *Example of locomotion of a deformable body in inviscid, irrotational, incompressible fluid.*

The motion depicted in the figure consists of four discrete segments, starting from the top of the diagram. The vehicle starts with a horizontal shape and slides the internal mass from right to left, causing the vehicle body to shift in the opposite direction. The vehicle then changes to a vertical ellipse, leaving the mass position unchanged. The mass is then shifted back to the right, resulting in motion of the body to the left, and, finally, the vehicle is returned to its original shape.

Because of the presence of a fluid, the distance that the vehicle moves in response to the shifting mass depends on the shape of the vehicle. In particular, the vehicle shifts more when its major axis is aligned with the axis of the mass (the horizontal configuration) than when the major axis is perpindicular to the direction of motion of the mass. This is because the amount of fluid that most be moved is greater when the vehicle is in the vertical configuration.

To cast this system into the locomotion framework of the previous sections, we give a brief description of the dynamics for mechanical systems with symmetry. A

more detailed description of this material can be found in [9] and the references therein.

As before, assume we have a mechanical system with configuration space $Q = R \times G$ and write $q = (r, g)$ for the configuration and let $g^{-1}\dot{g})$ represent the body velocity. If the system has a (left) symmetry, then the Lagrangian can be written as a *reduced Lagrangian*

$$l(r, \dot{r}, \xi) = \frac{1}{2} \begin{bmatrix} \dot{r} \\ \xi \end{bmatrix}^T \begin{bmatrix} m(r) & A^T(r)I(r) \\ I(r)A(r) & I(r) \end{bmatrix} \begin{bmatrix} \dot{r} \\ \xi \end{bmatrix}. \tag{9}$$

The matrix $I(r) : \mathfrak{g} \to \mathfrak{g}^*$ is the *local locked inertia tensor* and $A(r) : TR \to \mathfrak{g}$ is the *local connection one form* for the mechanical connection. Note that g does not appear explicitly in this formula.

If we define the body momentum as

$$p = \frac{\partial l}{\partial \xi}$$

then the equations of motion can be written as

$$\begin{aligned} \dot{g} &= g(-A(r)\dot{r} + I^{-1}(r)p) \\ \frac{d}{dt}\langle p, \eta \rangle &= \langle p, [-A(r)\dot{r} + I^{-1}(r)p, \eta] \rangle \\ \tilde{M}(r)\ddot{r} + \tilde{C}(r, \dot{r})\dot{r} + \tilde{N}(r, \dot{r}, p) &= F \end{aligned} \tag{10}$$

where

$$\begin{aligned} \tilde{M} &= m - A^T I A \\ \tilde{C}_{ij} &= \frac{1}{2}\Big(\frac{\partial \tilde{M}_{ij}}{\partial r^k} + \frac{\partial \tilde{M}_{ik}}{\partial r^j} - \frac{\partial \tilde{M}_{kj}}{\partial r^i}\Big)\dot{r}^k \\ \langle \tilde{N}, \delta r \rangle &= -\langle p, dA(\dot{r}, \delta r) - [A(\dot{r}), A(\delta r)] + \\ &\qquad \tfrac{1}{2}\tfrac{\partial (I^{-1}p)}{\partial r}(\delta r) + [I^{-1}p, A(\delta r)] \rangle \end{aligned} \tag{11}$$

and $\eta \in \mathfrak{g}$ is a free variable. Equation (10) is a splitting of the equations into a group component, a momentum equation, and the reduced base dynamics. This particular form of the equation, using body angular momentum, shows the reduction of the dynamics explicitly: the group variables only appear in the top equation and all coupling of the base dynamics with the group dynamics is captured by the momentum, the locked inertia tensor, and the mechanical connection.

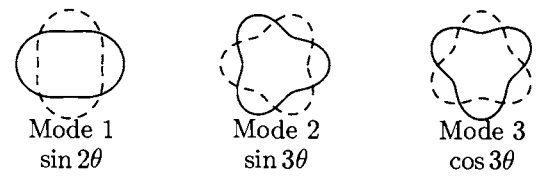


Figure 6: *Deformations for an underwater vehicle.*

Similar to our previous kinematic results, the motion of the full system can be retrieved from the trajectory of the base variables by integrating the momentum equation and the group equation. In particular, the group equation has two terms, one linear in $\dot{r}$ and one linear in p. The term which is linear in $\dot{r}$ determines the *geometric phase* or *holonomy* associated with the path in base space. This path is independent of the time-parameterization of the base space path. The term linear in p generates the so-called *dynamic phase* and determines the motion of the system when $\dot{r} = 0$.

In the specific case where $p = 0$ and there are no external forces that affect the momentum (e.g., if we only control the shape) then $p(t) \equiv 0$ and we are left we a group equation which is exactly in the same form as our previous locomotion systems:

$$\dot{g} = g(-A(r)\dot{r})$$

where $A(r)$ is now the *mechanical* connection resulting from system symmetries.

To apply this derivation of the mechanical connection to a deformable rigid body in a perfect fluid, we make use of a formulation due to Kirchhoff (see Lamb [6]) and model the interaction of the fluid with the deformable body using the assumption of potential flow. In order to carry out the potential flow computations, it is necessary to assume a class of deformations of the objectFigure refsquirm.fig shows one set of choices. Note that we do not make explicit use of a shifting mass. Instead, the second and third modes results in a shift of the center of mass of the deformed object relative to a fixed geometric center; this plays the role of the shifting mass. Concentrating on only the first and second modes, the deformation in the radius

has the form

$$r(t,\theta) = 1 + \epsilon\big(k_1(t)\cos 2\theta + k_2(t)\cos 3\theta\big),$$

where k_1 and k_2 can be regarded as the shape variables for this system.

Carrying out the computations to determine the (approximate) potential functions for this class of deformations and determining the resulting kinetic energy is straightforward, though tedious. Restricting our attention to motion in a single direction, x, and computing the potential flow by using a series expansion in ϵ, it can be shown that the Lagrangian for this system is given by

$$L = \frac{1}{2}\pi\dot{x}^2 - \epsilon\pi k_1\dot{x}^2 + \epsilon^2\pi\left(\tfrac{\dot{k}_1^2}{4} + \tfrac{\dot{k}_2^2}{6} + \dot{x}(k_1\dot{k}_2 - k_2\dot{k}_1) + \tfrac{\dot{x}^2}{4}(5k_1^2 + 9k_2^2)\right) + O(\epsilon^3)$$

(see [8] or [1] for details). Here the group variable is given by $g = x$ and the shape variables are $r = (k_1, k_2)$. The group action is simple translation and the body velocity is simply $\xi = \dot{x}$ (due to the fact that we are not considering rotation).

It is clear from the form of L that it can be written in the form $l(r, \dot{r}, x)$ and hence we can apply the analysis above to compute the mechanical connection:

$$\dot{x} = -A(k_1, k_2)\begin{bmatrix}\dot{k}_1\\ \dot{k}_2\end{bmatrix} = \epsilon^2(k_2\dot{k}_1 - k_1\dot{k}_2) + O(\epsilon^3).$$

Given this connection, we can proceed to analyze the controllability and gait structure. These computations are given in Table 1 and show that the mechanism can be made to translate by using sinusoidal inputs (in the two different modes). This is the direct analogy of the locomotion mechanism depicted in Figure 5, but using pure shape modulation to achive locomotion. An example of a more complicated gait, using all three modes simultaneously, is shown in Figure 7.

6 Summary and Conclusions

Many interesting mechanical systems do not have the special structure of locomotion systems. Some of these, such as the snakeboard [7] rely on more complicated locomotion mechanisms that cannot be modeled as a simple connection on a principal bundle. However, they are still systems defined on principal bundles and hence holonomy is still an essential concept in understanding the system. Many of the geometric tools and concepts presented here can be extended those more general cases (see, for example, the thesis work of Ostrowski [11]).

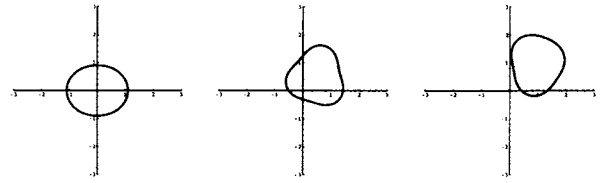


Figure 7: *The 1-1-1 gait for the inviscid amoeba:* $k = (\cos t, -\sin t, -\sin t)$.

More general mechanical systems may not have the underlying structure of a principle bundle or the controls acting on the system may not directly actuate the shape of the system. One simple example of such a system is a planar rigid body controlled by forces attached to the body. Here the controls act directly on the group variables (by changing the momentum of the object) and the geometry used here is less appropriate. Nonetheless, these systems still posses Lagrangian structure and hence it is possible to derive a number of specialized results that exploit this structure. For a recent survey, see [9].

Acknowledgements

This work has been funded in part by the National Science Foundation and the Office of Naval Research. The authors would like to thank Jerrold Marsden, Richard Mason, Helena Andreas, and Francesco Bullo for discussion and comments on this work.

References

[1] H. Andreas. Underwater locomotion using swimming cylinders. Master's thesis, Lund Institute of Technology, 1998.

[2] S. D. Kelly, R. J. Mason, C. T. Anhalt, R. M. Murray, and J. W. Burdick. Modelling and experimental investigation of carangiform locomotion for control. In *Proc. American Control Conference*, 1998. To Appear.

[3] S. D. Kelly and R. M. Murray. Geometric phases and robotic locomotion. *Journal of Robotic Systems*, 12(6):417–431, 1995.

[4] S. D. Kelly and R. M. Murray. The geometry and control of dissipative systems. In *Proc. IEEE Control and Decision Conference*, pages 981–986, 1996.

[5] S. D. Kelly and R. M. Murray. Lagrangian mechanics and carangiform locomotion. In *IFAC Symposium on Nonlinear Control Systems Design (NOLCOS)*, 1998. To Appear.

[6] Sir Horace Lamb. *Hydrodynamics*. Dover Publications, 1945.

[7] A. Lewis, J. Ostrowski, R. M. Murray, and J. W. Burdick. Nonholonomic mechanics and locomotion: The snakeboard example. In *Proc. IEEE International Conference on Robotics and Automation*, 1994.

[8] R. Mason. Notes on a planar amoeba swimming in inviscid fluid. Unpublished notes, 1997.

[9] R. M. Murray. Nonlinear control of mechanical systems: A Lagrangian perspective. *Annual Reviews in Control*, 21:31–45, 1997.

[10] R. M. Murray, Z. Li, and S. S. Sastry. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, 1994.

[11] J. P. Ostrowski. *The Mechanics and Control of Undulatory Robotic Locomotion*. PhD thesis, California Institute of Technology, 1995.

[12] J. P. Ostrowski and J. W. Burdick. Control of mechanical systems with symmetries and nonholonomic constraints. In *Proc. IEEE Control and Decision Conference*, 1995.

[13] J. E. Radford and J. W. Burdick. Local motion planning for nonholonomic controls systems evolving on principal bundles. In *Mathematical Theory of Networks and Systems*, 1998. Submitted.

[14] P. G. Saffman. The self-propulsion of a deformable body in a perfect fluid. *Journal of Fluid Mechanics*, 28, 1967.

Mechanics, Planning, and Control for Tapping

Wesley H. Huang, *Carnegie Mellon University, Pittsburgh, PA, USA*
Matthew T. Mason, *Carnegie Mellon University, Pittsburgh, PA, USA*

We have been studying manipulation using impulsive forces; in particular, we have focused on the manipulation of planar sliders by tapping. These objects then slide on a support surface, slowing down and coming to rest due to frictional forces. In this paper, we present analytical and experimental results in the mechanics, planning, and control of this system.

1 Introduction

We consider *impulsive manipulation* to be any form of manipulation consisting of two phases: a strike to the object imparting some initial velocities and free motion of that object subject to forces and constraints in the environment. In this work, we have focused on manipulation of planar sliders by tapping. After being struck, the object slides on the support surface, slowing down and coming to rest due to frictional forces.

Using impulsive forces for manipulation has several unique characteristics as compared to other modes of manipulation. Tapping is well suited for micropositioning — the impact instantaneously breaks static friction, but the amount of impulse delivered determines how far the object moves. Tapping results in quick dynamic motion, requires a relatively simple actuator, and can manipulate a wide variety of objects due to its nonprehensile (nongrasping) nature.

There are many ways tapping could be instantiated. It could be used for fast and relatively low precision parts transfer between workcells. Our analysis might be applied to the design and configuration of certain types of vibratory parts feeding systems. A "positioning cell" could be built using a number of tapping actuators placed around its perimeter; an object placed in the cell could be precisely positioned by firing an appropriate sequence of actuators. The setup we have used for our experiments features a general purpose robotic arm to position a tapping device with respect to the object; an overhead camera measures the position and orientation of the object.

1.1 Overview

We first consider how to plan a single tap to achieve a desired displacement (translation and rotation). Because the manipulator interacts with the object only at the instant of impact, we can consider separately the mechanics of a sliding object and the mechanics of striking that object to produce some initial velocities. The dynamics of a sliding rotating object give rise to a strongly coupled system for which there is no general analytic solution, so we have developed a numerical technique to determine the required initial velocities for a desired object translation and rotation. Determining how to generate those velocities by impact has been a relatively straightforward application of the classical impact models. In general, we must search the boundary for an appropriate contact point; simple shapes may have analytic solutions for the striker parameters required to generate the initial velocities.

We performed a series of single-tap experiments to validate the models used in our analysis. To our surprise, we found that objects did not rotate as much as our model predicted. Examination of the rotational velocity profiles revealed that there was more frictional torque than predicted, so we added an additional parameter to our model to scale the torque due to friction. Although this discrepancy is in part due to differences between our idealized model assumptions and the actual experimental setup, we believe there may be some additional mechanism giving rise to this increased fric-

tional torque. We are still able to use our original solution to the mechanics of a sliding object with this modified model.

A single tap cannot arbitrarily place an object in the plane; multiple taps may be required. We first characterize the set of states that can be reached with a single tap and then formulate a basic planning method which shows that any configuration can be reached with at most two taps. Given the large variations in single-taps, however, a two-tap plan may not be desirable. We extend this basic planning method to several specific planning methods which essentially serve as feedback control laws. Each planning method has an implicit model of the error from a single tap and makes some trade off between the expected number of taps required to position an object and the variation in the number of taps. The performance of these planning methods was demonstrated in several positioning experiments.

The final result of this paper is an experimental proof of a conjecture we made early in the course of this work: that tapping can be used to position an object more precisely than the manipulator can position the tapping device. Through a basic sensitivity analysis, we show that there are a wide range of taps for which the error in object configuration is smaller than the error in tapping device configuration.

1.2 Related work

The first appearance of impulsive manipulation in the robotics literature was due to Higuchi [10] who used an electromagnetic coil to deliver an impulsive force for linear micropositioning. Since then, Yamagata and Higuchi (e.g. [22]) have developed related devices for several different micropositioning applications.

The present work is the first to show formal results in the mechanics, planning, and control of tapping. Our analysis of the mechanics of a sliding object is built upon the work of Voyenli and Eriksen [20] who studied the motion of sliding disks and rings and the work of Goyal *et al.*([8] and [9]) who formulated a limit surface representation of the relationship between net frictional force and torque and the the velocities of a planar slider.

Our work makes use of impact analysis due to Routh, more recently presented in Wang and Mason [21]. This work presents a graphical method of using Newton's and Poisson's laws of restitution for two dimensional impact with friction. Although there has been much recent work on impact (e.g. Bowden and Tabor [1], Brach [2], Stronge [17], and Chatterjee and Ruina [6]), the simplicity of Poisson's hypothesis makes our analysis simpler; it seems to work reasonably well in practice.

This work has been inspired by the ideas behind minimalist robotics, articulated by Canny and Goldberg ([4] and [5]) and by work in many areas of non-prehensile manipulation, such as pushing (Mason [14] and many others), palmar manipulation (Trinkle *et al.*([19] and [18]), Erdmann [7], and Zumel [24]), and paddle juggling (Bühler and Koditschek [3], Rizzi and Koditschek [15], and Schaal and Atkeson [16]).

Experiments with planar sliders have been performed by Zhu *et al.* [23] in conjunction with what they call "releasing manipulation." In their experiments, objects were accelerated using a manipulator (instead of striking them) and then released, allowing them to slide on a support surface and come to rest due to friction. Their results, like ours, show considerable variation, presumably resulting from variations in friction.

The results presented in this paper appear in greater detail in Huang [11].

1.3 Assumptions

We assume that the support surface is homogeneous. The object is a planar lamina of mass M and moment of inertia I, and the support distribution between the object and the support surface is known. For this paper, we assume a uniform support distribution, though our methods can work with more general support distributions. Coulomb friction acts between the object and the support surface with a coefficient of friction μ.

We use Poisson's hypothesis in conjunction with the classical model of two dimensional impact with friction. This implies that the impact is instantaneous and is characterized by: the geometry and relative velocities of the object and the striker, a coefficient of restitution, and a coefficient of friction.

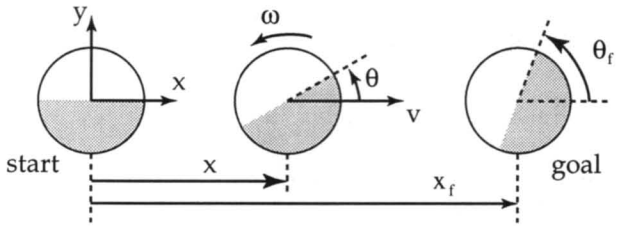


Figure 1: *Notation for the axisymmetric inverse sliding problem. The goal configuration is at a distance x_f along the $\hat{x}$ axis and a rotation of θ_f. The object's position and orientation during the sliding motion are x and θ, its velocities v and ω.*

2 Mechanics

In this section, we show how to plan a single-tap. This can be decomposed into two subproblems: the *inverse sliding problem* and the *impact problem*. The inverse sliding problem is to determine the initial velocities required for the object to slide the desired displacement (translation and rotation). The impact problem is to determine how (if possible) to generate those velocities by striking the object.

2.1 Inverse sliding problem

We address the inverse sliding problem separately for the classes of axisymmetric and nonaxisymmetric objects. Axisymmetric objects are those which have a pressure distribution that is a function of radius only; these objects have the property that they will always slide in a straight line. This is not the case for nonaxisymmetric objects, which we will address at the end of this subsection.

Since axisymmetric objects slide in a straight line, we only need consider the translation along the line from the start to the goal instead of translation in two dimensions. The problem is then:

> Determine the initial velocities v_0 and ω_0 such that the object comes to rest after sliding a desired distance d and angle α.

See Figure 1 for notation.

The frictional force load along the $\hat{x}$ axis is

$$F(\frac{\omega}{v}) = \mu g \lambda \int\int \frac{1 - \frac{\omega}{v}\sin\theta}{\sqrt{1 - 2\frac{\omega}{v} r \sin\theta + (\frac{\omega}{v})^2 r^2}} r dr d\theta \quad (1)$$

where λ is the pressure over the support distribution and the double integral is over this support distribution. The net torque load due to friction is:

$$T(\frac{\omega}{v}) = \mu g \lambda \int\int \frac{\frac{\omega}{v} r^2 - r\sin\theta}{\sqrt{1 - 2\frac{\omega}{v} r \sin\theta + (\frac{\omega}{v})^2 r^2}} r dr d\theta \quad (2)$$

Note that the force and torque are functions of the velocity ratio $\frac{\omega}{v}$, not the velocities themselves. Furthermore Mason [14] showed that as a function of $\frac{\omega}{v}$, the force is monotonic decreasing, the torque monotonic increasing.

The equations of motion of this system are:

$$M\dot{v} = -F(\frac{\omega}{v}) \quad (3)$$

$$I\dot{\omega} = -T(\frac{\omega}{v}) \quad (4)$$

This is a pair of strongly coupled first order differential equations. The net translation and rotation of the object are given by:

$$x_f(v_0, \omega_0) = \int_0^{t_f} v(t)\, dt \quad (5)$$

$$\theta_f(v_0, \omega_0) = \int_0^{t_f} \omega(t)\, dt \quad (6)$$

where t_f is the time at which the object comes to rest and $v(t)$ and $\omega(t)$ are solutions to the equations of motion, subject to the initial conditions:

$$v(0) = v_0 \qquad \omega(0) = \omega_0 \quad (7)$$

We consider x_f and θ_f to be functions over the space of initial velocities v_0 and ω_0.

In general, there are no analytic expressions for the net force and torque due to friction, so there will be no analytic expression for x_f and θ_f. However, we have been able to prove a number of properties about this system. We state the following without proof; the interested reader can consult Huang [11] or ([12] and [13]) for details.

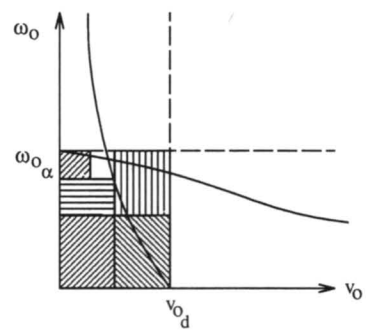


Figure 2: *Illustration of level curves. The level curve of x_f corresponding to a distance d asymptotically approaches the ω_0 axis and must lie to the left of its v_0 intercept. Likewise, the level curve of θ_f corresponding to a rotation α asymptotically approaches the v_0 axis and must lie below its ω_0 intercept.*

Property 1 *The coupling between translation and rotation disappears for $v_0 = 0$ or $\omega_0 = 0$, so we can formulate analytic expressions for x_f and θ_f along these axes.*

Property 2 *The functions $x_f(v_0, \omega_0)$ and $\theta_f(v_0, \omega_0)$ are monotonic in the sense that:*

$$v_{0_1} > v_{0_2} \quad \omega_{0_1} = \omega_{0_2} \quad \Longrightarrow x_{f_1} > x_{f_2} \quad (8)$$

$$v_{0_1} > v_{0_2} \quad \omega_{0_1} = \omega_{0_2} \neq 0 \quad \Longrightarrow \theta_{f_1} > \theta_{f_2} \quad (9)$$

$$v_{0_1} = v_{0_2} \neq 0 \quad \omega_{0_1} > \omega_{0_2} \quad \Longrightarrow x_{f_1} > x_{f_2} \quad (10)$$

$$v_{0_1} = v_{0_2} \quad \omega_{0_1} > \omega_{0_2} \quad \Longrightarrow \theta_{f_1} > \theta_{f_2} \quad (11)$$

where x_{f_i} and θ_{f_i} is the displacement resulting from the initial velocities v_{0_i} and ω_{0_i}.

Property 3 *The level curves of x_f and θ_f are strictly monotonic decreasing in the space of initial velocities.*

Property 4 *The value of x_f is strictly monotonic increasing along level curves of θ_f and vice versa.*

These properties imply that the level curve of x_f for a distance d and the level curve of θ_f for a rotation α must intersect exactly once and that this intersection point can be bounded. (See Figure 2 for an illustration.) The coordinates of this intersection point correspond to the initial velocities required to achieve the desired displacement.

These properties also lead us to a method for numerically finding the coordinates of this intersection point:

- Subdivide the bounding rectangle.
- Numerically integrate the initial conditions $(\bar{v_0}, \bar{\omega_0})$ corresponding to a corner of a subrectangle.
- If the distance or angle is greater than the desired distance or angle, then the region of the bounding rectangle where $v_0 > \bar{v_0}$ and $\omega_0 > \bar{\omega_0}$ can be eliminated from consideration.
- If the distance or angle is less than the desired distance or angle, then the region of the bounding rectangle where $v_0 < \bar{v_0}$ and $\omega_0 < \bar{\omega_0}$ can be eliminated from consideration.

In this manner, we can "zero in" on the coordinates of the intersection point.

2.1.1 Nonaxisymmetric objects

Nonaxisymmetric objects will in general have some component of the net frictional force that is normal to the net translational velocity, so these objects will not slide in a straight line. The inverse sliding problem is then difficult to solve because the monotonicity properties no longer hold and because the problem is higher dimensional.

In the simulations we have done with nonaxisymmetric objects, the movement of the object in the direction normal to the translational velocity has been very small, suggesting one possible solution to this problem — simply ignore this normal motion, essentially treating the object as axisymmetric. Given the large variations in actual experiments, this appears to be a very reasonable strategy. Errors due to this modeling simplification can be dealt with through feedback control.

The alternative is to employ a more general numerical root finding method; however, we have not been able to prove the existence or the uniqueness of a solution for the general nonaxisymmetric case.

2.2 Impact problem

Once we know what initial velocities are required, we must determine how, if possible, to generate those velocities via impact. We assume that to generate impact, we will essentially throw a free mass (the striker) at the object. We take as the starting point of our analysis the fact that any impulse within a friction cone at the contact point can be generated. (See Huang [11] for details.)

The friction cone at the impact point constrains only the direction of the impulse. The resulting velocities will scale with the magnitude of the impulse, so we need only consider the set of velocity ratios $\frac{\omega_0}{|\vec{v}_0|}$ and the direction of the translational velocity.

For a given contact point, we can produce a range of velocity ratios, each with translational velocity in a different direction. However, the question we face in planning is whether we can produce a desired velocity ratio in a given direction. This will generally lead to searching the boundary to find a suitable contact point.

For simple shapes, we can write analytic forms for the range of possible velocity ratios as a function of translational velocity direction; it may also be possible to write an analytic function mapping the velocity direction and velocity ratio to the contact point.

Once we have determined the contact point, we must then go through the mechanics of impact to determine the exact striker parameters (velocity and angle of incidence) in order to produce the necessary impulse.

Note that the limitation on tapping is due to the impact process — for any desired translation and rotation, we can determine initial velocities for the object to slide the proper displacement, but we are limited in the velocity ratios that can be produced via impact. Although this is in part due to the shape of the object, a more fundamental reason is the limited amount of rotation per translation that can be produced — friction limits the impulse to lie within a cone about the contact normal.

2.3 Single-tap experiments

Our first experiments were single-tap experiments designed to validate the model used in analysis of the mechanics. There were significant deviations between the idealized model for the analysis and the actual setup that was used in the experiments. First and foremost, the objects do not have a uniform pressure distribution; secondly, the objects are three dimensional, not planar.

We tested the model of the mechanics using three different combinations of object, materials, and support surfaces. Initial experiments were not successful — objects did not rotate as much as our model predicted. We then performed more carefully controlled experiments and used a high speed measurement system (an OptoTrak system which measured the position and orientation of the object at 500 Hz).

These measurements allowed us to calculate initial velocities and velocity profiles of the object, which in turn allowed us to evaluate the impact model and the sliding model separately.

Although the predicted and actual translational velocity profiles matched reasonably well, the rotational velocity profiles did not. Judging by the measured and predicted velocity profiles (see Figure 3 for an example), there was more frictional torque than predicted.

We then added an additional parameter to the model to scale the net torque due to friction. For the three sets of experiments we did, the scale factors were 1.0, 2.17, and 3.64. However, with this modified model (which still allows us to use the same method for solving the inverse sliding problem) we were able to attain reasonable matches between predicted and actual velocity profiles.

In the end, we were able to predict object motion on average to within 5–10% for translations and to within 10–25% for rotations. The impact model generally predicts initial translational velocities to with a few percent and initial rotational velocities to within a few percent up to 33%.

One of the major results of our single-tap experiments is simply that real three dimensional objects (even those that aren't quite axisymmetric) can be modeled reasonably well by the axisymmetric mechanics.

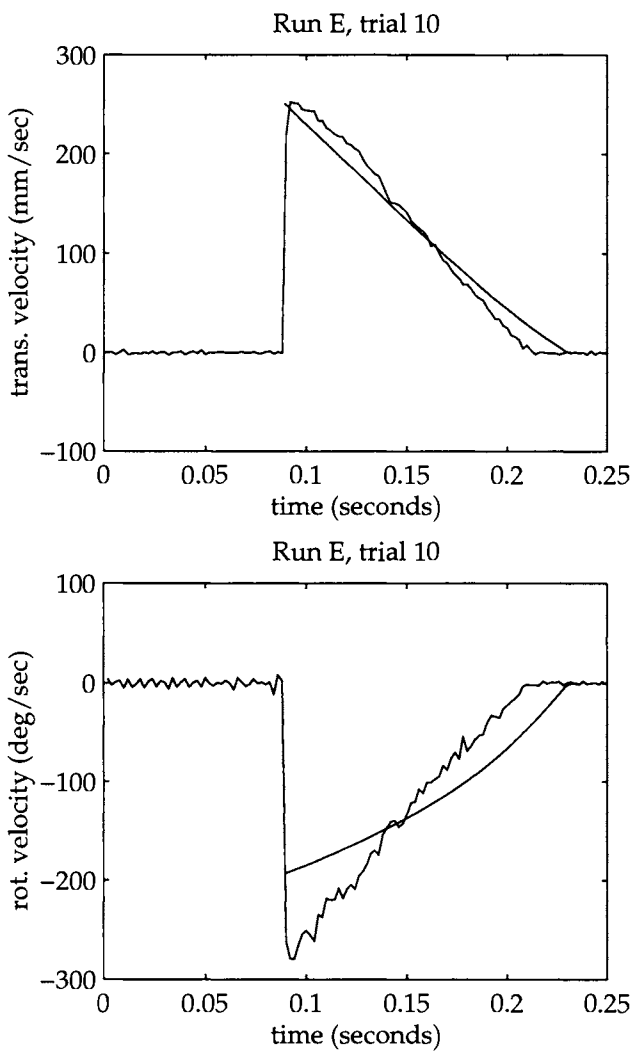


Figure 3: *Measured and calculated velocity profiles for the aluminum square experiment using the unscaled torque. Although the translational velocity matches well, the rotational velocity profile reveals that there is more frictional torque than predicted.*

3 Planning and Control

We know that a single tap will not be able to achieve any given displacement because of constraints imposed by impact. In addition, we must be able to deal with the large experimental variations that result from a single tap. We have found that for this problem, planning and control are intertwined — a basic planning method serves as the basis for feedback control.

Since the axisymmetric mechanics are more widely applicable that one would suspect (given the assumptions of the underlying model) we consider planning and feedback control using only these mechanics. Furthermore, we primarily consider manipulation of circular objects in the following subsections.

All our planning methods work in the configuration space of the object, specified by a position x and orientation θ.

3.1 Basic planning method

Before we can consider multiple taps, we must first characterize the displacements we can achieve with a single tap. Through straightforward use of the chain rule, we can show that if $v_0(t)$ and $\omega_0(t)$ are solutions to the equations of motion (Equations 3 and 4) then so are $kv_0(\frac{t}{k})$ and $k\omega_0(\frac{t}{k})$. If the former produces a displacement of x_f and θ_f, then the latter produces a displacement of k^2x_f and $k^2\theta_f$. Therefore, all initial velocities with the same velocity ratio $\frac{\omega_0}{|\vec{v}_0|}$ produce displacements with the same displacement ratio $\frac{x_f}{\theta_f}$.

For objects with smooth boundaries, we can produce a continuous range of velocity ratios in any given direction, which will result in a continuous range of displacement ratios. For circular objects, these ranges will be symmetric about zero and identical for all translational velocity directions. The range of displacement ratios can be viewed as a range of slopes in configuration space, so the set of states that can be reached in a single tap forms a cone symmetric about the x axis in configuration space; we refer to this cone as the reachable displacement cone.

The set of states then that can be reached in a single tap lie within this cone starting at the current state. The set of states that can reach the goal configuration in a single tap can be determined by reflecting this reachable displacement cone at the goal through the goal state. Since the reachable displacement cone is symmetric, this backprojection looks exactly the same.

It is easy to see (as illustrated in Figure 4) that by intersecting the reachable displacement cone from the

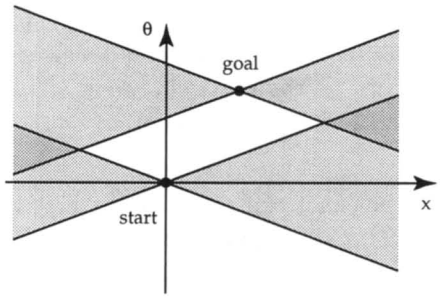


Figure 4: *Planning for axisymmetric objects. The reachable displacement cones for each direction are placed at the start configuration, the backprojections at the goal configuration. (For circular axisymmetric objects, these cones are identical.) The intersection is the set of all points that may be used as an intermediate configuration in getting from the start to the goal in two taps.*

start configuration with the backprojection of the goal, we can reach any state within two taps. Furthermore, there will be a two dimensional set of states which can be used as a subgoal to get from the start to the goal in two taps.

3.2 Extensions of the basic planning method for feedback control

The basic planning method does not tell us how to choose a subgoal when two taps are required. In addition, because of tapping device limitations and errors in modeling and actuation, a two tap plan is not necessarily the best plan. We have examined three different schemes based on the basic planning method; these schemes lie somewhere between traditional planners and feedback control laws.

3.2.1 Exact two-tap plans

The simplest way to implement the basic planning method is simply to choose the closest feasible subgoal. We implement this plan as follows: if the goal can be reached in a single tap, then we execute that tap, otherwise, we execute the first tap of an exact two-tap plan to reach the goal from the current configuration.

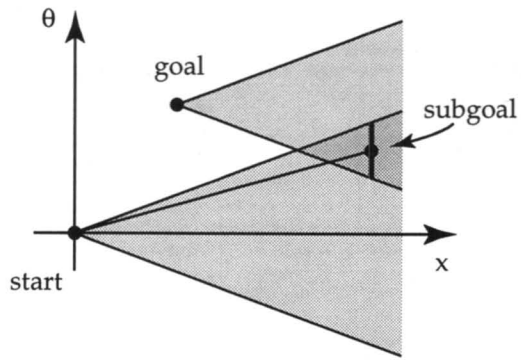


Figure 5: *The conservative two-tap planning method chooses a subgoal such that the (degenerate) error ellipse fits entirely within the space of feasible subgoals.*

3.2.2 Conservative two-tap plans

A further refinement we can make is to consider explicitly a model of the error in a single tap. In preliminary experiments, we observed that errors in rotation tended to be much larger (or more important) than errors in translation, so we use the degenerate case of the traditional error ellipse: an error "line" parallel to the θ axis in configuration space.

A potential problem with the exact two-tap plans is that if there is any error in the first tap, the object may not be in a state that can reach the goal configuration in one more tap. So in planning conservative two-tap plans, we select a subgoal so that the entire error ellipse for the first tap fits within the region of allowable subgoals. A conservative two-tap plan is illustrated in Figure 5.

Like the exact two-tap plans, this planning method is executed as follows: if the goal can be reached in one tap, then we execute that tap, otherwise, the first tap of a conservative two-tap plan from the current state to the goal is executed.

3.2.3 Multi-tap plans

One effect of using the conservative two-tap planning strategy is that the excursion in going from the start to the goal increases. Although we plan for the error ellipse from the first tap to lie entirely within a backprojection from the goal, we have not explicitly

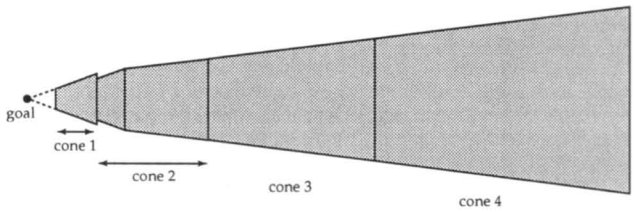


Figure 6: *The first four cones from the goal to the right for the multi-tap planning method.*

considered the error ellipse from the second tap. The larger the second tap is, the larger the error ellipse about the goal will be.

Another approach, then, is to consider planning backwards from the goal. We first consider a final tap small enough to have an acceptable error about the goal. There will be a small "cone" of states that can reach the goal with a sufficiently small tap. Then we consider the set of states that can reach this first cone such that the error ellipse from that tap lies entirely within the first cone. In this manner we construct a series of cones in configuration space (illustrated in Figure 6) extending backwards from the goal. We then need to plan a tap that takes us from the start configuration into one of these cones such that the error ellipse from that tap lies completely within the sequence of cones.

This planning method is executed as follows: for every tap, we determine whether the current state is within this sequence of cones. If so, then we execute a tap to take us to the center of the next cone. Otherwise, we execute a tap to take us into the closest cone.

3.3 Positioning experiments

We performed experiments using all three planning methods. The experimental setup used an aluminum disk sliding on a formica surface. An Adept 550 robot was used to position a tapping device with respect to the object, and an overhead camera measured the position and orientation of the object.

The task for all the positioning experiments was a displacement of 10 mm translation and 10 degrees rotation. This is about three times as much rotation as can be achieved in a single 10 mm tap. The desired accuracy was to within 1 mm and 1 degree; this was limited in part by the dynamic range of the tapping device and in part by the resolution of the overhead camera.

Overall, the exact two-tap planning method worked the best, taking an average of 4–5 taps. The best run took exactly two taps as illustrated in Figure 7; a typical run is shown in Figure 8. The conservative two-tap plans were only slightly worse, taking an average of 5–6 taps; a typical run is shown in Figure 9. The multi-tap plans, with one exception, were executed exactly as planned, i.e. they were supposed to take and did take exactly 5 taps to reach the goal. A typical run is shown in Figure 10.

There appears to be a tradeoff among these planning methods in the average number of taps required to reach the goal and the variance in the number of taps. The exact two tap plans have the lowest average number of taps but the highest variance. The multi-tap plans have very little variance but a higher average. We hope to more formally characterize this tradeoff in future work.

3.4 High-precision positioning

Early in the course of this work, we conjectured that by tapping, an object could be positioned more precisely than the manipulator could position the tapping device with respect to the object. In order to test this idea, we performed a number of experiments using the exact two-tap planning method in which the position and orientation of the tapping device requested by the planning method was rounded to the nearest 15 mm in position and 10 degrees in orientation (with respect to the world coordinates). The task was again to displace the object by 10 mm and 10 degrees to an accuracy of 1 mm and 1 degree. In all trials, the task was successfully completed; a typical trial took 5–6 taps.

3.4.1 Sensitivity analysis

In order to characterize this robustness for high-precision positioning, we did a basic sensitivity analysis by examining the Jacobian relating changes in the position and orientation of the tapping device to the

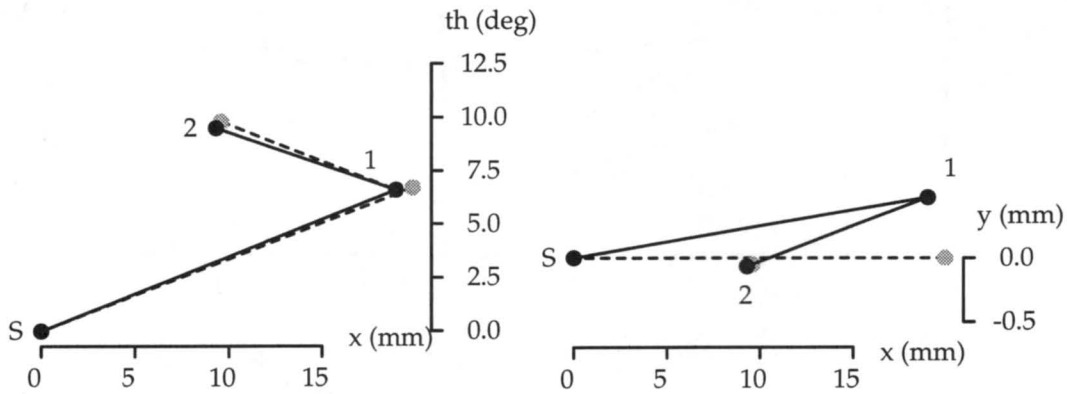


Figure 7: *A nearly ideal trial using the exact two-tap planning method. This Figure (and Figures 8–10) shows a positioning trial in configuration space on the left and in cartesian space on the right. The dotted lines and grey dots are the planned taps; the solid lines and solid dots are the actual taps.*

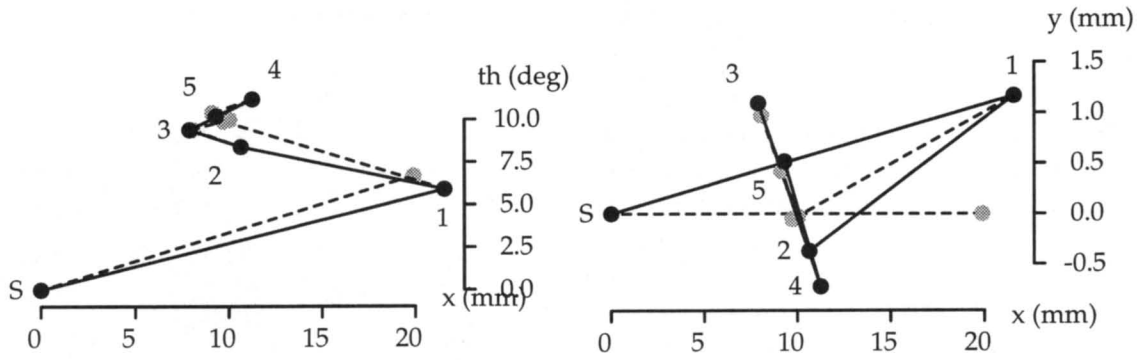


Figure 8: *Example of a typical trial using the exact two-tap planning method.*

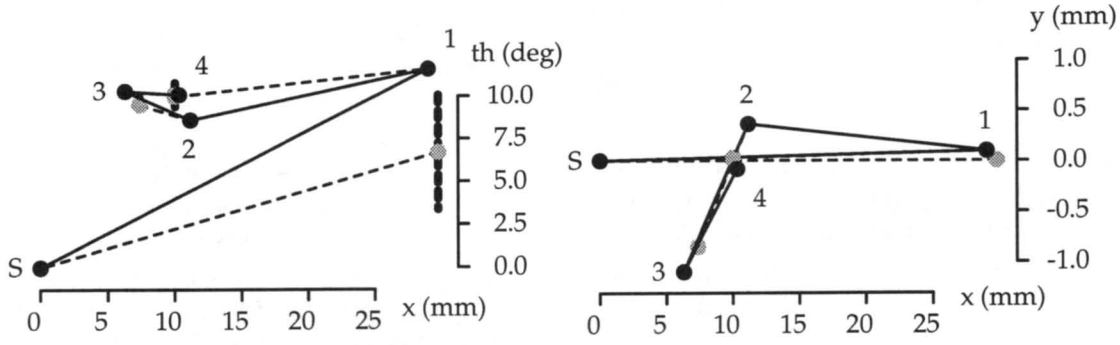


Figure 9: *Example of a typical trial using the conservative two-tap planning method.*

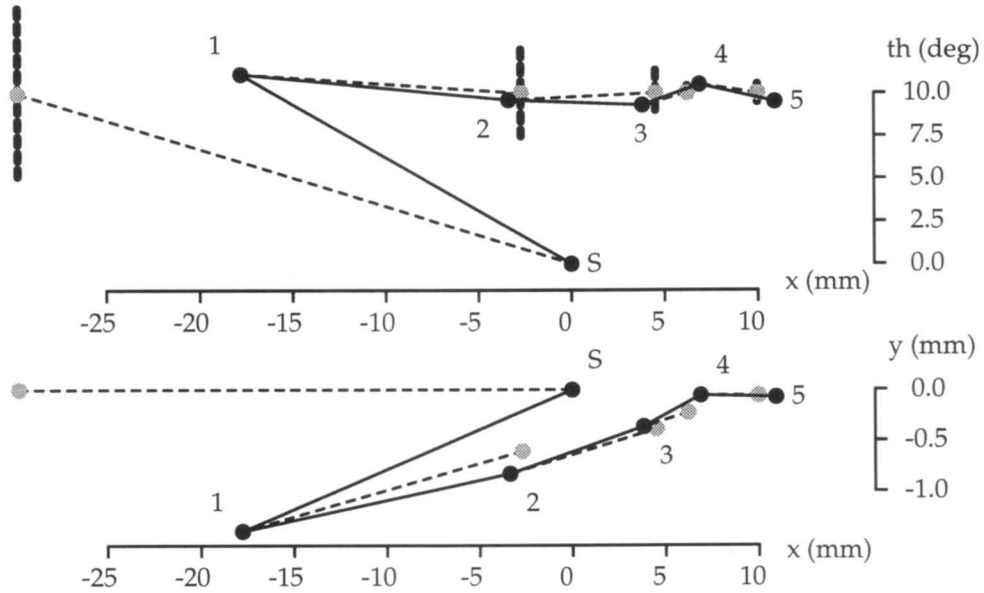


Figure 10: *Example of a typical trial using the multi-tap planning method.*

resulting change in the position and orientation of the object after a tap. All but one component of the Jacobian can be computed analytically; that component relates the change in initial object velocities to the changes in displacement (recall that there is no analytic form for the displacement functions x_f and θ_f). Since this component of the Jacobian must be evaluated numerically, we must look at variation with respect to a particular tap. For a 10 mm 3 degree tap (using the same object and materials as the high-precision positioning experiment), the maximum striker error vector $[x\ y\ \theta]^T = [7.5\ 7.5\ 5]^T$ (corresponding to the maximum error due to rounding in this experiment) is mapped to an object error of $[1.1\ 0.9\ 0.2]^T$. Here, the error is smaller in each dimension.

Furthermore, we have found that there is a broad range of taps for which the norm of the Jacobian is less than one and thus the error in object configuration is always smaller than that of the tapping device configuration. This is shown graphically in Figure 11.

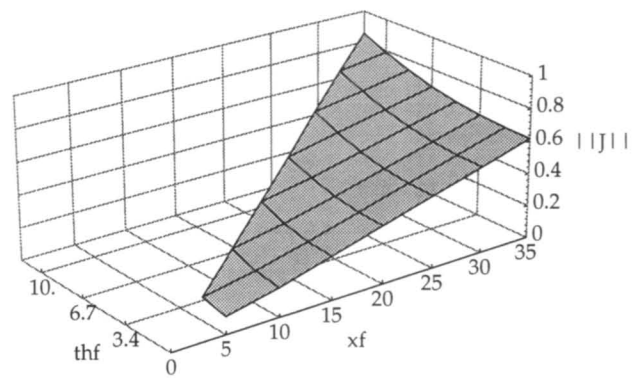


Figure 11: *Norm of the final configuration Jacobian as a function of displacement. x_f is in mm, θ_f in degrees.*

4 Conclusions

We have shown a framework for manipulation by tapping, one particular form of impulsive manipulation.

Through analysis of the mechanics of a planar slider, we have demonstrated a solution to the single-tap planning problem. We performed single-tap experiments to validate the models used in our analysis and found that there was more frictional torque acting upon the object than our model predicted. An additional parameter in our model scales the frictional torque to bring our model and experimental observations into agreement. Although this deviation from the original model

is in part due to differences between the physical experimental apparatus and the idealized model assumptions, primarily uniform support distribution and three dimensional extent, we believe there may be some additional mechanism that results in this increased torque.

We have formulated a basic planning method which shows that it is possible to position an axisymmetric object anywhere in the plane within two taps. This basic planning method was then adapted to create practical planning methods which essentially serve as feedback control laws; these planning methods specify how to choose intermediate state(s) in order to reach the goal in the presence of variations due to modeling and actuation errors. We demonstrated three different planning methods which display a tradeoff between the average number of taps required to complete a task and the variation in the number of taps.

Finally, we have demonstrated that by tapping it is possible to position an object more precisely than the manipulator positions the tapping device with respect to the object. We have presented a basic stability analysis in support of this result.

Acknowledgements

We would like to thank Jason Powell for building and helping design the tapping devices and the Medical Robotics and Computer Assisted Surgery lab at Carnegie Mellon for use of their OptoTrak system.

This work was supported in part by NASA through the Graduate Student Researchers Program and through grant NAGW 1175 and by the NSF under grants IRI–9318496 and IRI-9114208.

References

[1] F. P. Bowden and D. Tabor. *The Friction and Lubrication of Solids: Part II.* Oxford University Press, 1964.

[2] R. M. Brach. *Mechanical Impact Dynamics: Rigid Body Collisions.* John Wiley & Sons, 1991.

[3] M. Bühler and D. E. Koditschek. From stable to chaotic juggling: Theory, simulation, and experiments. In *IEEE International Conference on Robotics and Automation*, pages 1976–1981, Cincinnati, OH, 1990.

[4] J. Canny and K. Goldberg. A RISC approach to robotics. *IEEE Robotics & Automation Magazine*, 1(1):26–8, March 1994.

[5] J. F. Canny and K. Y. Goldberg. "RISC" industrial robotics: Recent results and open problems. In *IEEE International Conference on Robotics and Automation*, pages 1951–8, 1994.

[6] A. Chatterjee and A. L. Ruina. A new algebraic rigid body collision model with some useful properties. *Journal of Applied Mechanics*, July 1997. Submitted.

[7] M. A. Erdmann. An exploration of nonprehensile two-palm manipulation: Planning and execution. In G. Giralt and G. Hirzinger, editors, *International Symposium on Robotics Research: The Seventh International Symposium.* Springer-Verlag, 1996.

[8] S. Goyal, A. Ruina, and J. Papadopoulos. Planar sliding with dry friction. Part 1. Limit surface and moment function. *Wear*, 143:307–330, 1991.

[9] S. Goyal, A. Ruina, and J. Papadopoulos. Planar sliding with dry friction. Part 2. Dynamics of motion. *Wear*, 143:331–352, 1991.

[10] T. Higuchi. Application of electromagnetic impulsive force to precise positioning tools in robot systems. In O. Faugeras and G. Giralt, editors, *International Symposium on Robotics Research: The Third International Symposium.* MIT Press, 1986.

[11] W. H. Huang. *Impulsive Manipulation.* PhD thesis, Carnegie Mellon University, Aug. 1997. Available as Carnegie Mellon Robotics Institute technical report CMU–RI–TR–97–29.

[12] W. H. Huang, E. P. Krotkov, and M. T. Mason. Impulsive manipulation. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 120–5, 1995.

[13] W. H. Huang and M. T. Mason. Limiting cases of impulsive manipulation. Technical Report CMU–RI–TR–96–24, The Robotics Institute, Carnegie Mellon University, September 1996.

[14] M. T. Mason and J. K. Salisbury, Jr. *Robot Hands and the Mechanics of Manipulation.* The MIT Press, 1985.

[15] A. A. Rizzi and D. E. Koditschek. Progress in spatial robot juggling. In *IEEE International Conference on Robotics and Automation*, pages 775–780, Nice, France, 1992.

[16] S. Schaal and C. G. Atkeson. Open loop stable control strategies for robot juggling. In *IEEE International Conference on Robotics and Automation*, pages 3:913–918, Atlanta, GA, 1993.

[17] W. J. Stronge. Rigid body collisions with friction. *Proceedings of the Royal Society, London*, A431:169–181, 1990.

[18] J. C. Trinkle, A. O. Farahat, and P. F. Stiller. First-order stability cells of active multi-rigid-body systems. *IEEE Transactions on Robotics and Automation*, 11(4):545–557, Aug. 1995.

[19] J. C. Trinkle, R. C. Ram, A. O. Farahat, and P. F. Stiller. Dexterous manipulation planning and execution of an enveloped slippery workpiece. In *IEEE International Conference on Robotics and Automation*, pages 2: 442–448, 1993.

[20] K. Voyenli and E. Eriksen. On the motion of an ice hockey puck. *American Journal of Physics*, 53(12):1149–53, December 1985.

[21] Y. Wang and M. T. Mason. Two-dimensional rigid-body collisions with friction. *ASME Journal of Applied Mechanics*, 59:635–641, Sept. 1992.

[22] Y. Yamagata and T. Higuchi. A micropositioning device for precision automatic assembly using impact force of piezoelectric elements. In *IEEE International Conference on Robotics and Automation*, 1995.

[23] C. Zhu, Y. Aiyama, T. Chawanya, and T. Arai. Releasing manipulation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 2, page 1, Osaka, Japan, Nov. 1996.

[24] N. B. Zumel. *A Nonprehensile Method for Reliable Parts Orienting*. PhD thesis, Carnegie Mellon University, Jan. 1997.

Planning for In-Hand Dextrous Manipulation

Moëz Cherif, *School of Engineering Science, Simon Fraser University, Burnaby, B.C, Canada*
Kamal K. Gupta, *School of Engineering Science, Simon Fraser University, Burnaby, B.C, Canada*

We describe a global motion planner for manipulating 3D objects by a dextrous robotic hand. We focus on the so-called re-configuration problem: find a feasible trajectory (motions and contact forces) that moves a hand-object system from an initial grasp to a final desired configuration of the object. The planner is designed as a three-level process combining a graph search on the configuration space of the object and a local planner that solves for feasible instantaneous quasi-static motions of the entire manipulation system. The planner is applied in simulation for achieving several complex re-configuration tasks for (piecewise-) smooth convex objects demonstrating the promise of our approach.

1 Introduction

1.1 Overview of the paper

We are interested in planning dextrous motions of a multi-fingered hand for manipulating an object held in its grasp. The problem interests us for two main reasons. The first reason is the intrinsic complexity of the problem (from a motion planning perspective) that involves constraints at several different levels – kinematics of contact motion between fingertips and the held object, no-collision between multiple moving bodies (i.e., the finger chains), (non-linear) friction constraints, effects of rolling and sliding contacts on the stability and the generation of instantaneous motions, constraints due to hand kinematics, and the large number of degrees of freedom of the manipulation system. The second reason is that such planning capabilities are important for enhancing "high-level" capabilities of artificial hands to carry out complex real tasks.

For a better insight into the dextrous manipulation problem, we focus on what we have called the "re-configuration problem": *starting from a given initial grasp of an object O, find motions and contact forces for the fingertips (and the corresponding positions/torques of the finger joints) that manipulate O to a desired final configuration.* As an example, consider the re-configuration of an ellipsoidal object O by three fingers D_i as shown in Figure 1. The initial grasp and the final desired configuration of the ellipsoid are shown at the top left and the top right, respectively. The problem is to find rolling, sliding, and/or twisting motions of the D_i's that move O to the final configuration while satisfying the above-mentioned constraints. The planner we will present in this paper solves this problem. In the middle of Figure 1, we show a sequence of intermediate grasps (configurations of the hand-object system) computed by the planner. These correspond to the sub-goals reached during the search for the global solution. Starting from the initial grasp (left), one can see the evolution (left to right) of the manipulation system until the final grasp (right). In the bottom of Figure 1, we show the final grasp reached by the planner. The resulting trajectories and the corresponding contact forces may be used as an input for a hybrid force/position controller for real applications.

Our ultimate goal is to design a *practical* and *global* scheme to plan 3D dextrous manipulation motions. However, as a first step toward tackling this complex problem, we make some simplifying assumptions. First, we neglect dynamic effects and consider quasi-static motions. This assumption is justified since a large number of precise manipulation tasks are carried out at low speeds, and moreover, it also has the advantage of avoiding posing the planning problem in the (very high dimensional) state space of the system – as it is, the configuration space for the hand-object system

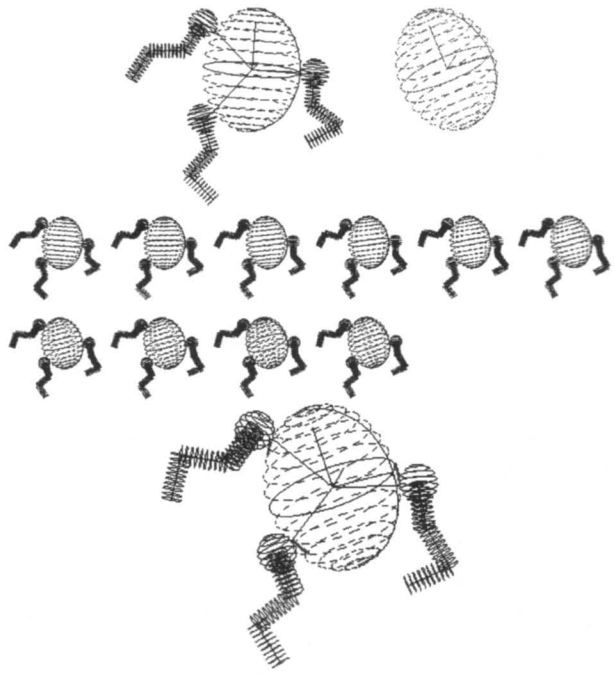


Figure 1: *Re-configuring an ellipsoid by three planar (3 dof) fingers.*

is quite large, the corresponding state space would double the dimensionality. Secondly (and somewhat more unrealistically), we ignore the inherent uncertainties that arise at the execution stage. For instance, a finger may not slide as intended in the planning stage; or, the contact sensors are imperfect. Subsequent stages of our research will consider these issues (see §5 for some remarks on this issue).

The basic idea behind our planning approach is as follows. Given that the robot hand is highly redundant kinematically, and since we are concerned only with achieving a desired configuration of the object, an intuitive solution consists in *decomposing the problem so that first, the object is moved independent of any constraints as if it were a free body.* For instance, a possible choice is to move the object along the shortest path (*w.r.t.* a given metric) toward its goal, i.e., a straight line in $SE(3)$, the C-space of the object. Knowing the nominal motion of the object, we now have to solve for the corresponding finger motions, *the inverse finger motion problem* (or, for brevity, the inverse problem). Note that the (governing equations for the) precise system may change during the search process. For instance, one could use all the fingers for some duration, and then keep some of them fixed, followed by using all the fingers again, resulting in a different set of governing equations at different time instants. Each of these actions corresponds to what we call a *manipulation mode* and the planner may choose different modes during the search process. Incorporating manipulation modes as a distinct level in our planning scheme is motivated by (i) the fact that humans naturally use several different "modes" while manipulating objects in hand, (ii) from a system design perspective, it offers flexibility in that a different *mode* can subsequently be integrated in the planner easily, and (iii) it enables the planner to escape "local minima" that may arise at certain grasps better than if we would have let lower search levels handle it exclusively.

The *inverse problem* is non-linear [7, 23, 29]. However, if we specify the relative velocity (at the contact point) between the object and the fingertips, the resulting problem is linear[1]. The dimensionality of the space of these relative velocities (we call them relative controls), is however, quite large ($\mathbb{R}^{2k}$ for pure rolling or pure sliding contacts with k fingers). To cope with this complexity, we considered only extreme values of the relative controls ($\pm$ maximum in each co-ordinate) and zero. The choice of searching the set of extremal relative controls has a parallel in optimal control theory (bang-bang control) and also in (non-holonomic) motion planning for car-like robots [16]. Furthermore, we choose randomly from this set, thereby tackling the combinatorics (for pure rolling or pure sliding, 2^k possible combinations of contact types exist corresponding to 3^{2k} control vectors). Indeed, random search of high dimensional spaces has been shown effective in gross motion planning [1, 16].

The inverse solution is used at the core of our planning scheme and operates as follows (algorithm `InstantaneousInverseSolution()`): *knowing the object motion, (1) we randomly choose (extremal) relative*

[1]This formulation is based on Peshkin's minimum power principle [23]. It will give a valid solution, but not necessarily the only one.

controls (between the object and each of the finger tips) at the contact points, and using it, we compute new contact configurations by solving Montana's kinematic equations [21], hence the new configurations of the fingertips; (2) we check if each fingertip lies within its workspace (forward kinematics of finger chains), and if there is an instantaneous displacement for the joints achieving that configuration (differential inverse kinematics of finger chains); and (3) knowing the new contact positions, we determine the contact forces by using quasi-static analysis.

The overall planning scheme, is then, a (interleaved) three-level process as follows (See Figure 2). At the first level, a graph search is applied on the C-space of the object that incrementally builds a tree of sub-goals within this space and through which the object could potentially be manipulated toward its desired goal. These sub-goals are generated assuming that the object can be moved for a period of time (which corresponds to an empirically chosen coarse discretization) along some canonical paths without considering any constraint. For each successive sub-goal, a particular sequence of *manipulation modes* is used. For the current implementation, we have identified three modes[2]: `move-all-fingers`: *manipulate the object by controlling the relative motions of all the fingertips*, (2) `fix-object (or, change-grasp)`: *keep the object stationary and move all the fingertips or a subset of them by controlling their rolling or sliding contact motions*, and (3) `fix-some-fingers`: *keep some fingers fixed and manipulate the object by rolling or sliding motions of the remaining fingers.* For each of these modes, algorithm `InstantaneousInverseSolution()` is used at a finer discretization of time to compute the desired finger motions and forces. If no such finger motion/forces exist, alternative sub-goals are generated at the appropriate levels.

We have implemented this planning approach in simulation. Our planner incorporates the constraints (as mentioned in the introduction) at several different levels, tackles the high dimensionality of the C-space and the control space of the manipulation system, and is capable of re-configuring both polyhedral and smooth (e.g., ellipsoid) objects. The example in Figure 1 was generated by this planner.

[2]These were chosen in an intuitive manner in the current implementation. The main idea was to illustrate the use of different modes in the overall planning scheme.

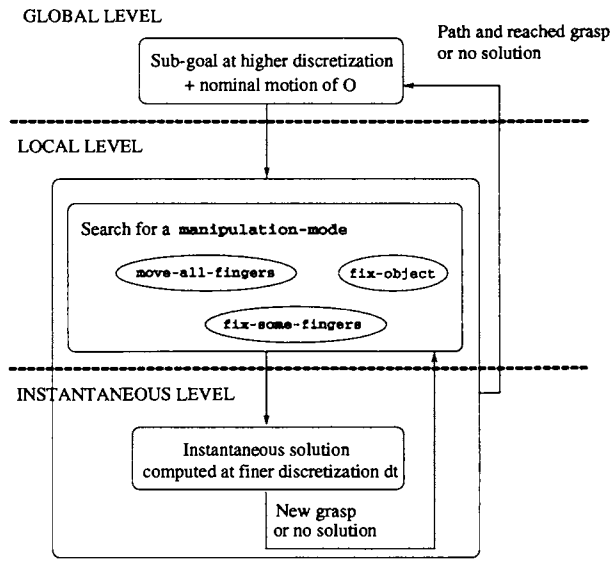


Figure 2: *A schematic view of the planning algorithm.*

1.2 Other related work

Most work in dextrous manipulation has addressed issues related to mechanical design and control aspects [8, 9, 15, 20, 22, 26], grasp characterization including synthesis and analysis of (form- and force-) closure grasps (see [27] for a survey), and generation of instantaneous motions [3, 7, 10, 11, 25, 29].

Li et al. [19] presented a general formulation of the motion planning problem for changing grasps but did not present algorithms for solving it. A planning algorithm was proposed by Trinkle and Paul for achieving quasi-static re-orientation of polygonal objects in the plane by sliding contacts and power grasps [30]. The algorithm however does not seem to be straightforward for 3D manipulation. More recently, Rus presented a local planning scheme for dynamic re-orientation of polyhedra [25]. Based on it, Gupta presented a global planner for re-orienting polyhedral objects [13]. Al-

though important, this body of work is in an idealized domain with frictionless contacts and point fingers. Leveroni and Salisbury have introduced the notion of *grasp gaits* i.e., grasps at which some fingertips can be re-located by breaking contacts, in order to enlarge the range of possible re-orientation of the held object [17]. This could be incorporated as an additional *manipulation mode* within our planner.

Zefran *et al.* have studied multi-fingered manipulation problem within a hybrid control type framework, where the state space is partitioned into regions corresponding to different grasp configurations [33]. However, they assume that the sequence of regions (i.e., grasp configurations) is given a priori. Our approach, on the other hand, performs a search across these levels. Nevertheless, there may be some interesting connections with our work and we will explore them. Along another line, researchers have considered the effect of non-holonomic rolling constraints on motion planning [2, 14]. For instance, Bicchi and Sorrentino studied manipulation of a sphere by a minimally actuated hand composed of two parallel planes [2]. Han *et al.* considered three 6-dof fingers with planar fingertips moving a sphere through rolling contacts [14].

1.3 Contribution of the paper

Given the complex nature of the problem, our approach is somewhat empirical. Nevertheless, it brings several novel ideas to dextrous manipulation: a novel decomposition of the overall problem, use of random techniques (borrowed from classical gross motion planning) for high dimensional configuration spaces, and finally an integrated implementation for 3D dextrous manipulation with simulated multi-fingered hands. We believe that this is the first comprehensive global motion planner for 3D dextrous manipulation. Admittedly, we have made some simplifying assumptions, in particular that the execution of these motions is perfect. In reality, this is not the case, and the interaction of execution uncertainty with the planner, a fundamental problem, merits further investigation in this context.

1.4 Organization of the paper

§2 states the re-configuration problem. §3 presents the planning algorithm, and §4 describes simulation results.

2 The Problem Formulation

2.1 Hand-object system description

The manipulation system is composed of k rigid articulated fingers D_i of an arbitrary structure grasping a 3D rigid object O – a polyhedron or a smooth object (as in Figure 3). Each D_i has n_i joints and is fixed at its base *w.r.t.* a frame fixed in the workspace. O and the D_i's are considered to be of known geometry, and the fingertips F_i of the D_i's are assumed to be (piecewise-) smooth and convex. We are interested in tasks where: (1) each D_i contacts O at a single point c_i located on its fingertips F_i, (2) the contacts are frictional and not sticky, and obey Coulomb model of friction, and (3) the system is moving at low velocities so that a quasi-static analysis can be applicable. Finally, we assume that we can control (via a low-level controller) rolling and sliding motions of the contacts, and that the effect of uncertainty can be neglected.

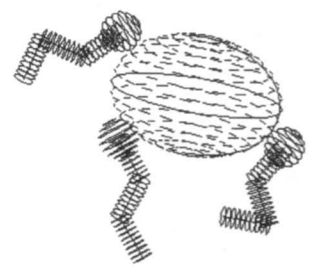


Figure 3: *Three 3-dof fingers grasping an ellipsoid.*

2.2 Notations and definitions

The configuration of each body X (e.g., O or F_i) is denoted by $g_X = (r_X, R_X) \in SE(3)$, where r_X and R_X are the position vector and rotation matrix (*w.r.t.* the palm frame $\mathcal{F}_P$) of a frame fixed on X. The C-space of O (resp. the fingertips-object system) is $SE(3)$ (resp. $SE(3)^{k+1}$) and is denoted by CS_O (resp. CS). A grasp $\mathbf{g} = (\mathbf{g_O}, \mathbf{g_{F_1}}, \ldots, \mathbf{g_{F_k}})$ is an element of CS.

Knowing the surface parameterization (or charts) of O and F_i, we attach at each contact c_i a Gaussian frame $\mathcal{F}_{c_i,O}$ on O and Gaussian frame $\mathcal{F}_{c_i,F_i}$ on F_i (see Fig. 4). Let $g_{c_i,O} = (r_{c_i,O}, R_{c_i,O})$ be the configuration of $\mathcal{F}_{c_i,O}$ *w.r.t.* $\mathcal{F}_O$, and $g_{c_i,F_i} = (r_{c_i,F_i}, R_{c_i,F_i})$ be the configuration of $\mathcal{F}_{c_i,F_i}$ *w.r.t.* $\mathcal{F}_{F_i}$. The contact configuration at c_i is given by $\eta_i = (u_{i,O}, v_{i,O}, u_{F_i}, v_{F_i}, \psi_i)^T \in I\!R^5$, where $(u_{i,O}, v_{i,O})$ (resp. (u_{F_i}, v_{F_i})) is the position of c_i on O (resp. F_i), and ψ_i is the angle between the x-axes of $\mathcal{F}_{c_i,O}$ and $\mathcal{F}_{c_i,F_i}$. When the k contacts are maintained, it can be shown that CS is (locally) diffeomorphic to $SE(3) \times I\!R^{5k}$.

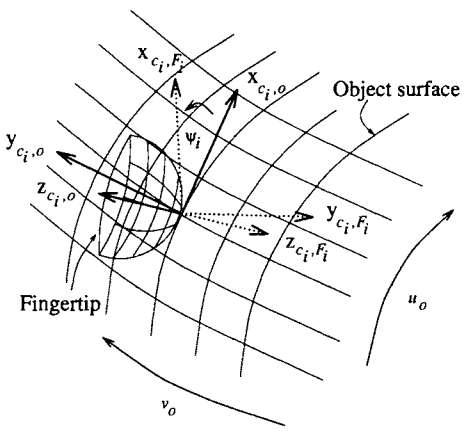


Figure 4: *The contact frames. The solid frame is $\mathcal{F}_{c_i,O}$, and the dotted frame is $\mathcal{F}_{c_i,F_i}$.*

2.3 Kinematics of manipulation

Let $\Theta_i \in I\!R^{n_i}$ be the joint vector of a finger D_i in contact with O at a point c_i, and let $\tilde{J}_i(\Theta_i, c_i) \in I\!R^{6\times n_i}$ be its Jacobian at c_i. Let $\dot{q}_r = ({v_r^1}^T, {\omega_r^1}^T, \ldots, {v_r^k}^T, {\omega_r^k}^T)^T \in I\!R^{6k}$ be the velocity vector of the frames $\mathcal{F}_{c_i,O}$ relative to the frames $\mathcal{F}_{c_i,F_i}$ written *w.r.t.* $\mathcal{F}_{c_i,O}$. For a k-contact grasp, $\dot{q}_r$ is given by the matrix form [22, 26]:

$$G^T \dot{q}_O - A_\Psi \tilde{J}(\Theta, c)\dot{\Theta} = \dot{q}_r \qquad (1)$$

where $G \in I\!R^{6\times 6k}$ is the grasp matrix, $\dot{q}_O = (v_O^T, \omega_O^T)^T \in I\!R^6$ is the velocity vector of O, $\dot{\Theta}$ is the joint velocity vector of the k fingers, $\tilde{J}(\Theta, c)$ is the Jacobian of the k fingers at the contact points c_i, and $A_\Psi \in I\!R^{6k\times 6k}$ is a diagonal matrix defined by: $A_\Psi = diag(A_{\Psi_1}, A_{\Psi_1}, A_{\Psi_2}, A_{\Psi_2}, \ldots, A_{\Psi_k}, A_{\Psi_k})$, where A_{Ψ_i} is the rotation matrix between contact frames $\mathcal{F}_{c_i,O}$ and $\mathcal{F}_{c_i,F_i}$. $\tilde{J}(\Theta, c)$ is given by the $6k \times (n_1 + \ldots + n_k)$ matrix $diag(\tilde{J}_1(\Theta_1, c_1), \ldots, \tilde{J}_k(\Theta_k, c_k))$.

The evolution of a contact configuration η_i is given by solving the following system of differential equations (Montana's equations [21]) which relate $\dot{q}_r$ and $\dot{\eta}_i$:

$$\begin{cases} \begin{bmatrix} \dot{u}_{i,O} \\ \dot{v}_{i,O} \end{bmatrix} = M_{i,O}^{-1} K_i^{-1} \left(\begin{bmatrix} -\omega_{r,y}^i \\ \omega_{r,x}^i \end{bmatrix} - \tilde{K}_{i,F_i} \begin{bmatrix} v_{r,x}^i \\ v_{r,y}^i \end{bmatrix} \right) \\ \begin{bmatrix} \dot{u}_{F_i} \\ \dot{v}_{F_i} \end{bmatrix} = M_{i,F_i}^{-1} R_\psi K_i^{-1} \left(\begin{bmatrix} -\omega_{r,y}^i \\ \omega_{r,x}^i \end{bmatrix} + K_{i,O} \begin{bmatrix} v_{r,x}^i \\ v_{r,y}^i \end{bmatrix} \right) \\ \dot{\psi}_i = \omega_{r,z}^i + T_{i,O} M_{i,O} \begin{bmatrix} \dot{u}_{i,O} \\ \dot{v}_{i,O} \end{bmatrix} + T_{i,F_i} M_{i,F_i} \begin{bmatrix} \dot{u}_{F_i} \\ \dot{v}_{F_i} \end{bmatrix} \end{cases} \qquad (2)$$

where $M \in I\!R^{2\times 2}$, $K \in I\!R^{2\times 2}$, and $T \in I\!R^{1\times 2}$ are the metric, curvature, and torsion tensor forms, respectively, associated with a given object and computed at the point of contact, $K_i = (K_{i,O} + \tilde{K}_{i,F_i})$ is the relative curvature form at contact c_i, $\tilde{K}_{i,F_i} = R_{\psi_i} K_{i,F_i} R_{\psi_i}$, and $R_{\psi_i} \in I\!R^{2\times 2}$ is a rotation matrix corresponding to ψ_i (see [18, 21] for more details on the computation of these forms and matrices). The system (2) holds if K_i is invertible, and the objects in contact are rigid and locally smooth, valid assumptions in our case. Knowing the current grasp $\mathbf{g}$, hence the η_i's, and the vector $\dot{q}_r$, the numerical integration of (2) for each contact c_i provides, at each instant, the new contact configuration. This configuration can be used for computing the corresponding new grasp.

Note that in our planning scheme, the search space is effectively the composition of the velocity space of O ($\dot{q}_O$) and the space of relative velocities (or, as we call them relative controls) of F_i's ($\dot{q}_r$) *w.r.t* O. We refer to this composite space as the control space. For instance, if all the F_i's have to move simultaneously with either pure rolling or pure sliding motions, this space is given by: $I\!R^6 \times I\!R^{2k} - I\!R^6$ is the velocity space of O and $I\!R^2$ is the relative controls space of a contact.

2.4 The problem

We concentrate on the case when the object is manipulated by either rolling, sliding, or twisting contacts – in other words, breaking contacts is not allowed. Hence, an instantaneous motion is said to be feasible (or admissible) if O is grasped at k contact points while avoiding collision between the links of the different fingers and satisfying that the contacts occur only at the F_i's. Formally, the first constraint is given by:

$$G_z^T \; \dot{q}_O - A_\Psi \tilde{J}_z(\Theta, c)\dot{\Theta} \; = \; 0 \qquad (3)$$

where G_z is the z-row of G and $\tilde{J}_z$ is the matrix composed of the k z-rows of the $\tilde{J}_{i,z}$'s. This constraint means that z-components $\dot{q}^i_{r,z} = 0, i = 1 \ldots k$. In addition, the k contact forces f_i acting on O have to balance the gravity wrench $f_{O,ext}$, i.e., $G\, f + [f^T_{O,ext}, 0, 0, 0]^T = 0$, and obey Coulomb friction model, i.e., $f^2_{i,x} + f^2_{i,y} \leq \mu^2 f^2_{i,z}$ and $f_{i,z} \leq 0$, where μ is the coefficient of contact friction, and $f_{i,s}$ $(s = x, y, z)$ is the s-component of the contact force f_i *w.r.t.* the contact frame $\mathcal{F}_{c_i,O}$.

The re-configuration problem is now formally stated as: *Given an initial grasp configuration* $\mathbf{g}_{start} \in CS$ *and a configuration* $g_{O,goal} \in CS_O$ *of* O*, find a feasible (piecewise-) smooth trajectory (a curve in* CS*) and the corresponding contact forces and joint torques that takes the hand-object system from* $\mathbf{g}_{start}$ *to a grasp* $\mathbf{g}(t_f)$ *in a finite period of time* t_f *such that* $g_O(t_f) = g_{O,goal}$. The condition on $\mathbf{g}(t_f)$ means that the goal to be reached in CS corresponds to a region $\{g_{O,goal}\} \times SE(3)^k$ (and not a single point). For ensuring the quasi-static assumption, we constrain the solution so that the velocities of O and the relative controls of the F_i's *w.r.t.* O are all bounded.

3 The Three-Level Motion Planner

3.1 Global level: Searching object C-space

Starting from the initial grasp $\mathbf{g}_{start}$, we generate for O a set of sub-goals located in a given neighborhood of its initial configuration $g_{O,start}$. Each of these sub-goals is computed assuming that O can move, for a given period of time ΔT, with a constant nominal velocity $\tilde{\dot{q}}_O$, that corresponds to a canonical path – a straight line motion in the object C-space CS_O without any constraints. Among these sub-goals, we choose one which minimizes an estimation of a cost function (e.g., the length) defined on the trajectory to the final desired goal configuration $g_{O,goal}$.

This global level is implemented as a grid search (e.g., an A^*-type algorithm) on a discretized representation of CS_O and aims to capture the connectivity of CS_O by expanding incrementally a tree of sub-goals having $g_{O,start}$ as a source. The expansion of this tree is performed by interleaving the processes of generating sub-goals and checking if a sub-goal can be achieved while coping with the manipulation constraints (using a local planner, described in §3.2), until a sub-goal lies within a small neighborhood of $g_{O,goal}$ or no new nodes (sub-goals) can be generated (i.e., the search space is fully explored and no solution is found by the planner). The discretized representation of CS_O is given by a collection of regular 6 dimensional adjacent cells, each defined on $I\!R^3 \times S^3$, where S is the unit circle (space of orientation angles). ΔT is chosen so that successive sub-goals are within adjacent cells. In order to limit the search, each cell (or node) is allowed to be visited only once, i.e. only a single sub-goal can be generated within a cell, hence each node is pointed to by only one arc in the search graph. Besides, the set of cells is not built a priori, and only the cells visited during the search process are maintained. For a further details of this scheme, see [4, 6]).

3.2 Local level: Searching relative control space

Let $g_{O,g}$ be a given sub-goal for which we have to check if it is achievable starting from a grasp $\mathbf{g}_s$, and let $\tilde{\dot{q}}_{O,g}$ be the nominal velocity applied by the higher level to generate $g_{O,g}$ from $g_{O,s}$ corresponding to $\mathbf{g}_s$. The local level aims to find a (piecewise-) smooth trajectory moving the system from $\mathbf{g}_s$ to a grasp $\mathbf{g}$ so that O is within a small neighborhood of $g_{O,g}$. For this purpose, we make use of a finer discretization of time (i.e., a much smaller increment of time $\delta t << \Delta T$) for computing accurate feasible instantaneous motions and we apply, at each instant, one of the three manipulation modes: `move-all-fingers`, `fix-object`, and `fix-some-fingers` (presented in §1).

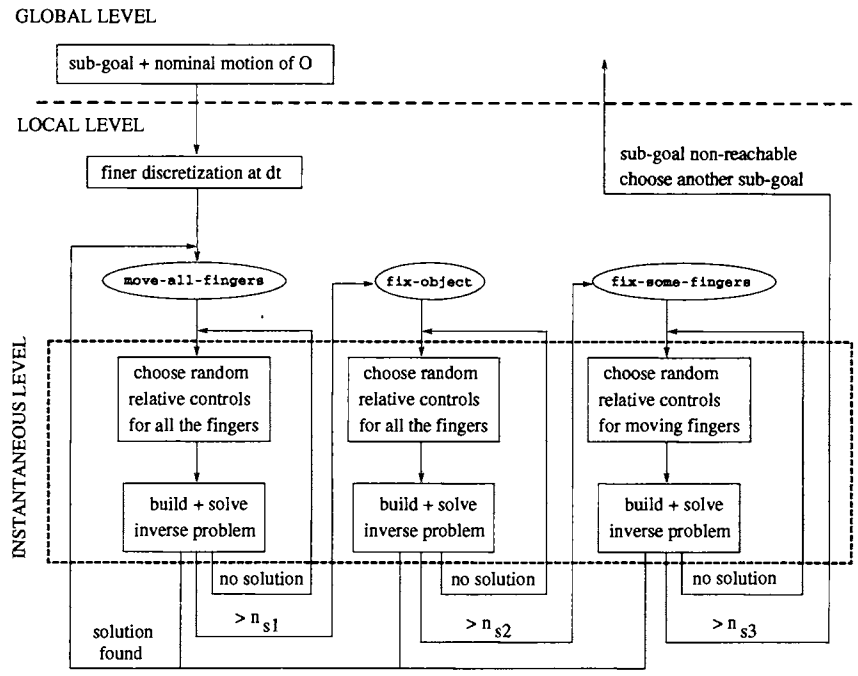


Figure 5: *A schematic view of local level and how different manipulation modes are used within it.*

A schematic description of the local level is shown in Fig. 5. Starting from $\mathbf{g}_s$, the inverse solution is applied for the period δt with `move-all-fingers` as the manipulation mode and $\tilde{\dot{q}}_{O,g}$ as the nominal velocity vector for O. When an admissible grasp has been achieved, we keep applying `move-all-fingers` until $g_{O,g}$ is reached or a failure occurs (i.e., one of the task constraints is not satisfied). In the latter case, we try performing the same scheme with a new random combination of relative controls for an arbitrary bounded number of trials. This bound, denoted $n_{\mathbf{S1}}$, is due to the random nature of the relative control space exploration and is used to terminate the local planner. If no feasible solution can be found, `fix-object` is applied starting from the last reached admissible grasp. Again, when a failure is detected, `fix-object` mode is used for up to $n_{\mathbf{S2}}$ (somewhat arbitrary) number of trials. If an admissible grasp is reached, the planner reverts to `move-all-fingers` again. If there is no solution after applying `fix-object`, a new series of trials (bounded by an arbitrary number $n_{\mathbf{S3}}$) is performed using `fix-some-fingers` as the manipulation mode. If a solution is found, the planner pursues moving O towards $g_{O,g}$ by performing `move-all-fingers`. Otherwise $g_{O,g}$ is said to be unreachable and the manipulation system is deemed in a local minima. The planner then tries another sub-goal provided by the higher level.

In the following, we briefly comment on why the local planner is structured as it is. As a matter of illustration, let us consider the structure of the object C-space CS_O (and the corresponding cells) when a grasp $\mathbf{g}_1$ is reached as shown at the top of Figure 6. The projection on CS_O of the planned solution between the initial grasp and $\mathbf{g}_1$ is also shown. Depending on the current configuration g_O of O, there may exist no new grasp when O is moved towards its sub-goal *SG1* while all the fingertips are moving (i.e., the system is deemed in a local minimum). This is illustrated by the dark region *FR1* that correspond to forbidden grasps. $\mathbf{g}_1$ is then the last feasible grasp the local planner can reach after trying all possible trials with `move-all-fingers` mode. By introducing one of `fix-object` and `fix-some-fingers` modes, *FR1* can

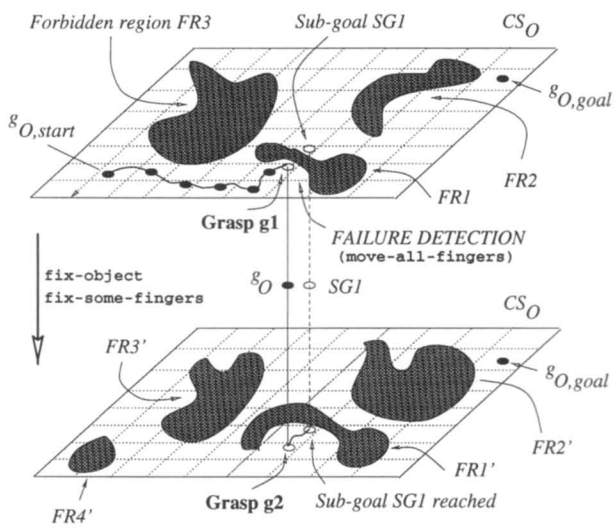


Figure 6: *Changing structure of the object C-space CS_O. The forbidden regions FR1 has moved to FR1′ when planning a feasible motion towards sub-goal SG1.*

be avoided; thereby making it possible to escape the local minimum. With `fix-object`, the local planner will try to find a new grasp $\mathbf{g}_2$ (while O is stationary) from which it can be moved later towards *SG1*. This is schematically shown at the bottom of Figure 6. In the new grasp $\mathbf{g}_2$, which corresponds to a different slice of the full C-space of the manipulation system, *FR1* has been removed (or its structure has been changed to *FR1′*) and now O can be moved. Similarly, `fix-some-fingers` aims to enlarge the range of object motions mainly in the neighborhood of the boundaries of forbidden regions due to reachability constraints due to the finger kinematics. Indeed, by introducing additional kinematic constraints, `fix-some-fingers` will make it possible to explore a larger set of velocities for O since the motion of O will not necessarily lie on the straight-line between the sub-goals (further details are given in §3.4).

We now explain the instantaneous inverse solution for the three manipulation modes in more detail.

3.3 Instantaneous level: Inverse solution

The inverse problem is: *given a velocity vector $\dot{q}_O$ of O (i.e., an instantaneous motion), one has to find a feasible motion of the fingers in contact with ∂O that achieve $\dot{q}_O$.* Recall that (§1.1) our approach is to choose a random relative control ($\dot{q}_r$), use it to compute new contact configuration (using Montana's kinematic equations), and then use quasi-static analysis to determine contact forces. We give below a pseudo-code description of algorithm `InstantaneousInverseMotion()` that achieves this and then describe each step in more detail. Starting from a current grasp $\mathbf{g}(t)$ and knowing the velocity $\dot{q}_O$ of O which is given by $\tilde{\dot{q}}_{O,g}$, it operates as follows:

algorithm `InstantaneousInverseMotion`($\mathbf{g}(t), \dot{q}_O$)
  choose a random relative control $\dot{q}_r$;
  compute the new contact configuration η_i
    by numerically solving (2) with time increment δt;
  compute the new object configuration $g_O(t+\delta t)$
    by moving O by $\dot{q}_O$ for time δt;
  if (there exists a solution $\dot{\Theta}$
      to the inverse kinematics of the fingers) {
    compute the new fingers configuration $g_{F_i}(t+\delta t)$;
    compute the new configuration of the fingers D_i;
    if ($\mathbf{g}(t+\delta t)$ is not reachable)
      return(`FALSE`);
    if (collision($\mathbf{g}(t+\delta t)$))
      return(`FALSE`);
    if (LP (8) has no solution)
      return(`FALSE`);
    return(`TRUE`); }
  return(`FALSE`);
endalgorithm;

Note that the precise details differ depending on which mode is selected. For instance, for the modes `move-all-fingers` and `fix-object`, random relative control is assigned for each contact, whereas, for `fix-some-fingers`, random relative control is assigned only for moving fingers (for fixed fingers, it depends on and is determined from the object velocity, see Section 3.4). For the discussion below (§3.3.1), implicitly assume that we are using `move-all-fingers`. We now explain different steps in `InstantaneousInverseMotion()` in more detail.

3.3.1 Discretizing the relative control space

We assume that we are able to execute pure motions (rolling and/or sliding) at contacts. To tackle the high dimensionality of the relative control space, we consider

a discrete representation of it which is then searched randomly. The relative controls for rolling and sliding are respectively within the following discrete sets:

$$\begin{cases} v^i_{r,x} = v^i_{r,y} = v^i_{r,z} = \omega^i_{r,z} = 0 \\ (\omega^i_{r,x}, \omega^i_{r,y}) \in \{-\omega^{max}_{r,x}, 0, \omega^{max}_{r,x}\} \times \{-\omega^{max}_{r,y}, 0, \omega^{max}_{r,y}\} \end{cases} \tag{4}$$

$$\begin{cases} v^i_{r,z} = \omega^i_{r,x} = \omega^i_{r,y} = \omega^i_{r,z} = 0 \\ (v^i_{r,x}, v^i_{r,y}) \in \{-v^{max}_{r,x}, 0, v^{max}_{r,x}\} \times \{-v^{max}_{r,y}, 0, v^{max}_{r,y}\} \end{cases} \tag{5}$$

where $(\omega^{max}_{r,x}, \omega^{max}_{r,y})$ (resp. $(v^{max}_{r,x}, v^{max}_{r,y})$) are constant upper-bounds for rolling (resp. sliding) controls along x and y axes (cf. §2.4). Instantaneous motion directions of a contact c_i obtained with this discrete set of relative controls are shown in Figure 7. Dark straight lines correspond to sliding motions, and rotations about x and y axes of $\mathcal{F}_{c_i,O}$ correspond to directions of rolling.

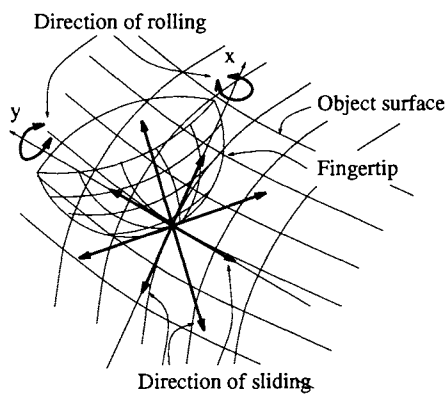


Figure 7: *The relative controls at a given contact point.*

3.3.2 Computing a new grasp

Knowing the vector of relative controls $\dot{q}_r$, we solve Montana's kinematic equations (2) for finding the new contact configuration $\eta_i(t+\delta t)$. This configuration is then used with the surface parameterizations of the F_i's and O for computing the new contact position vectors $c_{c_i,O}$ and c_{c_i,F_i}. Knowing these vectors and the new configuration $g_O(t+\delta t)$ of O (obtained after a displacement of velocity $\dot{q}_O$), we solve for the new configuration of the fingertips *w.r.t.* the palm frame using the relationship: $R_O c_{c_i,O} + r_O = R_{F_i} c_{c_i,F_i} + r_{F_i}$.

When O is smooth, the positions of points of contacts c_i (obtained as solution of Montana's equations) lie naturally on ∂O. When O is a polyhedron, the solution of (2) can provide a solution $(u_{i,O}, v_{i,O})$ which is out of the boundary of the contacted face. This is interpreted as a transition motion between two adjacent faces [5]. Once the grasp configuration is determined, tests concerning reachability of this grasp and collision safety are carried out. The first test consists in solving the inverse kinematics of the finger chains for finding an admissible velocity joint vector that can be applied to achieve $\dot{q}_r$. Knowing the grasp $\mathbf{g}(t)$, $\dot{q}_r$ and the velocity $\dot{q}_O$ of O, we solve for $\dot{\Theta}$ using (1). In our current implementation, we use the ***singular value decomposition*** algorithm for solving this system [12]. If a vector $\dot{\Theta}$ exists, we compute the new joint vector Θ_i of the D_i's.

The resulting configurations of O and of the D_i's links are used for performing the collision detection test. This includes collision between links of the same finger, links of two different fingers, and collision between the object and parts of the fingers other than the fingertips. Collision detection is based on a hierarchical representation of the links by spheres [28]. If the grasp is collision-free, the next step is to solve for frictional contact forces satisfying equilibrium of the manipulation system.

3.3.3 Solving for equilibrium contact forces

For pure rolling contacts, the friction constraints are linearized by considering an inscribed pyramid within the friction cone at each contact point (as in [15, 24]). In our implementation, the inscribed pyramid is described by 4 half-planes:

$$\begin{cases} -\mu' \, f_{i,z} \leq f_{i,x} \leq \mu' \, f_{i,z} \\ -\mu' \, f_{i,z} \leq f_{i,y} \leq \mu' \, f_{i,z} \end{cases} \tag{6}$$

where μ' is deduced from the initial friction coefficient μ and the desired number of approximating planes. When the motion mode of a fingertip F_i is of pure sliding, i.e. $\omega^i_{r,x} = \omega^i_{r,y} = \omega^i_{r,z} = 0$, the tangential (friction) force at a given contact point must lie in the same plane as the translational relative controls (given by $v^i_{r,x}$ and $v^i_{r,y}$) and must be of the same sign. The first constraint can be stated as: $det(f_i, n_i, v^i_{r,t}) =$

$f_i.(n_i \times v^i_{r,t}) = 0$, where $det()$ is the determinant function. The condition on the direction of the contact force and the sliding velocity is: $f_i.v^i_{r,t} \geq 0$. Since the vector of sliding velocity control $(v^i_{r,x}, v^i_{r,y})$ is known, we reformulate the former conditions in the form:

$$\begin{cases} f_{i,x} & = & \alpha^i v^i_{r,x} \\ f_{i,y} & = & \alpha^i v^i_{r,y} \\ \alpha^i |v^i_{r,t}| & = & -\mu f_{i,z} \end{cases} \quad (7)$$

where $\alpha^i \geq 0$ is the unknown instead of $f_{i,x}$ and $f_{i,y}$.

Assuming a quasi-static analysis, and knowing the new grasp and the relative control vector used to compute it, the contact forces are determined by solving the following linear program $(LP1)$:

$$\begin{cases} Minimize & P_v = -\dot{q}_O \,.\, (f_{O,ext} + G\ f') \\ subject\ to & f_{i,z} \leq 0 \quad i = 1 \ldots k \\ & G \left[\ f_1^T, 0_3^T, \ldots, f_k^T, 0_3^T\ \right]^T = \begin{bmatrix} -f_{o,ext} \\ 0_3 \end{bmatrix} \\ and & (6) \text{ when } c_i \text{ is rolling contact} \\ or & (7) \text{ when } c_i \text{ is sliding contact} \end{cases} \quad (8)$$

where P_v corresponds to the instantaneous power of the manipulation system [23], and f' is a vector equal to the wrench f where the z components are replaced by 0. According to [23], forces involved in P_v are composed of all forces acting on O except forces of constraints. These refer to normal forces $(f_{i,z})$ applied at the contact points for maintaining contact between the fingertips and the object. Note that the friction constraints depend on the mode of contact at a fingertip – rolling or sliding motion.

For each finger D_i, the resulting contact forces f_i can be used to compute at each time the torque vector $\tau_i \in I\!R^{n_i}$ that can be applied at the finger actuators. This is done by using the system: $\tau_i = \tilde{J}_i(\Theta_i, c_i)^T\ f_i$.

3.4 Revisiting the instantaneous solution for `fix-some-fingers` mode

In this section, we revisit some steps of the instantaneous solution when the manipulation mode is `fix-some-fingers`. Let $\mathbf{g}(t)$ be the current grasp, and let $\tilde{\dot{q}}_O(t)$ be the nominal velocity that takes O from $g_O(t)$ to its sub-goal $g_{O,g}$ along a straight line motion in the C-space CS_O. When `fix-some-fingers` is processed for the first time, $\tilde{\dot{q}}_O(t)$ is equal to the nominal velocity $\tilde{\dot{q}}_{O,g}$ used by the higher planning level to generate $g_{O,g}$ (see 3.2).

The manipulation mode `fix-some-fingers` is performed as follows. First, we choose randomly a subset of $k' \leq k$ fingertips to be fixed in the workspace. *Wlog*, let $F_1, \ldots, F_{k'}$ be these fingertips. When O is moving, $\dot{q}_O$ becomes constrained by the following $k' \times 6$ linear system:

$$G^T_{i,z}\dot{q}_O = 0\ , \quad i = 1, \ldots, k' \quad (9)$$

where $G^T_{i,z}$ is the z column corresponding to the terms of G encoding the contact position of F_i for $\mathbf{g}(t)$. This system has its effect in reducing the number of degrees of freedom of O to $6-k'$, and the problem now is to determine these degrees of freedom. A first solution consists in selecting $6-k'$ parameters of $\dot{q}_O$ and assigning them to the corresponding parameters of the nominal velocity $\tilde{\dot{q}}_O(t)$, and solving (9). In absence of degeneracy, this has a unique solution. The selection of the $6-k'$ parameters can be done exhaustively or randomly among $C_6^{k'}$ combinations. However, depending on the current grasp configuration, one has to run several trials before avoiding degeneracy. Besides, the solution of (9) does not ensure that the resulting parameters of $\dot{q}_O$ are bounded. Therefore, we pose the velocity determination problem as a linear program aiming to find the closest velocity $\dot{q}_O$ to $\tilde{\dot{q}}_O(t)$. This program $(LP2)$ is formulated as follows:

$$\begin{cases} Minimize & \tilde{\dot{q}}^T_O(t).\dot{q}_O - \tilde{\dot{q}}^2_O(t) \\ subject\ to & (9) \\ & -v_O^{max} \leq v_{t,O} \leq v_O^{max} \quad t = x, y, z \\ and & -\omega_O^{max} \leq \omega_{t,O} \leq \omega_O^{max} \quad t = x, y, z \end{cases} \quad (10)$$

and is solved by the *simplex* method. When (10) has no solution, another set of fingers is fixed randomly and considered for determining an admissible $\dot{q}_O$. Knowing $\dot{q}_O$, we generate for the $k-k'$ moving fingertips random relative control vectors (as in other two modes). For the fixed F_i's, the relative controls are determined by solving the system

$$\dot{q}^i_r = G^T_i\,\dot{q}_O, i = 1, \ldots, k' \quad (11)$$

where $G_i \in I\!R^{6\times 6}$ is given by columns of the grasp matrix G corresponding to F_i. Note that, depending

on the contact positions on ∂O and $\dot{q}_O$, the resulting relative motions at the fixed F_i's can be either pure motions, or a combination of sliding/rolling and twisting, i.e., $\dot{w}^i_{r,z} \neq 0$.

Using this combined set of relative controls, algorithm `InstantaneousInverseMotion()` is used to solve the inverse problem, i.e., to compute the trajectories (motion and force) of the moving fingertips. If an admissible trajectory is found, we revert to `move-all-fingers` mode. Otherwise, a new trial is considered for finding the F_i's to be fixed and applying the same process.

Let us now have a closer look at the motion of O within its C-space after having displaced it with the constrained velocity $\dot{q}_O$. Since some parameters of $\dot{q}_O(t)$ resulting from (10) are different from those of $\tilde{\dot{q}}_O(t)$, $g_O(t+\delta t)$ will be different than the sub-goal $g_{O,g}$. Maintaining $\tilde{\dot{q}}_O(t)$ as a nominal velocity while subsequently applying the mode `move-all-fingers` may take O far from $g_{O,g}$ (in its C-space) and decreases the ability of the local planner to converge toward the sub-goal. For this reason, each time the mode `fix-some-fingers` is successfully performed, we update the nominal velocity $\tilde{\dot{q}}_O(t+\delta t)$ to be used later by `move-all-fingers` and set it to a vector that makes O move toward $g_{O,g}$ (see Figure 8). Thus, we set the translational velocity $\tilde{v}_O(t+\delta t)$ to $v_O^{max} r_{1,2}$, where $r_{1,2}$ is the unit vector along the direction between $r_O(t+\delta t)$ and $r_{O,g}$ – current and sub-goal positions of O. For rotational velocity, we consider $\tilde{\omega}_O(t+\delta t) = \omega_O^{max} r_{\theta,1,2}$, where $r_{\theta,1,2}$ is the unit vector along the rotation direction $\theta_{1,2}$ given by: $R(\theta_{1,2}) = R_O^T(t+\delta t) R_{O,g}$ – the shortest rotation motion that takes O to $R_{O,g}$.

3.5 Complexity and completeness

As we pointed out in §1, the diversity of the constraints coupled with the high number of degrees of freedom of the manipulation system makes the motion planning problem very complex. This has been tackled in our approach by decomposing the search process into several levels and by applying an approximate scheme based on sampling the search spaces to be explored – the object and finger motion spaces.

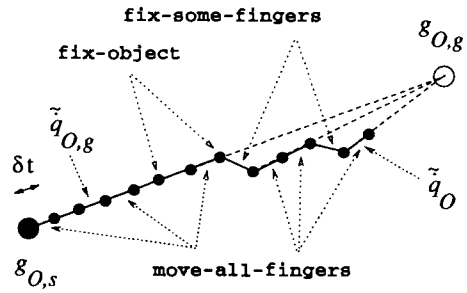


Figure 8: *Resulting path of O in CS_O when* `fix-some-fingers` *is introduced as a manipulation mode. The bold points refer to configurations of O and each step is computed for the period of time δt. When* `fix-object` *is applied, $\dot{q}_O = 0$ so O stays at rest.*

3.5.1 Complexity

In this section, we give a somewhat informal analysis of what would be the expected complexity of overall the motion planner. The discretization of CS_O results in $n_{cell} = n_x^3 n_\theta^3$ cells (n_x and n_θ are determined by the discretization along the translation/orientation axes). As we mentioned in §3.1, each cell is visited only once. This leads in the worst-case to a graph with n_{cell} arcs and nodes. The heuristic we have used is based on the Euclidean distance, and so it is locally consistent [16]. In this case, the worst-case complexity of the A^* search is $O(t_{lp} n_{cell} \log n_{cell})$, where t_{lp} is the worst-case time the local planner can take.

In practice, the workspace where the object motion has to take place is small and is of the size of the interior of the hand. Gross motions when they are needed can be handled by a manipulator on which the hand is mounted. Thus if ΔT is not chosen too small, the cell representation of CS_O would not require a high number of cells along the translation axes. Similarly, the discretization along rotation axes (i.e., n_θ) would not require inordinately large number of cells, even if O takes its rotational motion in the entire space $SO(3)$.

Due to the random nature of the local planner, it is difficult to estimate exactly what would be t_{lp}. For this reason and for ensuring the termination of the lo-

cal planner, we considered that the number of trials for each manipulation mode is upper-bounded. Thus, t_{lp} is of the form $n_{is}t_{is}$, where n_{is} is the maximum number of trials, and t_{is} is the worst-case time taken by the instantaneous solution. This includes mainly the times needed to solve the contact kinematics, collision detection at the fingers, the linear programs (1 or 2 depending on the applied manipulation mode), and the inverse kinematics for the moving fingers.

3.5.2 Completeness

Clearly, since the planning algorithm is based on a discretized search, it is not exact and is not complete. Furthermore, due to the random search made at the local level, it is not resolution-complete. But, in practice, we have solved several non-trivial tasks. We believe that the way the planner is structured – multi-level planning, combination of graph search and random techniques, and the manipulation modes – strongly exploits the redundancy of the manipulation system, thereby yielding a solution often.

4 Simulation Results

The planning algorithm has been implemented on a Silicon Graphics Indy R4400SC workstation. *Our ultimate goal is to empirically bring out effects of various factors such as the manipulation mode, the contact type (i.e., rolling, sliding, and twisting), friction, and finger chain kinematics constraints on the search process and on the structure of the resulting trajectories of the hand-object system.* Here, we report some preliminary simulation experiments for re-configuring convex smooth objects (e.g., spheres and ellipsoids) by a three-fingered hand.

For the examples shown in the following, we considered an ellipsoid-like object manipulated by 3 fingers in presence of friction. The mass of O is 1 Kg, and the considered friction coefficient is $\mu = 0.7$. Table 1 gives the discretization used by the planner.

We considered 2 different models of fingers. In the example shown in Figure 1, each finger is considered to be planar and has 3 degrees of freedom. The considered initial grasp and the final configuration of O are shown on the top left and the top right of the figure. In the middle, we show a sequence of grasps (from left to right) corresponding to the sub-goals achieved along the final trajectory. In the bottom of the figure, we show the final reached grasp and the contact trajectory[3] on the surface of O (shown by dark lines). 137 sub-goals have been generated in the search graph by the high planning level, and 17 among them have been processed by the local planner. The final trajectory goes through 10 of these sub-goals and the required run time was 343 seconds. This relatively high time illustrates the difficulty of finding steering motions of the fingertips satisfying the task constraints. We observed that the planar structure of the fingers is restrictive and reduces the range of relative sliding and rolling motions. The same task has also been processed considering a hand model composed of 3 polar fingers (i.e., 4 degrees of freedom) (see Figure 9). The run time has been decreased to 236 seconds, and 13 sub-goals (selected among 139 sub-goals generated by the high level) have been processed locally. 10 sub-goals lie on the final solution. Although not conclusive, this example shows that adding 1 degree of freedom has enlarged the range of feasible relative motions; thereby decreasing the run time of the planner.

ΔT	δt	δ_{xyz} (length cell)	δ_θ (length cell)
2.0 s	0.1 s	6 mm	2.406 deg.

v_O^{max}	ω_O^{max}	v_r^{max}	ω_r^{max}
2.5 mm/s	1.43 deg./s	2.5 mm/s	1.43 deg./s

Table 1: *Time and configuration discretization (top), and velocity discretization (bottom).*

Figure 10 shows another example solved by the planner to illustrate the application of `fix-some-fingers` mode to solve a re-orientation task performed by four 6-dof fingers (for simplicity, only the fingertips are shown). In this example, only 3 fingertips can be fixed instantaneously. At each instant, the moving finger is shown on the figure by the largest radius of its segment. 138 nodes were generated in the search graph, and 21 of them were processed locally. The run time for this

[3]Unfortunately, they are not clear on Figures 1 and 9.

task was 272 seconds. This time can be decreased by the two other modes. Further experiments are being carried out to demonstrate the full implementation of the planner, and to study issues such as what percentage of the manipulation problems can be solved by the planner, the complexity of tasks that can be solved by the planner, and the effect of discretization on the solution.

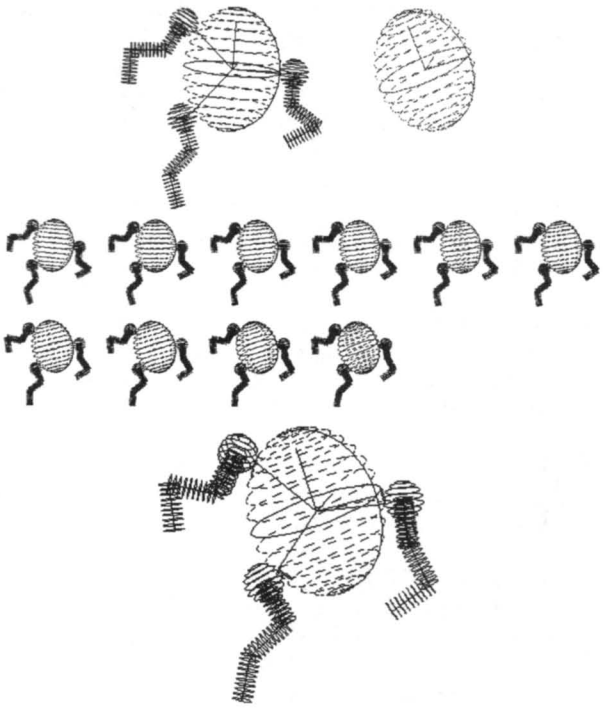


Figure 9: *Re-configuring an ellipsoid by three polar (4 dof) fingers.*

5 Discussion and Concluding Remarks

We described a global motion planner for *re-configuring* 3D objects with a multi-fingered hand. Our approach – a three-level planning scheme – deals with several aspects of dextrous manipulation across different levels: high dimensionality of the search spaces, frictional rolling and/or sliding constraints, kinematics of contact, collision avoidance, kinematics of the finger chains, equilibrium of grasps. The motion generation scheme is independent of the geometry of the interacting objects, so it can be applied to a large variety of

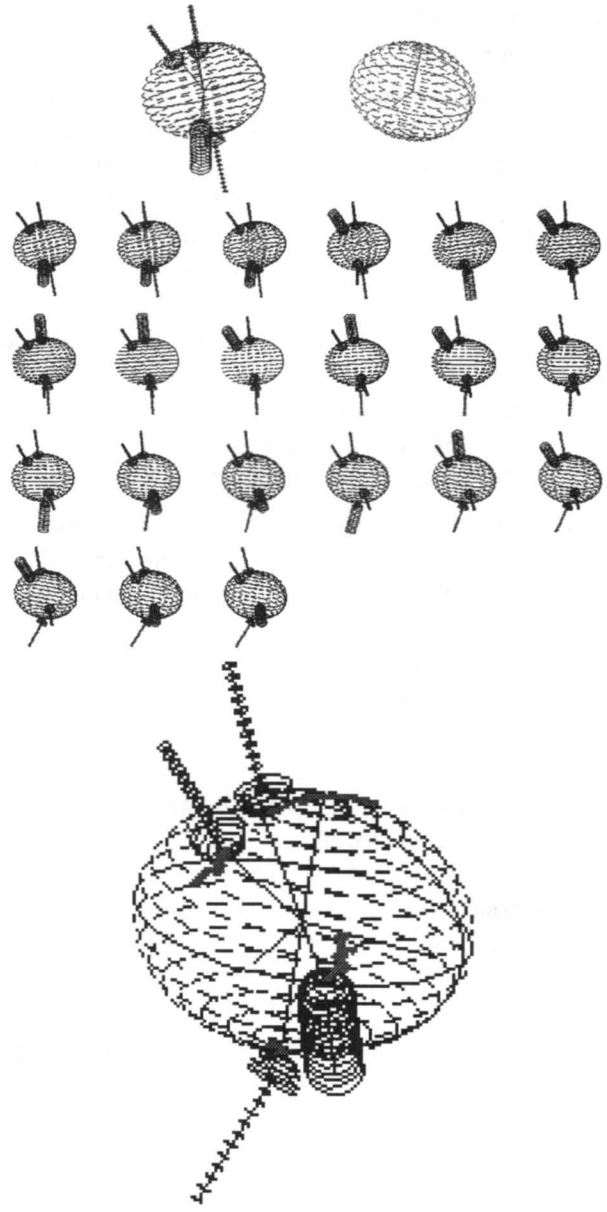


Figure 10: *Re-configuring an ellipsoid by 4 hemi-spherical fingertips. Top: the initial grasp and the desired configuration of O. Middle: the sequence of achieved sub-goals. Bottom: Final reached grasp with the contact trajectories. The gray curves show the trace of the contact trajectories on the object surface.*

rigid (piecewise-) smooth objects with invertible relative curvature (see [21]). To the best of our knowledge, the body of work described in this paper and in [4, 5, 6]

represents the first comprehensive global planner for 3D dextrous manipulation.

Our approach is conceptually extensible for another important manipulation mode not considered here, i.e., when some fingers are allowed to break contact with the object. For achieving this, we need to solve for new positions at which a given fingertip can contact the object, and to plan a collision-free motion for the considered finger to achieve that position (i.e., to re-locate the contact on the new position). Such a scheme can be used as an additional manipulation mode. The introduction of this re-location mode could significantly enlarge the range of achievable grasps (as mentioned by [17]).

We have also assumed that a low-level controller is able to control the relative rolling/sliding motions between the fingertips and the object. For enhancing the applicability of the resulting trajectories in a real context, we must incorporate uncertainty (for instance, a finger may not slide as planned). This could be done in two very different (but not necessarily mutually exclusive) ways. One would be to explicitly incorporate uncertainty models at the planning time. The other would be to impose additional conditions guaranteeing the admissibility of the trajectories to be planned and also tackling the indeterminacy problems when these trajectories are used by real mechanisms. For instance, first-order stability conditions that guarantee that rolling contacts can be maintained [31], and/or non-jamming conditions under which sliding contacts cannot switch to rolling [32] can be incorporated while planning. We believe that this second approach, at least in a conceptual sense, would be incorporated without any major modification to our planning framework. Of course, we need to characterize such conditions within the dextrous manipulation context. We plan to address these issues in our future work.

Acknowledgments

This research was partly supported by a grant from Natural Science and Engineering Research Council, Canada. The work of M. Cherif was partly done during a research appointment at INRIA Rhône-Alpes & GRAVIR-CNRS Lab., France (thanks to Christian Laugier who made this stay possible). We would like to thank the anonymous reviewers for their insightful comments.

References

[1] P. Bessière *et al.* Planning in a continuous space with forbidden regions : The ariadne's clew algorithm. In K. Goldberg, D. Halperin, J-C. Latombe, and R. Wilson, editors, *Algorithmic Fondations of Robotics*. A.K. Peters Publisher, Wellesley, MA (USA), 1995.

[2] A. Bicchi and R. Sorrentino. Dextrous manipulation through rolling. In *IEEE ICRA*, pages 452–457, Nagoya (J), May 1995.

[3] D. L. Brock. Enhancing the dexterity of a robot hand using controlled slip. In *IEEE ICRA*, pages 249–254, 1988.

[4] M. Cherif and K. K. Gupta. Planning quasi-static motions for re-configuring objects with a multi-fingered robotic hand. In *IEEE ICRA*, pages 986–991, Albuquerque (USA), April 1997.

[5] M. Cherif and K. K. Gupta. Practical motion planning for dextrous re-orientation of polyhedra. In *IEEE/RSJ IROS*, pages 291–297, Grenoble (F), September 1997.

[6] M. Cherif and K. K. Gupta. Planning quasi-static motions for re-configuring objects with a multi-fingered robotic hand. *Under revision for IEEE Trans. on Robotics & Automation*, 1998. (also available as Tech. Report CSS-IS TR 96-05, School of Engineering Science, Simon Fraser University, BC, Canada).

[7] N. Y. Chong *et al.* A generalized motion/force planning strategy for multifingered hands using both rolling and sliding contacts. In *IEEE/RSJ IROS*, pages 113–120, 1993.

[8] A. B. Cole, J. E. Hauser, and S. S. Sastry. Kinematics and control of multifingered hands with rolling contact. *IEEE Trans. on Automatic Control*, 34(4):398–404, 1989.

[9] A. B. Cole, P. Hsu, and S. S. Sastry. Dynamic control of sliding by robot hands for regrasping. *IEEE Trans. on Automatic Control*, 8(1):42–52, 1991.

[10] R. S. Fearing. Simplified grasping and manipulation with dextrous robot hands. Technical Report, A.I. Memo 809, A.I. Lab. MIT, MA, 1984.

[11] R. S. Fearing. Implementing a force strategy for object re-orientation. In *IEEE ICRA*, pages 96–101, San Francisco (USA), 1986.

[12] G. H. Golub and C. F. Van Loan. *Matrix computations.* The Johns Hopkins University Press, 1989.

[13] K. K. Gupta. Motion planning for re-orientation using finger tracking: Landmarks in $so(3) \times \omega$. In *IEEE ICRA*, pages 446–451, Nagoya (J), 1995.

[14] L. Han, Y.S. Guan, Z.X. Li, Q. Shi, and J.C. Trinkle. Dextrous manipulation with rolling contacts. In *IEEE ICRA*, pages 992–997, Albuquerque (USA), April 1997.

[15] J. Kerr and B. Roth. Analysis of multifingered hands. *IJRR*, 4(4):3–17, 1986.

[16] J-C. Latombe. *Robot motion planning.* Kluwer Academic Press, 1991.

[17] S. Leveroni and K. Salisbury. Reorienting objects with a robot hand using grasp gaits. In G. Giralt and G. Hirzinger, editors, *The 7th Int. Symposium on Robotics Research*, pages 2–15. Springer-Verlag, Munich (FRG), 1995.

[18] Z. Li and J. Canny. Motion of two rigid bodies with rolling constraint. *IEEE Trans. on Robotics & Automation*, 6(1):62–72, 1990.

[19] Z. Li, J. Canny, and S. S. Sastry. On motion planning for dextrous manipulation, part I: the problem formulation. In *IEEE ICRA*, pages 775–780, Scottsdale, AZ (USA), 1989.

[20] Z. Li, P. Hsu, and S. S. Sastry. Grasping and coordinated manipulation by a multifingered robot hand. *IJRR*, 8(4):33–50, August 1989.

[21] D. Montana. The kinematics of contact and grasp. *IJRR*, 7(3):17–31, 1988.

[22] R. M. Murray, Z. Li, and S. S. Sastry. *A Mathematical Introduction to Robotics Manipulation.* CRC Press, 1994.

[23] M. A. Peshkin and A. C. Sanderson. Minimization of energy in quasi-static manipulation. *IEEE Trans. on Robotics & Automation*, 5(1):53–60, 1989.

[24] J. Ponce *et al.* Algorithms for computing force-closure grasps of polyhedral objects. In K. Goldberg, D. Halperin, J-C. Latombe, and R. Wilson, editors, *Algorithmic Foundations of Robotics*, pages 167–184. A.K. Peters Publisher, Wellesley, MA (USA), 1995.

[25] D. Rus. *Fine Manipulation Planning for Dextrous Manipulation.* PhD thesis, Dep. of Computer Science, Cornell University, 1992.

[26] K. Salisbury. *Kinematics and Force Analysis of Articulated hands.* PhD thesis, CS Dept., Stanford University, 1982.

[27] K. B. Shimoga. Robot grasp synthesis algorithms: a survey. *IJRR*, 15(3):230–266, 1996.

[28] J. Tornero, G.J. Hamlin, and R.B. Kelley. Spherical-object representation and fast distance computation for robotic applications. In *IEEE ICRA*, pages 1602–1608, Sacramento (USA), May 1991.

[29] J. C. Trinkle. A quasi-static analysis of dextrerous manipulation with sliding and rolling contacts. In *IEEE ICRA*, pages 788–793, Scottsdale, AZ (USA), 1989.

[30] J. C. Trinkle and R. Paul. Planning dextrous manipulation with sliding contacts. *IJRR*, pages 24–48, 1990.

[31] J.C. Trinkle, A.O. Farahat, and P.F. Stiller. First-order stability cells of active multi-rigid-body systems. *IEEE Trans. on Robotics & Automation*, 11(4):545–557, 1995.

[32] J.C. Trinkle, S.L. Yeap, and L. Han. When quasistatic jamming is impossible. In *IEEE ICRA*, pages 3401–3406, Minneapolis (MN), April 1996.

[33] M. Zefran, J. P. Desai, and V. Kumar. Continuous motion plans for robotic systems with changing dynamic behaviour. In Jean-Paul Laumond and Mark Overmars, editors, *Algorithmic Fondations of Robotics*, pages 113–128. A. K. Peters Publisher, 1996.

Taylor & Francis
Taylor & Francis Group
http://taylorandfrancis.com

Shape Recovery from Passive Locally Dense Tactile Data

Michael Erdmann, *Carnegie Mellon University, Pittsburgh, PA, USA*

1 Introduction

As a robotic hand grasps an object of unknown shape, the object may slip in the robot's fingers. If the robot can infer the local shape as the object slips, the robot may be able to readjust its hand and fingers so as to better grasp the object.

This paper considers the problem of inferring local contact geometry from passive tactile information. The term "passive" means that the information results from motions of the object that are not necessarily under the control of the robot. To juxtapose, generally the term "active sensing" means that a robot actively explores an object, for instance with visual and/or tactile sensors that move around the object. This paper is not concerned with such active exploration. Instead, our goal is to determine how local shape geometry may be inferred from purely passive information. Shape recovery based on passive information will be useful both for dealing with unexpected events, as in the slippery grasping example above, and for more elaborate manipulation strategies, such as active exploration.

1.1 Motivation and context

This paper concentrates on planar objects manipulated by flat linear fingers, which we refer to as "palms". The motivation for this setup comes from our earlier work on nonprehensile palmar manipulation [Erdmann 1995] (see Figure 1). In that work, two robots manipulated objects by rolling and sliding the objects between linear palms. The objects' shape, mass properties, and friction were known in advance. Thus an automatic planning system could produce open-loop (sensorless) strategies for manipulating the objects from given start configurations to desired goal configurations.

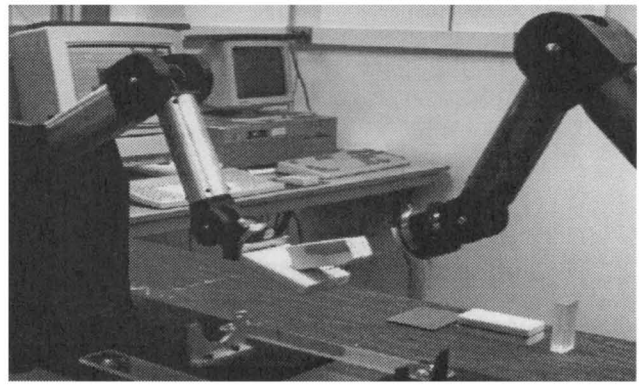


Figure 1: *Two robots, with flat palms attached to each wrist, manipulate an irregular block.*

Sensorless palms provide an existence proof of the basic manipulation strategies one can expect a robot to perform. Of particular interest to us previously was the ability to effectively turn friction on or off at a contact, by reorienting a palm. The ability to control friction allowed the robots to reposition their hands via sliding without effectively releasing the object, that is, without dropping or reorienting the object.

In a natural setting one cannot expect to know the object shape or friction coefficients precisely, or even expect the coefficients of friction to be constant over time. Instead, one might endow the palms with tactile sensors and hope to infer geometric and frictional characteristics of the object from the tactile sensors. For instance, contact location sensors might be used to infer the local object geometry and nibbed sensors might be used to detect impending slip.

In this paper we will focus on acquiring local geometry from contact location sensors. Specifically, we

envision a linear palm[1] endowed with a tactile sensor that can report the location of any contacts with the palm. We note in passing that this is a fairly traditional view of a tactile sensor; many more elaborate sensors have been studied and built. The main issue for us is density of information. We will assume that our sensor can report contact locations to infinite precision. In practice, even achieving sub-millimeter precision at reasonable cost is beyond the state of the art today.

1.2 Basic results

1.2.1 Assumptions

We consider planar objects in contact with linear tactile sensors, called "palms". Our assumptions are:

1. The objects are locally convex at the contacts. This assumption ensures that contact is possible locally with the palms.
2. A given object makes contact with a given palm at no more than one location. This is a natural extension of the local convexity assumption. Strictly convex shapes satisfy this assumption.
3. The objects are locally smooth. In other words, at each contact, the object's outward normal is anti-parallel to the palm's outward normal.
4. Each sensor reports the location of contact on the sensor with infinite precision.
5. The palms maintain contact passively. By this we mean that the palms do not move in an exploratory fashion; at most they move compliantly to maintain contact with the object.

1.2.2 Results

Our basic question is:

Given a set of linear tactile sensors in passive contact with an object, can we recover the local shape of the object at the contacts?

The answer has three parts:

[1]In 2D, a linear palm is just a straight line segment; in 3D, it would be a flat surface.

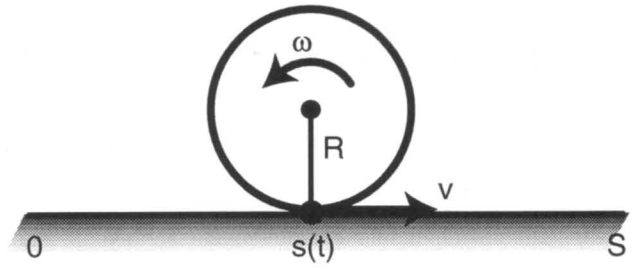


Figure 2: *Relationship between local object curvature, object motion, and sensory data.*

1. A single tactile sensor is generally unsatisfactory. We need to know too much to infer the local contact shape.
2. Two tactile sensors are satisfactory, given the angular rate at which the object is turning. In other words, if we know the angular velocity of the object, then we can recover its local geometry.
3. Three tactile sensors are generally sufficient to recover both the turning rate of the object and its local geometry. There are some non-generic exceptions, such as objects of constant local breadth.

1.2.3 Discussion of the Results

The negative result for a single sensor is not surprising. What is interesting, are the different remedies.

Figure 2 shows a circle of radius R on a flat horizontal palm. The circle maintains contact with the palm. The location of the contact point at time t is $s(t)$; its time derivative is $\dot{s}(t)$. The instantaneous contact point on the circle translates with horizontal velocity v; the circle rotates with angular velocity ω. The relationship between these terms is:

$$\dot{s} = v - \omega R. \tag{1}$$

This equation applies to any smooth object with instantaneous radius of curvature R at the contact point. The equation is a special case of the contact equations found in [MacMillan 1936], [Cai and Roth 1986], and [Montana 1988].

The problem of local shape recovery is quite simply: Determine the radius of curvature R. If R can be determined for each point along some curve segment, then that curve segment can be fully reconstructed. This is a consequence of the Frenet formulas for planar curves (see, for instance, [O'Neill 1997]).

The problem thus is to infer R from $s(t)$, using Equation (1). Unfortunately, Equation (1) is only one equation in four variables. We can sense $s(t)$ and thus compute $\dot{s}(t)$, but the velocity terms v and ω cause difficulty.

If v and ω are known, and if ω is non-zero, then one can easily solve for R. Otherwise, there is insufficient information. One positive instance occurs when one knows that the object is rolling without slipping and one also knows the object's angular rotation rate. In that case, $v = 0$, and thus R is just $-\dot{s}/\omega$.

Much of the work on active tactile sensing is motivated by this last positive observation. Rather than allow the object to roll and slide over a stationary sensor, one instead moves the sensor around the object (usually the object is held stationary). Thus one has complete control over ω and v; in particular, one can rotate the sensor on the object without slip and thus calculate the object's radius of curvature, much as we just did.[2]

The advantage of this active approach is knowledge by control of ω and v. The disadvantage is that one may not actually be able to achieve the desired ω and v, since the shape of the object is unknown. Thus the sensor may slip, even though one may be trying to prevent slip in order to enforce $v = 0$. These issues aside, even if control over ω and v is perfect, one is still faced with the more general problem of how to infer R from $s(t)$ when the object being manipulated can move on its own.

Our approach in this paper is to add additional sensory palms. Each such palm provides a further constraint on the variables in Equation (1). For instance, when we add a second palm, the velocities ω and v can no longer be completely arbitrary. Instead, they must satisfy the constraint that contact be maintained with both palms. Of course, we do not know exactly what that constraint is, since we also do not know the shape of the object at the second palm. However, the tactile information at the second palm provides a second equation like Equation (1). Intuitively speaking, we have thus gone from one equation and three unknowns to two equations and three unknowns. We can think of these three unknowns as the radii of curvature at the two contact points and the angular velocity of the part. Thus if we somehow know the angular velocity of the part, then we can recover the local geometries at the contacts.

Finally, when we add a third, compliant, palm, we can further constrain the unknown ω. The compliance in the third palm means that we do not actually get quite as strong a constraint as we did when we added the second palm. However, if we keep track of the location of the third palm relative to the first two as it moves, then we do have essentially an identical constraint. Thus, with some exceptions, we can recover the local contact geometries.

1.2.4 Discussion of the Assumptions

As will become clear, the assumptions are not terribly important, but they simplify the development. With regard to assumption (1), one would generalize to nonconvex objects by using nonlinear sensors. With regard to assumption (2), multiple contacts with a single palm actually provide more information than does a single contact. The smoothness assumption (3) in some sense makes shape inference more difficult. For instance, suppose we have two contacts with two palms, that is, one contact with each palm. Suppose further that we *know* one of the contacts to consist of an object vertex and a palm. Then the local geometry of the unknown part at the second contact with the second palm can be recovered much more easily than it can when both contacts are locally smooth.

Assumption (4) has the standard drawbacks. Finally, here are three comments regarding the "passivity" assumption (5):

[2] Usually one uses a circular tactile sensor rather than a linear one, and a more general version of Equation (1), but the underlying idea is the same.

First, our main reason for the passivity assumption is to make clear that we will only be considering information that comes directly from contact with the tactile sensors, as opposed to any knowledge of the sensors' absolute motion in space. In an actual manipulation example all palms might be moving in space, while manipulating an object. Our results still hold in that case. However, to tie things down a bit, let us elaborate the passivity assumption with the following specific requirements:

1. If the sensing apparatus consists of a single palm, then that palm does not move at all. The object is free to rotate and slide in any manner it chooses, while maintaining contact with the palm. Relative to the palm, the object has two degrees of freedom.

2. If the sensing apparatus consists of two palms, then the two palms do not move at all. Again, the object is free to rotate and slide in any manner it chooses, while maintaining contact with both palms. Relative to the two palms, the object has a single degree of freedom.

3. If the sensing apparatus consists of three or more palms, then two palms are stationary while the remaining palms move compliantly, so as to maintain contact with the object. Again, relative to the two stationary palms, the object has one degree of freedom.

Second, we could insist that the sensors always remain completely stationary, even when three or more palms are involved, but this would be pointless. With three or more contacts, generically, the object would be immobilized. We are interested in the cases in which the object does slide or slip, and hope to infer information from such motions.

Third, we tacitly assume that the object does indeed maintain contact with the sensors. In practice, there would probably be a mix of times involving contact and times in which contact is broken. We are here concerned with those, possibly small, intervals of time in which contact exists continuously.

2 Related Work

Prior work related to this paper falls roughly into six categories: contact kinematics, algorithmic shape recovery, local shape estimation, global shape and/or pose inference, active sensing, and tactile sensor technologies.

2.1 Contact kinematics

Much of the research on shape recovery points back to the works of Cai and Roth and Montana on contact kinematics. Four relevant papers are: [Cai and Roth 1986], [Cai and Roth 1987], [Montana 1988], [Montana 1995]. These papers contain kinematic analyses for fairly arbitrary shapes in contact with each other. The papers derive relationships between the local motions of the contact points on the shapes and the motions of the shapes relative to each other in world coordinates. We cited the simplest of these in the Introduction.

The resulting kinematic equations may be used to infer unknown geometric properties from observed geometric properties. For instance, [Montana 1988] proposes a strategy for determining the local curvature of a shape by repeatedly rolling a sensor, such as a cylindrical or spherical tactile sensor, over the shape. Much of the subsequent work on curvature estimation has taken this idea as its starting point.

We feel that the basic results of the current paper are implicit in the derivations of [Cai and Roth 1986], even if the tools are different. Towards the end of that paper, Cai and Roth consider the problem of manipulating a known object with two fingers. In particular, they propose a numerical method for computing the pose of an object, along with its angular velocity and angular acceleration, from the contact information sensed at the fingers. One difference is that the work by Cai and Roth assumes the object shape to be known.

2.2 Algorithmic shape recovery

Closely related to the techniques in this paper is the paper by [Rao and Goldberg 1994], which assesses the information contained in the *diameter function* of an object. [Sinden 1985] discusses related information provided by chords inscribed in an object. The diameter

function of a planar object is the distance between the jaws of a parallel-jaw gripper grasping the object, expressed as a function of the relative orientation between the gripper and the object.

The diameter function $D(\theta)$ of an object is closely related to its *radius function* $r(\theta)$. In order to define the radius function, let us first define some auxiliary terms. By a *supporting line* we mean a line touching the object, with the object wholly on one side or other of the line. By an *oriented supporting line at angle* θ, denoted $\ell(\theta)$, we mean a supporting line with an *exterior* side and an *interior* side. The normal to this line is given by the direction vector $(\cos\theta, \sin\theta)$; this normal vector points towards the exterior side of the line; the object of interest must lie wholly on the other side, that is, on the interior side of the line or on the line itself.

The radius function measures the distance from some fixed interior point of an object to its oriented supporting lines. Specifically, $r(\theta)$ is the distance from this fixed point to the oriented supporting line $\ell(\theta)$. Note that $D(\theta) = r(\theta) + r(\theta + \pi)$ regardless of the choice of fixed interior point.

It is well-known that the shape of a convex object with piecewise smooth boundary is uniquely determined by its radius function, computed relative to any fixed interior point. See also [Santaló 1976]. A natural question to ask is whether the diameter function of an object contains as much information. For general shapes, the answer is no, since there are objects of constant diameter other than circles [Reuleuax 1876]. [Rao and Goldberg 1994] observe that even when the class of objects is restricted to be polygonal, infinitely many objects can share the same diameter function. They then consider the problem of distinguishing between a known finite set of polygonal objects based on sparse diameter information. In a similar spirit, [Akella 1996] uses the radius function to distinguish between a finite set of polygonal shapes. In both cases, the objective is to find a discriminating tree of minimal height. A node in this tree represents a single sampling of the diameter or radius function. These techniques are useful in pose estimation during parts orienting.

In this paper we will be using a generalization of the radius function [Santaló 1976]. Since we do not know the object shape or its motion *a priori*, we cannot choose a fixed interior point. Thus we cannot compute the radius function. However, we do have access to the vector between the two contact points, and from this we will build a generalized two-dimensional support function. This will serve us as well as would a radius function, permitting us to reconstruct the object shape. This two-dimensional support function is a generalization of the radius function in the following sense: Suppose that one of the contacts involves a vertex of the object rather than a smooth curve. Then the (yet to be defined) 2D generalized support function will contain precisely the same information as would a radius function constructed by using that vertex as the fixed interior point.

Additional related algorithmic methods include the work on shape recovery from probing and projection operations. [Cole and Yap 1987] investigate the minimum number of *finger probes* required to reconstruct the shape of an unknown convex polygon. A finger probe entails choosing a ray in space, then moving along this ray, and recording the location at which the ray encounters the unknown object. [Boissonnat and Yvinec 1992] extend this basic finger probe to return not only the contact location but also the contact normal; they give algorithms for reconstructing a class of nonconvex polygons and polyhedra from such extended finger probes.

Another type of probe is the *line probe*. A line probe consists of choosing a line normal, then sweeping the corresponding line parallel to its normal, and recording the location of the line at which it touches the unknown object. In terms of our previous notation, a line probe entails choosing an angle θ, then determining the oriented supporting line $\ell(\theta)$ for the object. Generally, the sensor returns the location of the line in space, but not the location of the contact point of the object on the line. For a survey of algorithmic probing problems and results see [Skiena 1989].

Both finger and line probes constitute types of active tactile sensing. In each case, a robot actively moves a

finger or palm around an object, touching it at various locations.

[Lindenbaum and Bruckstein 1992] use parallel line probes and parallel finger probes to reconstruct convex polygons; they derive upper and lower bounds on the required number of such parallel probes. Their line probes are generalized line probes, i.e., lines that rotate around a specific point until they make contact with the unknown object. [Li 1988] considers line probes and projection probes to reconstruct the shape of unknown convex polygons. A *projection probe* is simply a pair of line probes that approach the unknown object from opposite directions; effectively the probe "projects" the object onto the common perpendicular of the two line probes. [Lindenbaum and Bruckstein 1994] give algorithms for the approximate reconstruction of arbitrary convex planar shapes using line probes. That work is interesting in particular because it moves away from the restricted class of polygonal shapes. Finally, [Kölzow, Kuba, and Volčič 1989] reconstruct convex shapes from sets of cross-sections.

2.3 Local shape estimation

As we indicated earlier, the rigid body contact kinematics and applications in [Montana 1988] inspired many researchers to investigate methods for estimating local curvature (and thus local shape). We will mention a few here. [Fearing and Binford 1988] use a cylindrical tactile sensor to determine principal curvatures, normal force, and contact location from strain measurements, i.e., from a theory of elasticity. [Chen, Rink, and Zhang 1996] describe an unknown surface locally in terms of a general quadratic form, then fit experimental tactile information to this form using active exploration and least-squares. [Charlebois, Gupta, and Payandeh 1996, 1997] describe two different methods for curvature estimation. The first entails rolling a sensor over the unknown surface, *à la* Montana; the paper analyzes the sensitivity of this strategy to sensor noise. The second entails dragging a set of fingers over the surface, then fitting the resulting contact data to a B-spline description of the surface patch. Any given patch is used to guide the exploration further. Along a different vein, [Kaneko, Higashimori, and Tsuji 1997] describe a strategy for estimating local surface shape for nonconvex objects, by pulling probes out of concavities.

One issue in curvature estimation is sensitivity. There is reason to be cautious. [Charlebois, Gupta, and Payandeh 1996] investigate the sensitivity of curvature estimates based on rolling to the accuracy with which the turning velocity is known, the distance rolled, and the tactile sensor resolution. One insidious problem is the interaction between the sensor resolution and the distance rolled. Rolling a larger distance will overcome low sensor resolution, but will lead to increased inaccuracies in the curvature estimate for objects of nonconstant curvature. [Nicolson and Fearing 1995] examine the sensitivity of curvature estimates based on elastic sensor models. [Zhang, Maekawa, and Tanie 1996] perform a sensitivity analysis and observe that rolling motions may be inadequate for determining curvature when the object's curvature is much larger or smaller than that of the sensor. The discussion at the end of that paper, comparing elastic models to rigid body models, is interesting as well.

2.4 Global shape and/or pose inference

We have already indicated that techniques used for unknown shape recovery may also be used to distinguish between different shapes taken from some restricted class of possible shapes. Merely restricting the class of possible shapes to polygons or polyhedra is often a profitable strategy. In industrial settings, the class may be much smaller yet. For instance, in parts orienting, one may simply need to distinguish between the different possible poses of a single part. In some cases, this set of possible poses may even be finite. Examples include the stable resting configurations of a polygonal part under gravity or in the grasp of a frictionless parallel jaw gripper. We have already pointed to [Rao and Goldberg 1994] and [Akella 1996] in this context. Related work includes the grasping strategies of [Goldberg, Mason, and Erdmann 1991] and [Goldberg 1993].

The work on pose and model recognition from tactile information has its origins in [Gaston and Lozano-Pérez 1984] and [Grimson and Lozano-Pérez 1984].

These papers build interpretation trees that use sparse tactile data to distinguish between different models and poses of 3D objects known to lie in some class of possible models. [Fearing 1990] discusses shape recognition of a certain class of generalized cylinders from sparse tactile information. [Caselli, Magnanini, Zanichelli, and Caraffi 1996] and [Beccari, Caselli, and Zanichelli 1997] describe techniques for recognizing convex polyhedra from sparse tactile data, using volumetric representations and active tactile exploration. [Siegel 1991] describes an interpretation-tree method for determining the pose of an object grasped by a robotic hand.

Another type of planar pose recognition strategy entails sampling a finite set of locations in the plane to distinguish between different possible objects and poses. [Jia and Erdmann 1996a] and [Romanik and Salzberg 1992] discuss this problem from an algorithmic perspective, examine its complexity, and present optimal strategies and simulations. [Jia and Erdmann 1996a] also consider the problem of distinguishing object poses by inscribing the objects in cones. A cone may be thought of as a pair of generalized line probes, each rotating around the same point. The paper shows experimentally that generally two cone inscriptions are sufficient to determine an object's pose.

The pose recognition strategies discussed above have two characteristics: (i) sensing is sparse and (ii) the object is fixed in space. [Jia and Erdmann 1996b] consider a problem on the other end of the spectrum. Specifically, the hand moves and purposefully disturbs the object. As a result, the contact point on the hand traces out some curve as a function of time. An assumed high-resolution tactile sensor embedded in the hand records this curve. The robot then employs techniques from nonlinear observer theory to recover the object pose from the tactile information and knowledge of the hand/object dynamics.

2.5 Active sensing

There has been considerable work on active sensing. We will only mention a few papers here. A seminal paper is [Allen and Michelman 1990]. The paper discusses an implemented system, consisting of a Utah-MIT hand mounted on a PUMA arm, with Interlink tactile sensors attached to the fingertips. The system performs exploratory operations such as enveloping grasps to recover shape, fitting the resultant tactile data to a superquadric description of the unknown shape. The paper also presents several other exploratory procedures, such as contour following. [Bay 1989] proposes an active exploration method, in which tactile data from multiple fingers is used to estimate a surface equation in a Kalman-filtering style. Exploration consists of rotating the hand, while using active force compliance to maintain contact between the fingertips and the unknown surface.

A number of researchers have considered the problem of planning exploratory strategies for obtaining useful tactile information. Both [Roberts 1990] and [Ellis 1992] present strategies that take account of the geometric constraints implied by the tactile information acquired thus far. Their methods determine the "best next" sensory operation based on these geometric constraints. Ellis considers sensory operations analogous to finger probes, while Roberts considers surface tracing moves. In recent work [Okamura, Turner, and Cutkosky 1997] describe a system with two types of fingers: (i) fingers that manipulate the object, and (ii) fingers with embedded tactile sensors that roll and slide over the object surface. Exploration is divided into phases. During each phase some subset of the fingers manipulate the object, while another subset traces over the object's surfaces. These second fingers record sensory information as they move, and they move to locations from which they can profitably act as the manipulating fingers during the next phase. The actual motions during a phase may be determined by the information gained in the previous phase.

2.6 Tactile sensor technologies

We point to [Howe and Cutkosky 1992] for an excellent survey of tactile sensor technologies. Some interesting papers that have appeared since that article include [Bicchi, Salisbury, and Brock 1993] on estimating contact locations from force measurements, [Howe and Cutkosky 1993] on sensing fine surface features with

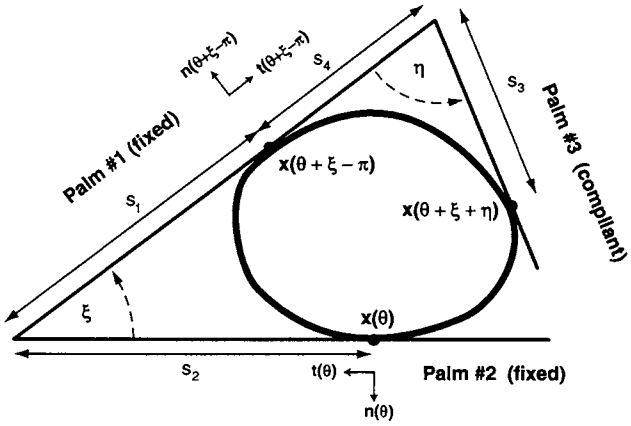


Figure 3: *Three sensors (two fixed, one compliant) in arbitrary configurations can usually recover the shape and rotation rate of the object.* $\mathbf{x}(\theta)$ *is a boundary parameterization of the object in its body coordinates. The object is shown at orientation* $-\theta - \pi/2$ *in hand/world coordinates.*

dynamic tactile sensors, and [Tremblay and Cutkosky 1993] on sensing incipient slip and estimating friction.

3 Three Arbitrarily-Positioned Linear Sensors

The full version of this paper contains a more detailed development leading up to this section. To find the full version of the paper, please follow the links underneath the author's homepage at

`http://www.cs.cmu.edu/~me`

Figure 3 shows the setup for three arbitrarily positioned tactile sensors. We assume that two of the sensors, namely those labeled "Palm 1" and "Palm 2" are fixed, while one sensor, labeled "Palm 3", complies to the motion of the object. Although, for the sake of simplicity, we have drawn the sensors as if they connect, this is not necessary. All that is required is three linear sensors at the contact points, lying on lines as drawn in the figure; the sensors themselves need not have particularly large extent.

We assume that the robot knows the relative configurations of the palms to each other, but does not know the object shape. Our goal is to recover the object shape locally around its contact with Palm 1 (of course, we can also then recover the shape at the other two contacts).

First, we will assume contact only with two sensors, namely Palms 1 and 2. We will show how the information from these sensors is sufficient to recover the local shape of the object, assuming the rotation rate of the object is known. Second, we will show how all three sensors may be used to ascertain the rotation rate of the object.

We assume that our unknown shape has a smooth strictly convex boundary. We parameterize that boundary curve in body coordinates by normal angle θ, written as $\mathbf{x}(\theta)$. We note that the results generalize to piecewise smooth boundaries, and even, with appropriately modified sensors, to nonconvex boundaries.

3.1 Shape from two sensors

3.1.1 Contact Support Function

First, let us define a variation of the well-known radius function [Santaló 1976], in order to account for the relative orientation of two sensors. In Figure 3, Palms 1 and 2 have relative orientation ξ. We assume that $0 < \xi < \pi$. We denote the contact point with the horizontal sensor, Palm 2, by $\mathbf{x}(\theta)$. Then the contact with Palm 1 must be $\mathbf{x}(\theta + \xi - \pi)$. We now define a *generalized support function* for such relative orientations, by the rules:

$$\tilde{r}_\xi(\theta) = [\mathbf{x}(\theta) - \mathbf{x}(\theta + \xi - \pi)] \cdot \mathbf{n}(\theta), \quad (2)$$

$$\tilde{d}_\xi(\theta) = [\mathbf{x}(\theta) - \mathbf{x}(\theta + \xi - \pi)] \cdot \mathbf{t}(\theta), \quad (3)$$

where $\mathbf{n}(\theta) = (\cos\theta, \sin\theta)$ and $\mathbf{t}(\theta) = (\sin\theta, -\cos\theta)$ in body coordinates.

In other words, suppose we look at the vector pointing from the contact point with Palm 1 to the contact point with Palm 2. $\tilde{r}_\xi(\theta)$ is the projection of this vector onto the outward normal $\mathbf{n}(\theta)$ at Palm 2, while $\tilde{d}_\xi(\theta)$ is the projection of this vector onto the tangent $\mathbf{t}(\theta)$.

We will also find it useful to look at the velocity of the $\mathbf{x}(\theta)$ parameterization of the boundary curve. Specifically, let us write in body coordinates

$$\mathbf{x}'(\theta) = -v(\theta)\,\mathbf{t}(\theta). \tag{4}$$

The θ-parameterization velocity $v(\theta)$ is also the radius of curvature of the shape boundary at the point $\mathbf{x}(\theta)$. (This follows, for instance, from the Frenet formulas [O'Neill 1997]).

Differentiating Equations (2) and (3) with respect to θ, we obtain a most useful pair of relationships:

$$\begin{aligned} \tilde{r}'_\xi(\theta) + \tilde{d}_\xi(\theta) &= -v(\theta+\xi-\pi)\,\sin\xi, && (5)\\ \tilde{d}'_\xi(\theta) - \tilde{r}_\xi(\theta) &= -v(\theta) - v(\theta+\xi-\pi)\cos\xi. && (6) \end{aligned}$$

3.1.2 Shape Recovered

Combining Equations (5) and (6) we obtain an explicit expression for the velocity $v(\theta)$ of the curve (or equivalently, the radius of curvature) at the contact point $\mathbf{x}(\theta)$:

$$v(\theta) = \left[\tilde{r}_\xi(\theta) - \tilde{d}'_\xi(\theta)\right] + \cot\xi\,\left[\tilde{r}'_\xi(\theta) + \tilde{d}_\xi(\theta)\right]. \tag{7}$$

We can put this into a familiar form by defining $(\hat{r}, \hat{d})$ as follows:

$$\begin{aligned} \hat{r}(\theta) &= \tilde{r}_\xi(\theta) + \cot\xi\,\tilde{d}_\xi(\theta),\\ \hat{d}(\theta) &= \tilde{d}_\xi(\theta) - \cot\xi\,\tilde{r}_\xi(\theta). \end{aligned}$$

Then

$$\hat{d}'(\theta) - \hat{r}(\theta) = -v(\theta). \tag{8}$$

Combining Equations (4) and (8) and integrating, we obtain

$$\boxed{\mathbf{x}(\theta) = \hat{d}(\theta)\,\mathbf{t}(\theta) - \int \mathbf{c}(\theta)\mathbf{A}(\theta)d\theta,} \tag{9}$$

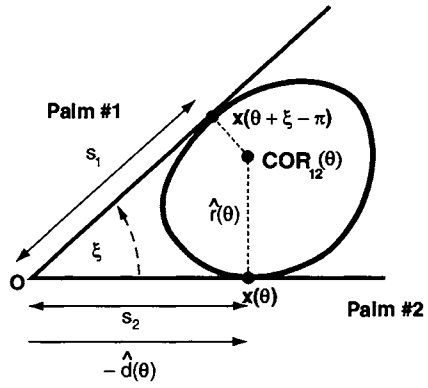


Figure 4: *Another support function expresses the location of the center of rotation in local hand coordinates.*

where all vectors are to be thought of as row vectors. $\mathbf{c}(\theta) = (\hat{r}(\theta), \hat{d}(\theta))$, and $\mathbf{A}(\theta)$ is the orthogonal matrix

$$\mathbf{A}(\theta) = \begin{pmatrix} \sin\theta & -\cos\theta \\ \cos\theta & \sin\theta \end{pmatrix}.$$

In short, Equation (9) is our shape recovered.

3.1.3 Geometric Interpretation of $(\hat{r}, \hat{d})$

The terms $\hat{r}(\theta)$ and $\hat{d}(\theta)$ have a simple geometric significance, as shown in Figure 4. In particular, they are easily computable from the observed sensor data s_1 and s_2.

Let $\mathbf{COR}_{12}(\theta)$ denote the instantaneous rotation center of the object while in contact with Palms 1 and 2 at the given orientation. $\hat{r}(\theta)$ is the distance of this rotation center from the horizontal sensor, Palm 2. And $-\hat{d}(\theta)$ is the tangential distance of the rotation center from the intersection of the two sensors, that is, from the point $\mathbf{O}$.

3.2 Determining the rotation rate

We have seen how to recover the object shape from two sensors, when the problem is cast in terms of θ. This is equivalent to assuming that the object rotation rate is known. We now add a third palm, as in Figure 3, in order to determine the object rotation rate.

We now have two generalized support functions with which to work, namely $(\tilde{r}_\xi, \tilde{d}_\xi)$ and $(\tilde{r}_\eta, \tilde{d}_\eta)$. Here ξ is

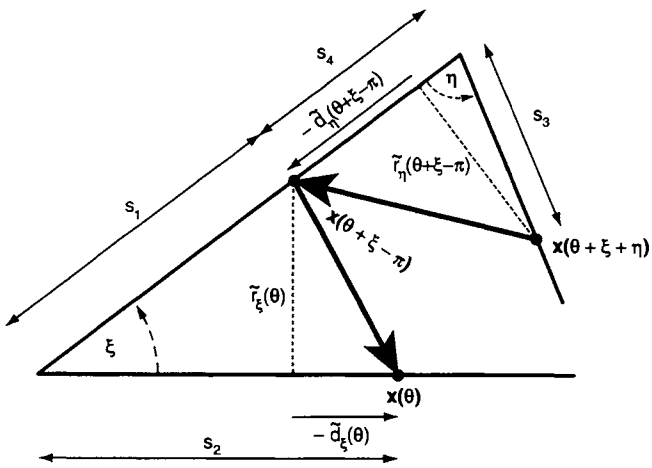


Figure 5: *Relationship between sensor data and contact support functions.*

the relative orientation between Palms 1 and 2, while η is the relative orientation between Palms 3 and 1. As before, we assume $0 < \xi, \eta < \pi$. Note that the contact point with Palm 3 is $\mathbf{x}(\theta + \xi + \eta)$.

We now have two expressions for $v(\theta + \xi - \pi)$. The first comes from sensor data at Palms 1 and 2, and is simply Equation (5) rewritten:

$$v(\theta + \xi - \pi) = -\frac{\tilde{r}'_\xi(\theta) + \tilde{d}_\xi(\theta)}{\sin \xi}. \tag{10}$$

The second comes from sensor data at Palms 3 and 1, and is Equation (7) rewritten in terms of $(\tilde{r}_\eta, \tilde{d}_\eta)$ and recentered at $\theta + \xi - \pi$:

$$\begin{aligned} v(\theta + \xi - \pi) = & \\ & \left[\tilde{r}_\eta(\theta + \xi - \pi) - \tilde{d}'_\eta(\theta + \xi - \pi)\right] \\ + & \cot \eta \left[\tilde{r}'_\eta(\theta + \xi - \pi) + \tilde{d}_\eta(\theta + \xi - \pi)\right]. \end{aligned} \tag{11}$$

Now, let us determine the relationship between the four sensed values $\{s_1(t), s_2(t), s_3(t), s_4(t)\}$ and the four support values $\{\tilde{r}_\xi(\theta), \tilde{d}_\xi(\theta), \tilde{r}_\eta(\theta + \xi - \pi), \tilde{d}_\eta(\theta + \xi - \pi)\}$. We assume that θ is an unknown function of time t, written as $\theta(t)$. Figure 5 is a recasting of Figure 3 with the object hidden but some extra labels shown. From this figure we see that:

$$\begin{aligned} s_1(t) \sin \xi &= \tilde{r}_\xi(\theta(t)), && (12) \\ s_1(t) \cos \xi - s_2(t) &= \tilde{d}_\xi(\theta(t)), && (13) \\ s_3(t) \sin \eta &= \tilde{r}_\eta(\theta(t) + \xi - \pi), && (14) \\ s_3(t) \cos \eta - s_4(t) &= \tilde{d}_\eta(\theta(t) + \xi - \pi). && (15) \end{aligned}$$

Differentiating with respect to time we obtain:

$$\begin{aligned} \dot{s}_1(t) \sin \xi &= \tilde{r}'_\xi(\theta(t))\, \omega(t), && (16) \\ \dot{s}_1(t) \cos \xi - \dot{s}_2(t) &= \tilde{d}'_\xi(\theta(t))\, \omega(t), && (17) \\ \dot{s}_3(t) \sin \eta &= \tilde{r}'_\eta(\theta(t) + \xi - \pi)\, \omega(t), && (18) \\ \dot{s}_3(t) \cos \eta - \dot{s}_4(t) &= \tilde{d}'_\eta(\theta(t) + \xi - \pi)\, \omega(t). && (19) \end{aligned}$$

Plugging these expressions into Equations (10) and (11), and multiplying by $\omega(t)$, yields the following two equations:

$$v(\theta(t) + \xi - \pi)\omega(t) = -\dot{s}_1(t) - \omega(t)\, \frac{s_1(t) \cos \xi - s_2(t)}{\sin \xi}, \tag{20}$$

$$v(\theta(t) + \xi - \pi)\omega(t) = \dot{s}_4(t) + \omega(t)\, \frac{s_3(t) - s_4(t) \cos \eta}{\sin \eta}. \tag{21}$$

Equating the right hand sides of (20) and (21), we finally arrive at:

$$\begin{aligned} \omega(t) \left[\frac{s_2(t) - s_1(t) \cos \xi}{\sin \xi} - \frac{s_3(t) - s_4(t) \cos \eta}{\sin \eta}\right] & \\ = \dot{s}_1(t) + \dot{s}_4(t). & \end{aligned} \tag{22}$$

Equation (22) allows us to solve for $\omega(t)$, and thus, by integration, for $\theta(t)$. If we now write $\mathbf{x}(t) = \mathbf{x}(\theta(t))$, then we see that

$$\dot{\mathbf{x}}(t) \;=\; -v(\theta(t))\,\omega(t)\,\mathbf{t}(\theta(t)).$$

Let us define $\dot{s}(t) = v(\theta(t))\omega(t)$. Then, using Equations (7) and (12)–(19), we see that

$$\dot{s}(t) = \frac{s_1(t) - s_2(t)\,\cos\xi}{\sin\xi}\,\omega(t) + \dot{s}_2(t)$$

Integrating, we obtain the following expression for $\mathbf{x}(t)$:

$$\boxed{\mathbf{x}(t) \;=\; -\int \dot{s}(t)\,\mathbf{t}(\theta(t))dt,} \qquad (23)$$

where $\mathbf{t}(\theta(t)) = (\sin(\theta(t)), -\cos(\theta(t)))$ is the contact tangent vector at time t, in body coordinates. Here we have parameterized $\mathbf{x}$ as a function of time rather than angle; in either case, the shape is explicitly recovered. Equation (23) is our shape recovered, as a function of the set of four sensor signals $\{s_1(t), s_2(t), s_3(t), s_4(t)\}$.

3.3 Exceptions

Finally, we should comment on the exceptions. Equation (22) only allows us to solve for $\omega(t)$ if the term

$$\left[\frac{s_2(t) - s_1(t)\,\cos\xi}{\sin\xi} - \frac{s_3(t) - s_4(t)\,\cos\eta}{\sin\eta}\right] \qquad (24)$$

is non-zero. This term actually represents the distance between two instantaneous rotation centers of the object, namely $\mathbf{COR}_{12}$ and $\mathbf{COR}_{13}$. Here $\mathbf{COR}_{12}$ is the instantaneous rotation center due to contact with Palms 1 and 2, just as before. Similarly, $\mathbf{COR}_{13}$ is the instantaneous rotation center due to contact with Palms 1 and 3.

Equation (22) now makes a lot of sense. It is telling us that the separation between Palms 2 and 3, as measured along Palm 1, is changing at a rate that is equal to the product of the rotation rate of the object and the distance between the two centers of rotation.

In particular, if the two centers of rotation coincide, then Palm 3 does not need to move compliantly as the object rotates. Conversely, if Palm 3 does not need to move compliantly, then either the object is not rotating or the two centers of rotation $\mathbf{COR}_{12}$ and $\mathbf{COR}_{13}$ coincide.

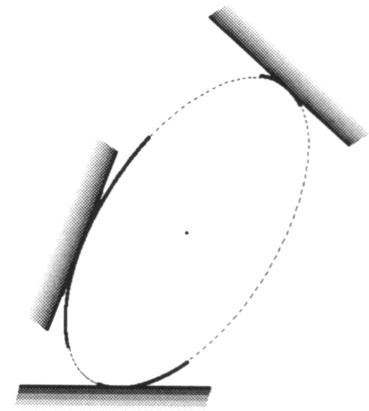


Figure 6: *Three sensors touch a moving ellipse and recover its shape locally.*

We can think of $s_1(t)+s_4(t)$ as a *generalized diameter function.* It is giving us the separation between two skew jaws of a gripper grasping our unknown object at the points $\mathbf{x}(\theta)$ and $\mathbf{x}(\theta + \xi + \eta)$, measured along a particular line in space. If the object locally has constant "diameter", as measured by this function, then we cannot recover the object's rotation rate, at least not with the given arrangement of the palms.

3.4 An example

Figure 6 shows three tactile sensors touching an ellipse at an instant in time. The ellipse is rotating clockwise with constant angular acceleration. Two of the sensors are fixed, the third is compliant. The true ellipse appears as a thin dashed outline. The local shape recovered at the contacts appears as three thick curves. The ellipse rotated for 1 second total time. The data consisted of 1000 toplevel sample points. However, the internal integration routines required additional data at intermediate points, effectively multiplying by four the actual sample values required. Thus the sensors must deliver data at 4000 Hz, with infinite precision.

The match obtained is fairly reasonable, although one can see slight discrepancies at the end of the sampling period. The maximum error was approximately 2.7%, occurring near the end of the sampling period. Here 100% corresponds to the length of the minor half-axis of the ellipse.

Acknowledgments

Many thanks to Yan-Bin Jia and Matt Mason for inspiration and engaging discussions about tactile sensing. Financial support for this research was provided in part by Carnegie Mellon University and in part by the National Science Foundation through the following grants: NSF Presidential Young Investigator award IRI-9157643 and NSF Grant IRI-9503648.

References

[1] Akella, S. 1996. Robotic Manipulation for Parts Transfer and Orienting: Mechanics, Planning, and Shape Uncertainty. CMU-RI-TR-96-38. Ph.D. thesis. The Robotics Institute, Carnegie Mellon University.

[2] Allen, P. K., and Michelman, P. 1990. Acquisition and Interpretation of 3-D Sensor Data from Touch. *IEEE Transactions on Robotics and Automation.* **6**(4):397–404.

[3] Bay, J. S. 1989. Tactile Shape Sensing via Single- and Multi-Fingered Hands. *Proceedings of the 1989 IEEE International Conference on Robotics and Automation*, Scottsdale, Arizona, Vol 1, pp. 290–295.

[4] Beccari, G., Caselli, S., and Zanichelli. F. 1997. Pose-Independent Recognition of Convex Objects from Sparse Tactile Data. *Proceedings of the 1997 IEEE International Conference on Robotics and Automation*, Albuquerque, New Mexico, pp. 3397–3402.

[5] Bicchi, A., Salisbury, J. K., and Brock, D. L. 1993. Contact Sensing from Force Measurements. *International Journal of Robotics Research.* **12**(3):249–262.

[6] Boissonnat, J. D., and Yvinec, M. 1992. Probing a Scene of Nonconvex Polyhedra. *Algorithmica.* **8**:321–342.

[7] Cai, C. S, and Roth, B. 1986. On the Planar Motion of Rigid Bodies with Point Contact. *Mechanism and Machine Theory.* **21**(6):453–466.

[8] Cai, C. S, and Roth, B. 1987. On the Spatial Motion of a Rigid Body with Point Contact. *Proceedings of the 1987 IEEE International Conference on Robotics and Automation*, Raleigh, North Carolina, pp. 686–695.

[9] Caselli, S., Magnanini, C., Zanichelli, F., and Caraffi, E. 1996. Efficient Exploration and Recognition of Convex Objects Based on Haptic Perception. *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*, Minneapolis, Minnesota, pp. 3508–3513.

[10] Charlebois, M., Gupta, K., and Payandeh, S. 1996. Curvature Based Shape Estimation Using Tactile Sensing. *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*, Minneapolis, Minnesota, pp. 3502–3507.

[11] Charlebois, M., Gupta, K., and Payandeh, S. 1997. Shape Description of General, Curved Surfaces Using Tactile Sensing and Surface Normal Information. *Proceedings of the 1997 IEEE International Conference on Robotics and Automation*, Albuquerque, New Mexico, pp. 2819–2824.

[12] Chen, N., Rink, R., and Zhang, H. 1996. Local Object Shape from Tactile Sensing. *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*, Minneapolis, Minnesota, pp. 3496–3501.

[13] Cole, R., and Yap, C. K. 1987. Shape from Probing. *Journal of Algorithms.* **8**(1):19–38.

[14] Ellis, R. E. 1992. Planning Tactile Recognition Paths in Two and Three Dimensions. *International Journal of Robotics Research.* **11**(2):87–111.

[15] Erdmann, M. A. 1995. An Exploration of Nonprehensile Two-Palm Manipulation: Planning and Execution. *Seventh International Symposium on Robotics Research*, October 21–24, 1995, Herrsching, Germany, pp. 16–27.

[16] Fearing, R. S. 1990. Tactile Sensing for Shape Interpretation. In *Dexterous Robot Hands,* Venkataraman, S .T., and Iberall, T., eds. Berlin: Springer-Verlag, pp. 209–238.

[17] Fearing, R. S., and Binford, T. O. 1988. Using a Cylindrical Tactile Sensor for Determining Curvature. *Proceedings of the 1988 IEEE International Conference on Robotics and Automation*, Philadelphia, Pennsylvania, pp. 765–771.

[18] Gaston, P. C., and Lozano-Pérez, T. 1984. Tactile Recognition and Localization Using Object Models:

The Case of Polyhedra on a Plane. *IEEE Transactions on Pattern Analysis and Machine Intelligence.* **PAMI-6**(3):257–266.

[19] Goldberg, K. 1993. Orienting Polygonal Parts Without Sensors. *Algorithmica.* **10**(2,3,4):201–225.

[20] Goldberg, K., Mason, M., and Erdmann, M. 1991. Generating Stochastic Plans for a Programmable Parts Feeder. *Proceedings of the 1991 IEEE International Conference on Robotics and Automation*, Sacramento, California, pp. 352-359.

[21] Grimson, W. E. L., and Lozano-Pérez, T. 1984. Model-Based Recognition and Localization from Sparse Range or Tactile Data. *International Journal of Robotics Research.* **3**(3):3–35.

[22] Howe, R. D., and Cutkosky, M. R. 1992. Touch Sensing for Robotic Manipulation and Recognition. In *The Robotics Review 2*, eds. Khatib, O., Craig, J., and Lozano-Pérez, T., Cambridge, MA: MIT Press.

[23] Howe, R. D., and Cutkosky, M. R. 1993. Dynamic Tactile Sensing: Perception of Fine Surface Features with Stress Rate Sensing. *IEEE Transactions on Robotics and Automation.* **9**(2):140–151.

[24] Jia, Y.-B., and Erdmann, M. A. 1996a. Geometric Sensing of Known Planar Shapes. *International Journal of Robotics Research.* **15**(4):365–392.

[25] Jia, Y.-B., and Erdmann, M. A. 1996b. Pose from Pushing. *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*, Minneapolis, Minnesota, pp. 165–171.

[26] Kaneko, M., Higashimori, M., and Tsuji, T. 1997. Pulling Motion Based Tactile Sensing for Concave Surface. *Proceedings of the 1997 IEEE International Conference on Robotics and Automation*, Albuquerque, New Mexico, pp. 2477–2484.

[27] Kölzow, D., Kuba, A., and Volčič, A. 1989. An Algorithm for Reconstructing Convex Bodies from Their Projections. *Discrete Computational Geometry.* **4**:205–237.

[28] Li, S.-Y. R. 1988. Reconstruction of Polygons from Projections. *Information Processing Letters.* **28**(5):235–240.

[29] Lindenbaum, M., and Bruckstein, A. 1992. Parallel Strategies for Geometric Probing. *Journal of Algorithms.* **13**(2):320–349.

[30] Lindenbaum, M., and Bruckstein, A. 1994. Blind Approximation of Planar Convex Sets. *IEEE Transactions on Robotics and Automation.* **10**(4):517–529.

[31] MacMillan, W. D. 1936. *Dynamics of Rigid Bodies.* New York: Dover Publications. (1960 reprint of a 1936 McGraw-Hill edition.)

[32] Montana, D. J. 1988. The Kinematics of Contact and Grasp. *International Journal of Robotics Research.* **7**(3):17–32.

[33] Montana, D. J. 1995. The Kinematics of Multi-Fingered Manipulation. *IEEE Transactions on Robotics and Automation.* **11**(4):491–503.

[34] Nicolson, E. J., and Fearing, R. S. 1995. The Reliability of Curvature Estimates from Linear Elastic Tactile Sensors. *Proceedings of the 1995 IEEE International Conference on Robotics and Automation*, Nagoya, Japan, Vol 1, pp. 1126–1133.

[35] Okamura, A. M., Turner, M. L., and Cutkosky, M. R. 1997. Haptic Exploration of Objects with Rolling and Sliding. *Proceedings of the 1997 IEEE International Conference on Robotics and Automation*, Albuquerque, New Mexico, pp. 2485–2490.

[36] O'Neill, B. 1997. *Elementary Differential Geometry.* Second Edition. New York: Academic Press.

[37] Rao, A. S., and Goldberg, K. Y. 1994. Shape from Diameter: Recognizing Polygonal Parts With a Parallel-Jaw Gripper. *International Journal of Robotics Research.* **13**(1):16–37.

[38] Reuleuax, F. 1876. *The Kinematics of Machinery.* New York: Dover Publications. (1963 reprint of an 1876 Macmillan and Company edition.)

[39] Roberts, K. S. 1990. Robot Active Touch Exploration: Constraints and Strategies. *Proceedings of the 1990 IEEE International Conference on Robotics and Automation*, Cincinnati, Ohio, Vol 2, pp. 980–985.

[40] Romanik, K., and Salzberg, S. 1992. Testing Orthogonal Shapes. *Proceedings of the Fourth Canadian Conference on Computational Geometry*, pp. 216–222.

[41] Santaló, L. A. 1976. *Integral Geometry and Geometric Probability.* Encyclopedia of Mathematics and its Applications. Volume 1. Reading, MA: Addison-Wesley.

[42] Siegel, D. M. 1991. Finding the Pose of an Object in a Hand. *Proceedings of the 1991 IEEE International Conference on Robotics and Automation*, Sacramento, California, pp. 406–411.

[43] Sinden, F. W. 1985. Shape Information from Rotated Scans. *IEEE Transactions on Pattern Analysis and Machine Intelligence.* **PAMI-7**(6):726–730.

[44] Skiena, S. S. 1989. Problems in Geometric Probing. *Algorithmica.* **4**(4):599–605.

[45] Tremblay, M. R., and Cutkosky, M. R. 1993. Estimating Friction Using Incipient Slip Sensing During a Manipulation Task. *Proceedings of the 1993 IEEE International Conference on Robotics and Automation*, Atlanta, Georgia, Vol 1, pp. 429–434.

[46] Zhang, H., Maekawa, H., and Tanie, K. 1996. Sensitivity Analysis and Experiments of Curvature Estimation Based on Rolling Contact. *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*, Minneapolis, Minnesota, pp. 3514–3519.

Geometric Algorithms in AdeptRAPID

John J. Craig, *SILMA Division of Adept Technology Inc., San Jose, CA, USA*

AdeptRAPID is a software package used to design and simulate robot-based flexible automation cells. Its goal is to simplify the concepting and design of automation installations used for applications such as mechanical assembly, material handling, or packaging.

The need for ease of use drives the need for the simulation system to behave like the actual physical world. The more the simulator acts like the real world, the simpler the user interface paradigm becomes for the user, since the physical world is the one we are all familiar with. At the same time, trade-offs for computational speed and other factors have driven a design in which a particular "slice" of reality is simulated, while many details are not.

AdeptRAPID is well-suited as a host for a variety of geometric algorithms. The need to model various portions of the real world, as well as the need to unburden the user by making various computations of a geometric nature, drive the need for such algorithms. AdeptRAPID provides the environment in which some advanced algorithms can be brought to bear on real problems occuring in industry.

This paper presents the current state of the AdeptRAPID simulator. Attention is paid to the various geometric algorithms already in place as well as the need for more and better algorithms.

1 Introduction

One decision made very early on in the design of the AdeptRAPID simulation system was that the *programming paradigm* should be as close as possible to the way the actual robot system would be programmed. While there are certain higher level planning and optimization tools, it was deemed important to have the basic programming interaction be similar to actual hardware systems. This decision has led the product's development down a path along which we find a genuine need for various geometric algorithms. The algorithms needed range widely from extremely simple to quite complex.

If a simulator is to be programmed like the physical system would be, then the actions and reactions of the physical world must be modelled "automatically" by the simulator. The goal is to free the user of the system from having to write any "simulation-specific code". As a simple example, if the robot gripper is commanded to open, a grasped part should fall due to gravity, and possibly bounce and settle into a certain stable state. Forcing the user of the system to specify these real-world actions would make the simulator fall short of its goal to be programmed just like the actual system. Ultimate ease of use can only be achieved when the simulated world "knows how" to behave like the real world without burdening the user.

Most, if not all, commercial systems for simulating robots or other mechanisms do not attempt to deal directly with this problem. Rather, they typically allow the user (that is, *force* the user) to embed simulation-specific commands within the program written to control the simulated device. A simple example would be the folowing code sequence:

```
MOVE TO pick_part
CLOSE gripper
affix(gripper,part[i]);
MOVE TO place_part
OPEN gripper
unaffix(gripper,part[i]);
```

Here, the user has been forced to insert the "affix" and "unaffix" commands which cause the part to move with the gripper when grasped, and to stop moving with it when released. If the simulator allows the robot to be programmed in its native language, then generally that language is not rich enough to support these required "simulation-specific" commands. Hence there is a need for a second set of commands, possibly even with a different syntax, for dealing with interactions with the real world. Such a scheme is inherently *not* programmed just like the physical system and must inherently have an increased programming burden for the user.

From the above example, we see the first geometric algorithm that one finds a need for: based on the geometry of the gripper and the relative placements of parts, determine which part (if any) will be grasped when the gripper closes and possibly how the part will self-align within the gripper. In the case of AdeptRAPID, we solve the first part of this problem with a simple algorithm. In limited cases, the "alignment action" of the part in the gripper is computed, but in general such alignments need to be pre-taught by the system's user. Hence, in our product development we have not reached the ultimate goal yet, but we have taken some steps in that direction. As the development continues, we plan to continually add algorithms and refine the current ones to increase ease of use of the system.

2 Physical Modelling and Interactive Systems

In a simulation system one always trades off complexity of the model in terms of computation time versus accuracy of the simulation. In the case of AdeptRAPID and its intended goals, it is particularly important to keep the system fully interactive. This has led us to design AdeptRAPID to use various approximate models, for example, the use of quasi-static approximations where a full dynamic model might be more accurate. Although there appears to be a possibility that "full dynamic" models might soon be applicable [1], given the current state of computer hardware, dynamic algorithms, and the complexity of the CAD models that industrial users wish to employ, we feel these trade offs still need to be made.

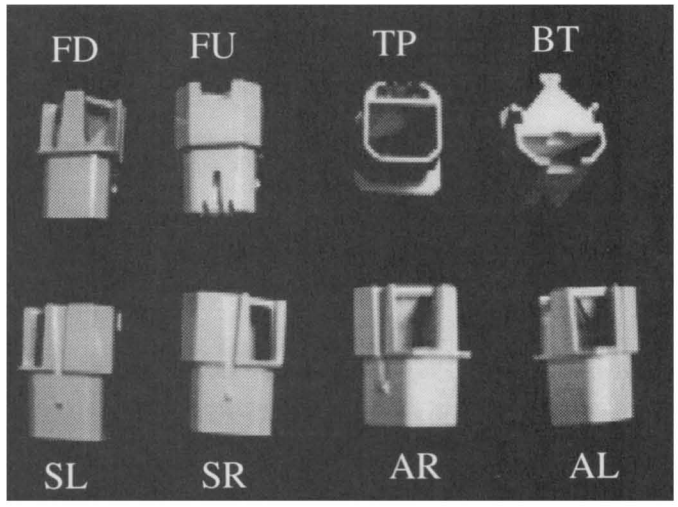


Figure 1: *The eight stable states of the part.*

3 Geometric Algorithms for Part Tumbling

In some feeding systems employed in industrial practice, parts tumble from some form of infeed conveyor onto a presentation surface. Then, computer vision is used to locate parts which can be acquired by the robot. Designing such automation systems with the aid of a simulator means that the simulator must be able to predict how parts fall, bounce and take on a stable orientation, or *stable state.*

3.1 Stable state probabilities

As reported elsewhere ([2]) we have implemented an algorithm which takes as input any geometric shape (represented by a CAD model) and for that shape can compute the N possible ways that it can rest stably on a horizontal surface. These are called stable states of the part. Further, the algorithm uses a perturbed quasi static approach to estimate the probability associated with each of the N stable states. We have performed physical experiments with sample parts in order to assess the resulting accuracy of stable state prediction.

Figure 2 shows the eight stable states of a particular test part. Using an Adept robot and vision system, we dropped this part over 26,000 times and recorded the resulting stable state in order to compare our stable state prediction algorithm to reality. Table 1 shows the results for this test part. These results are typical of our current algorithm – stable state liklihood prediction error may be 5% to 10%.

Table 1: *Predicted versus actual stable state probabilities for the test part.*

Stable State	Actual #	% Actual	% Predicted
FU	1,871	7.03%	8.91%
FD	10,600	39.80%	44.29%
TP	648	2.43%	7.42%
BT	33	0.12%	8.19%
SR	6,467	24.28%	15.90%
SL	6,583	24.72%	15.29%
AR/AL	428	1.61%	0.00%
Total	26,630	100%	100%

3.2 Adjusting probabilities as a function of drop height

Clearly if a part is dropped from a gripper from a very small height (e.g. 1 mm) above a surface, the probabilities of the various stable states differ from those if the part is dropped from higher than some *critical height.* In AdeptRAPID, we use probabilities from the stable pose estimator algorithm when parts are dropped from heights equal or greater than the largest dimension of the part. For drop heights below that value, probabilities are adjusted depending on the initial orientation of the part and the height of the drop. The adjustment is such that as an infinitesimal drop height is approached, the part remains in its current orientation (assuming it is a stable orientation). This is an important addition to the overall pose probability algorithm as it is typical that parts are released a small distance above a support surface.

3.3 Simulation of bounce

Parts in AdeptRAPID are tagged with their coefficient of restitution as are all surfaces on which parts may be placed. The product of these two factors is used in a formula for predicting how far the part will bounce when dropped. These details are important as they affect how parts scatter or clump in simulating some feeding systems. When bouncing, parts are scatterred radially using a uniform distribution. The distance of bounce (away from the initial contact point) is a certain distribution function out to a maximum distance which is computed as a function of drop height (energy input) and the coefficients of restitution at play.

Parts in AdeptRAPID can recursively bounce from surface to surface in certain arrangements. It is also possible to mark certain surfaces such that parts are not able to bounce *off* them, but only bounce *within* them. Entities known as *bins* in AdeptRAPID have this property – parts may fall into them, but they never bounce out.

3.4 Simulation of stacking and tangling

As a simplification, parts in AdeptRAPID always rest on planar support surfaces. If parts are tangled or stacked on one another, this is displayed as parts that are intersecting each other (that is, the boolean intersection of their volumes would be non-empty). This saves the enormous amount of computation that would be needed to compute the various ways a part might be stacked or tangled with another part's geometry.

Parts in AdeptRAPID are tagged with a *tangle factor.* For example, something like a marble would have a tangle factor of 0.0 because when tumbled onto a support surface, marbles tend never to stack or tangle, but rather tend to spread out on the surface. On the other hand, a part like a coiled spring might have a tangle factor approaching 1.0 since they quite readily become entangled with one another. When parts fall and bounce, a *findspace* algorithm runs in which the parts try to bounce into an open space on the surface. However, exactly how "hard they try" to find an open space is a factor of their tangle factor. By adjustment of this coefficient, AdeptRAPID can simulate

parts that tumble and become entangled more or less. Currently there is no algorithm for automatically computing the tangle factor from the part geometry – this is an interesting open problem. Through the user interface, the AdeptRAPID user can set the tangle factor as s/he feels appropriate.

4 Geometric Algorithms for Part Grasping

Much of the difficulty in programming and using actual robots has to do with the details of teaching grasp locations on parts, and the detailed design of grippers. This is an area in which additional planning algorithms in a simulator system could have a large impact. In this section we discuss the algorithms currently inplace in AdeptRAPID. The current approaches are quite simple, so this is an area of on-going work.

4.1 Computing which part to grasp

When a tool closes, or a suction end-effector actuates, AdeptRAPID applies a simple algorithm to compute which part (if any) should become grasped by the robot. First, the system determines which support surface is immediately beneath the gripper. Then, for all parts on that surface, search for those whose bounding box (for the current stable state) contain the TCP (tool center point) of the gripper. If more than one part satisfy this criteria, then choose the nearest among those that do.

4.2 Computation of default grasp location

AdeptRAPID automatically assigns a grasp location for each stable orientation predicted by the stable pose estimator described above. The current algorithm is simplistic, so a graphical user interface is also provided so that the user can edit and redefine these grasp points. The current grasp algorithm is a function of the part's bounding box and the geometry of the gripper, which is assumed to be either a parallel jaw gripper or a suction cup. Along with computing a default grasp location for each stable state, a default approach and depart height is also automatically computed.

We are now building an interactive grasp planner. Rather than try to automate the task entirely, we feel the user should indicate approximate grasp location, but let the system then make adjustments or fill in some of the details. We are also looking at letting the system automatically redesign the finger tips (or suction cup) of the end-effector as part of the grasp point design process.

4.3 Computation of alignment of the part during grasp

In some important cases in industrial practice, the system designer counts on the fact that when the robot end-effector actuates, the captured part will align itself in some way with surfaces of the end-effector. This can be important in removing small misalignments in the presentation of parts to the robot.

A very real effect which needs to be simulated is that with suction cup grippers, it may be the case that when suction is applied, the part is "lifted" up against the suction cup in a way which significantly alters its orientation relative to the end-effector. AdeptRAPID simulates this effect by piercing the part geometry with a vertical line aligned with the center line of the suction cup. Whichever facet of the polygonal part model is pierced is used in computing the orientation at grasp – the normal of this facet becomes anti-aligned with the normal of the bottom of the suction cup. In altering the part orientation, rotation about this peircing line is minimized (the part does not spin about the axis of the suction cup when picked). Without simulation of this effect, the simulator would be unable to realistically depict some pick and place strategies employing suction grippers.

We have also implemented a planner which allows parts to rotate about the Z axis when a parallel jaw gripper closes on them. This case is only automatic for a simple case – in other situations, the user must teach the resulting alignment (that is, we are still waiting for a more complete algorithm).

5 Geometric Algorithms for Part Pushing

One style of part-pushing occurs between the jaws of a gripper, as mentioned in the previous section. In current industrial practice, parts sometimes get pushed by simple mechanisms. For example, after a part is presented by a bowl feeder, it might be pushed by a linear actuator right into an assembly that has been brought into the cell by a tray conveyor system.

AdeptRAPID has recently added support for simulating the pushing of parts. We have introduced an entity called a *push-bar* which can be attched to a pneumatic cylinder or a leadscrew actuator in the simulator. When the actuator moves the push-bar along a linear path, the leading surface of the push-bar will move parts. In the future it is planned that push-bars will also be able to be added as guides along conveors, or placed anywhere that requires that parts motion be affected by their presence. The current pushing is still very simple, but it suffices for many real world tasks.

6 Geometric Algorithms for Tray Conveyors

AdeptRAPID supports the simulation of tray conveyor systems in which trays move along tracks composed of straight line and circular section components. Placed along the tracks at key locations may be *gates* which pop up to temporarily block a tray when so commanded. Additionally, *sensors* which detect a passing tray may be placed in the track a user-specified locations. Such conveyor systems are typical in many automation schemes.

6.1 Connecting tray conveyors and sources and sinks

Tray conveyors can be connected together to allow various styles of branching. Where two conveyors "flow together" a simple collision avoidance scheme is provided to cause trays from the *spur* conveyor to be subordinate to trays on the *main* conveyor. Trays on the spur conveyor will wait if necessary if otherwise a collision would occur. At "flow apart" connections, a device called a *director* is added to the main conveyor which can be used to control which direction a tray will take at the intersection. Digital I/O lines connected to the simulated robot controller are used to read sensors, activate gates, and activate directors.

At the ends of a tray conveyor are a *source* and a *sink*. Sources are set up by the user to generate trays at certain statistical intervals. The trays generated may be empty or may be pre-loaded with parts or fixtures. At the end of a tray conveyor, trays (and their contents) dissappear into sinks. Each time a tray enters a sink, its arrival time and contents are recorded. These so-called *sink records* can then be re-played through a source elsewhere in the system. Hence, a line of cells can be studied in the simulator one cell at a time, by setting the source of cell N+1 to the sink record from cell N.

6.2 Pushing of trays

Pushing is also implemented for trays: a push-bar can be used to push a tray off of a tray conveyor system and into a particular work cell. Likewise, trays can be pushed onto a tray conveyor. The updating of various data structures when trays come off of or onto a conveyor is an automatic part of the pushing code.

7 Geometric Algorithms for Sensors

Simulation of various sensor systems is required so that the user need not be burdened with writing code to emulate their behaviour in the cell.

7.1 Proximity sensors

AdeptRAPID supports the simulation of proximity sensors and other sensors. In the case of proximity sensors, the user tags the device with its minimum and maximum range, and a threshhold. If an object is within range and closer than the threshhold, then the sensor will detect it. To perform this computation in the simulated world, a line segment is temporarily added to the world which stretches from minimum to maximum sensor range. Using a collision algorithm, we compute the locations at which this line segment

intersects other CAD geometry. The intersection point nearest the sensor corresponds to the real world item that would have stopped the beam. A comparison of the distance to this point and the threshhold gives the output of the sensor. At present, we do not make use of the angle of the encountered surface or its reflectance properties, although those features might be added in the future.

7.2 2-D vision systems

AdeptRAPID simulates the performance of the Adept 2-D vision system. Elements of this system which are simulated are:

- The shape and extent of the field of view.
- The stand-off distance and a simple model of focus.
- The time required to perform vision processing (approximate).
- The spatial ordering of results in the queue in the case of many parts found in one image.
- The ability to distinguish parts according to which stable state they are in.
- The inability to recognize parts that are touching or overlapping.
- Within the context of AIM, the ability to update robot goals based on vision results.

Since the use of a vision system is well integrated with the AIM robot programming system, implementation of the AIM language in the simulator implies implementation of vision system emulation. AIM supports several constructs which make the use of vision easy for robot guidance. Picking parts which are identified visually from both indexing and tracking conveyors are easily accomplished.

A data structure maintains which support surface the vision system is looking at. For all parts supported on that surface, we compute which are within the vision system's field of view. We prune out any parts which are too near or too far from the camera (e.g. out of focus). We prune out any parts which are touching neighboring parts. From the remaining parts, we choose those which are in the sought-after stable state and put them in a list. Finally, this list is sorted to emulate the ordering the Adept vision system uses when multiple parts are found in one scene.

7.3 Inspector sensors

A special class of sensor is provided which is called an *inspector*. The inspector is used to give a binary output based on a part placed in front of it. Parts in AdeptRAPID can be tagged with a *defect rate* and inspectors can ferret out the defective parts. Inspectors play the role of several real world sensor systems.

8 Conclusion

As mentioned throughout this paper, although some simple geometric algorithms are currently in place in the simulator, there is a need for more and better algorithms. In particular, we would like to investigate the possibility of adding a quasi-static simulation capability for predicting the motion of objects in situations in which friction effects dominate any inertial effects. This could be used to simulate parts being pushed or tipped by various actions of end-effectors or other pushing mechanisms.

Acknowledgements

We thank Brian Carlisle and Bruce Shimano for their vision and belief in the AdeptRAPID product. We thank Ken Goldberg for his help in design and implementation of the "part tumbling" algorithms during a portion of his sabbatical spent at SILMA. We thank Lou Paul who is currently spending a portion of a sabbatical at SILMA and working on "part grasping" and finger-tip design problems. We hope to continue to improve the capabilities of the simulator, and look forward to continued interaction with the academic community in the area of geometric algorithms.

References

[1] B. Mirtich and J. Canny. Impulse Based Dynamic Simulation of Rigid Bodies, *Symposium on Interactive 3D Graphics*, New York, 1995. ACM Press.

[2] B. Mirtich, Y. Zhuang, K. Goldberg, et al. Estimating Pose Statistics for Robotic Part Feeders, *Proceedings of the IEEE Robotics and Automation Conference*, Minneapolis, Minnesota, April, 1996.

Taylor & Francis
Taylor & Francis Group
http://taylorandfrancis.com

On Finding Narrow Passages with Probabilistic Roadmap Planners

David Hsu, *Stanford University, Stanford, CA, USA*
Lydia E. Kavraki, *Rice University, Houston, TX, USA*
Jean-Claude Latombe, *Stanford University, Stanford, CA, USA*
Rajeev Motwani, *Stanford University, Stanford, CA, USA*
Stephen Sorkin, *Stanford University, Stanford, CA, USA*

A probabilistic roadmap is a network of simple paths connecting collision-free configurations obtained by sampling a robot's configuration space at random. Several probabilistic roadmap planners have solved unusually difficult path-planning problems, but their efficiency remains disappointing when the free space contains narrow passages. This paper provides foundations for understanding the effect of passages on the connectedness of probabilistic roadmaps. It also proposes a new random sampling scheme for finding such passages. An initial roadmap is built in a "dilated" free space allowing some penetration distance of the robot into the obstacles. This roadmap is then modified by resampling around the links that do not lie in the true free space. Experiments show that this strategy allows relatively small roadmaps to reliably capture the free space connectivity.

1 Introduction

Probabilistic roadmap planners (PRMs) construct a network of simple paths (usually straight paths in configuration space) connecting collision-free configurations picked at random [2, 4, 5, 10, 11, 13, 16, 24, 27]. They have been successful in solving difficult path-planning problems, but their efficiency remains disappointing when the free space contains narrow passages. In [4, 10, 14, 15], we have formally investigated the performance of PRMs. The property of ϵ-goodness [15] allows us to determine how well a probabilistic roadmap "covers" the free space, while the properties of expansiveness [10] and path clearance [14] allow us to analyze the connectedness of a roadmap.

Exploiting these results we now motivate a new sampling strategy to find paths through narrow passages more efficiently than with previous strategies. An initial roadmap R' is computed in a "dilated" free space F' allowing some penetration distance of the robot into the obstacles. Next, the milestones and links of R' that do not lie in the free space F are "pushed" into F by local resampling operations, in order to construct a final roadmap R. The underlying intuition is that, by widening narrow passages, dilatation improves ϵ-goodness, expansiveness, and path clearance, so that the connectivity of F' is easier to capture than that of F. Preliminary experiments have produced very encouraging results.

Path planners using techniques to expand or shrink obstacles and/or robots have previously been proposed [3, 9], but with different planning approaches.

Section 2 describes a "basic" PRM. Section 3 surveys other sampling strategies proposed in the literature. Section 4 establishes the relation between the results in [4, 10, 14, 15] and the complexity of the basic PRM in the presence of narrow passages. Section 5 describes and discusses the new sampling strategy.

2 Basic PRM

Let $\mathcal{C}$ denote the configuration space of a robot and F the open subset of collision-free configurations, *i.e.*, the *free space*. For simplification, let $\mathcal{C}$ be the Euclidean hyper-cube $[0,1]^n$. We say that two free configurations *see* each other if they can be connected by a straight-line path in F.

The basic PRM is a simplified version of the planner in [16]. First, the procedure `roadmap` precomputes

procedure `roadmap`:

1. Pick s configurations uniformly at random in F. Call them *milestones* and let M be the set of milestones.
2. Construct the graph $R = (M, L)$ in which L consists of every pair of milestones that see each other. Call R the *roadmap*.

Figure 1: *Roadmap-construction algorithm.*

a roadmap R; then `query` uses R to process path-planning queries. Each query is defined by two free configurations, q_b and q_e, called the *query configurations*. A correct answer is a path connecting them in F, if one such path exists, and the indication that no such path exists, otherwise.

The procedure `roadmap` (Figure 1) chooses the milestones at Step 1 and creates the links between milestones at Step 2. Let `dist`(q) be a procedure that computes the Euclidean distance between the robot placed at q and the obstacles. Step 1 generates each milestone by picking successive configurations q in $\mathcal{C}$, until one satisfies `dist`$(q) > 0$. Every q is obtained by choosing each of its coordinates uniformly at random in $[0, 1]$. Step 2 checks the straight path between every two milestones for collision, by recursively decomposing it into two half segments and invoking `dist` at each segment endpoint [4].

The procedure `query` (Figure 2) tries to connect each query configuration to a milestone of the roadmap, either directly (Step 1(a)), or through an intermediate configuration chosen in a neighborhood of the query configuration (Step 1(b)), using `dist` to check connections for collision. Each free configuration q at Step 1(b)i is obtained by picking successive configurations at random in a hyper-cube centered at q_i until one is collision-free. The procedure may output FAILURE at Step 1(b)ii (if it cannot connect a query configuration to some milestone of the roadmap) and NO-PATH at Step 2 (if it connects the query configurations to two distinct components of the roadmap). The NO-PATH answer may be incorrect if two components of the roadmap lie in the same component of F. Clearly,

procedure `query`:

1. **for** $i = \{b, e\}$ do:
   (a) **if there exists** a milestone m that sees q_i **then** $m_i \leftarrow m$.
   (b) **else**
      i. **repeat** t **times:**
         Pick a configuration q in F uniformly at random in a neighborhood of q_i
         until q sees both q_i and a milestone m.
      ii. **if all** t trials failed **then return** FAILURE, **else** $m_i \leftarrow m$.
2. **if** m_b and m_e are in the same component of the roadmap, **then return** a path connecting them, **else return** NO-PATH.

Figure 2: *Query processing algorithm.*

we would like the planner to rarely return FAILURE or an incorrect NO-PATH answer.

The cost of computing an explicit representation of F in a high-dimensional space is prohibitive. Instead, a PRM uses the implicit representation of F provided by `dist`, which only computes distances in $\mathbb{R}^2$ or $\mathbb{R}^3$ and admits several reasonably efficient implementations (*e.g.*, [7, 8, 12, 18, 19, 20, 22, 25]). In this paper we often use simple illustrative examples of 2-D free spaces, which could easily be handled by other planning techniques. Keep in mind that in practical problems, it is often not realistic to explicitly represent F.

3 Other Sampling Strategies

The planner in [5] uses a partially random sampling strategy. It deterministically follows the steepest descent of a heuristic potential field defined over $\mathcal{C}$ until it reaches a minimum of the potential. If this minimum is the goal, it stops; otherwise it tries to escape its basin of attraction by performing a series of random walks. Therefore, the potential field strongly biases the planner's sampling strategy. In some cases, it successfully guides the planner through narrow passages. But it may also have exactly the opposite effect, when a local basin of attraction contains a narrow passage.

The planner in [13, 16] includes a resampling step to improve the connectedness of the roadmap. An initial roadmap is generated using an algorithm similar to `roadmap`. Next, additional milestones are added in the neighborhoods of milestones that see no or few other milestones. Experiments have shown that this resampling scheme is very effective. But it still does not provide an efficient means of finding connections through narrow passages. Note that the rationale underlying this two-stage sampling strategy is similar to the one in this paper. In both cases, the first stage attempts to capture the connectivity of a free space using no a priori knowledge, while the second stage adds milestones in subsets of F chosen on the basis of the information revealed by the first stage.

We will see that the sampling strategy proposed in this paper creates a greater density of milestones near the boundary of the free space. Other roadmap planners try to attain this goal using different strategies. In [2], when a configuration q is generated outside F, a number of rays are shot from q along random directions uniformly distributed in $\mathcal{C}$. For each ray, a binary search is used to identify a point near the boundary of F. In [23] a single ray is shot from q along a random direction; the procedure then simulates a walk of the robot along this direction, until it reaches free space. In [11] new milestones are created near F's boundary to connect roadmap components that could not be connected by straight paths. A ray is shot from a milestone in one component along a direction picked at random. Using a technique similar to [23], a milestone is created where this ray encounters the free space boundary, and the ray is reflected in a random direction at this point to find another boundary point. All three references listed above observe that adding milestones near the free space boundary improves the planners' performance.

Various sampling and connection strategies are described and experimentally evaluated in [1]. In particular, a multi-stage strategy is proposed which allows milestones to be connected by multiple types of paths.

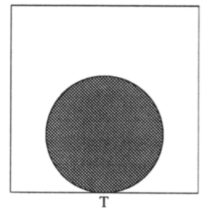


Figure 3: *A free space that is not ϵ-good.*

4 Complexity of the Basic PRM

To avoid the FAILURE outcome, a PRM must pick milestones that collectively see a large portion of F, so that any query configuration can easily be connected to one of them. To avoid incorrect NO-PATH replies, it must connect the milestones so that there is a one-to-one correspondence between the components of F and those of the roadmap.

In this section we analyze the complexity of the basic PRM by estimating the number s of milestones (the *size* of the roadmap) that must be generated in order to answer queries correctly with high probability. We relate this number to the extent to which F satisfies desirable geometric properties: ϵ-goodness [4, 15], expansiveness [10], and path clearance [13, 14]. Though not perfect, s is a reasonable measure of the work done by the planner.

For any subset $S \subseteq \mathcal{C}$, $\mu(S)$ denotes its volume. For any $q \in F$, $\mathcal{V}(q)$ denotes the set of all free configurations seen by q; we call it the *visibility set* of q.

4.1 Roadmap coverage

Definition 1. *Let ϵ be a constant in $(0,1]$. A free configuration q is* ϵ-good *if $\mu(\mathcal{V}(q)) \geq \epsilon\mu(F)$. The free space F is* ϵ-good *if every $q \in F$ is ϵ-good.*

There exist spaces that are not ϵ-good. This is the case of the 2-D free space shown in Figure 3. Every point q in this space is ϵ-good for some $\epsilon \in (0,1]$. However, if q tends toward the tangency point T, $\epsilon \to 0$.

Definition 2. *A set of milestones provides* adequate coverage *of an ϵ-good free space F if the volume of the subset of F not visible from any of these milestones is at most $(\epsilon/2)\mu(F)$.*

A greater ϵ will make it easier for `query` to connect query configurations to the roadmap. Hence, as ϵ increases, the coverage requirement grows weaker, *i.e.*, the portion of F that has to be visible by at least one milestone gets smaller.

Theorem 1. *Assume that F is ϵ-good. Let ϕ be a constant in $(0,1]$ and K be a positive real large enough that for any $x \in (0,1]$, $(1-x)^{(K/x)\log(2/x\phi)} \leq x\phi/2$. If s is chosen such that:*

$$s \geq \frac{K}{\epsilon}(\log\frac{1}{\epsilon} + \log\frac{2}{\phi}),$$

then `roadmap` *generates a set of milestones that adequately covers F, with probability at least $1-\phi$.*

See [4, 15] for a proof of this theorem.

Theorem 1 does not allow us to compute s since we do not know the value of ϵ, except for simple spaces. Nevertheless, it tells us that although adequate coverage of F is not guaranteed, the probability that inadequate coverages happens decreases exponentially with the number of milestones. Moreover, the number of milestones needed increases moderately when ϵ decreases.

It now remains to establish that adequate coverage allows `query` to connect any query configuration to the roadmap, with high probability.

Theorem 2. *Let the maximum number of iterations t at Step 1(b)i of* `query` *be set to $\log(2/\psi)$, where ψ is a constant in $(0,1]$. If the milestones adequately cover F, then the probability that* `query` *outputs* FAILURE *is at most ψ.*

In other words, the probability that `query` returns FAILURE decreases exponentially with the number t of iterations at Step 1(b)i. See [15] for a proof of this theorem. This proof assumes that Step 1(b)i of `query` samples the visibility set $\mathcal{V}(q_i)$ of q_i to find a configuration q that sees both q_i and a milestone m. Since $\mathcal{V}(q_i)$ is unknown, any implementation of `query` is an approximation of the algorithm to which Theorem 2 strictly applies.

However, ϵ-goodness is too weak a requirement to guarantee that `roadmap` will construct a roadmap whose connectivity correctly represents that of the free space. For example, the free space of Figure 4 is ϵ-good for $\epsilon \approx 0.5$. But a roadmap constructed by `roadmap` will most likely consist of two connected components. This type of example leads us to introduce the notion of an expansive free space [10].

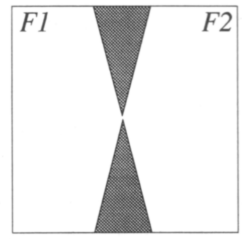


Figure 4: *A narrow passage in an ϵ-good space.*

4.2 Roadmap connectedness

Definition 3. *Let F be an ϵ-good free space. A roadmap R is an* adequate representation *of F if its milestones provide adequate coverage of F and no two components of R lie in the same component of F.*

Let R be an adequate representation of F. Since F is ϵ-good, no component of F has volume less than $\epsilon\mu(F)$. Therefore, at least one milestone of R lies in every component of F. Since no two components of R lie in the same component of F, there is a one-to-one correspondence between the components of R and those of F.

The notion of an expansive free space is directly related to the difficulty that `roadmap` has to connect milestones through narrow passages. The reason why it would require considerable time for `roadmap` to build a connected roadmap in the free space of Figure 4 is that a very small subset of points in F_1 see a large fraction of F_2; therefore, the probability that the planner picks a milestone in F_1 that sees a milestone in F_2 is small. By narrowing the passage between F_1 and F_2, one can make this probability arbitrarily small. We refer to the subset of points in a subset $S \subset F$ that can see a large portion of $F \backslash S$ as the lookout of S. If it is large enough, it is easy to connect any point in S to points outside S by picking points at random in S and $F \backslash S$.

Definition 4. *Let β be a constant in $(0,1]$ and S be any subset of any component E of the free space F.*

*The β-*lookout *of S is the set:*

$$\beta\text{-LOOKOUT}(S) = \{q \in S \mid \mu(\mathcal{V}(q)\backslash S) \geq \beta\mu(E\backslash S)\}.$$

Definition 5. *Let ϵ, α, and β be constants in $(0,1]$. The free space F is (ϵ,α,β)-*expansive *if it is ϵ-good and, for every connected subset S, we have:*

$$\mu(\beta\text{-LOOKOUT}(S)) \geq \alpha\mu(S).$$

We abbreviate "(ϵ,α,β)-expansive" by "expansive".

Theorem 3. *Assume that F is (ϵ,α,β)-expansive. Let ξ be a constant in $(0,1]$. If s is chosen such that:*

$$s \geq \frac{16}{\epsilon\alpha}\log\frac{8}{\epsilon\alpha\xi} + \frac{6}{\beta} + 2,$$

then with probability at least $1-\xi$, `roadmap` *generates a roadmap such that no two of its components lie in the same component of F.*

Given a set M of milestones, let a *linking sequence* of a milestone m_0 be a sequence $m_1, m_2, \ldots$, such that for all $i > 1$, $m_i \in$ LOOKOUT(V_{i-1}), with $V_0 = \mathcal{V}(m_0)$ and $V_i = V_{i-1} \cup \mathcal{V}(m_i)$. Let the visibility set of a linking sequence be the union of the visibility sets $\mathcal{V}(m_i)$ of all the milestones m_i in the sequence. The proof of Theorem 3, given in [10], first establishes that in a properly sized roadmap, for every pair of milestones (m, m') in the same component of F, with high probability m admits a linking sequence whose visibility set intersects that of a linking sequence of m'; it then shows that with high probability there exists a milestone which lies in the intersection of the visibility sets of these two linking sequences, so that m and m' are in the same component of the roadmap.

Theorems 1 and 3 imply that with high probability, `roadmap` generates a roadmap that adequately represents F. Theorem 3 tells us that the probability that a roadmap does not capture F's connectivity decreases exponentially with the number of milestones, and the number of milestones needed increases moderately when ϵ, α, and β decrease.

Theorems 1, 2, and 3 do not explicitly mention the dimension n of $\mathcal{C}$. This comes from the fact that the

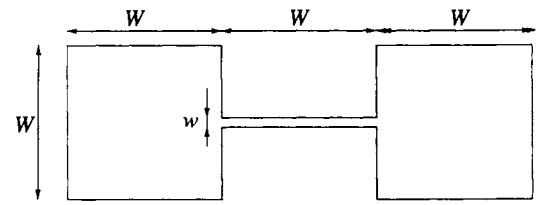


Figure 5: *An expansive free space where $\epsilon, \alpha, \beta \sim w/W$.*

definitions of ϵ-goodness and expansiveness only refer to volumes of subsets of F. But the dependence on n may be hidden in the parameters ϵ, α, and β. Consider the example of Figure 5. The free space consists of two squares whose sides have length W; these cubes are connected by a narrow passage of length W and width w, with $w \ll W$. Up to a constant factor, each of the parameters ϵ, α, and β is on the order of w/W. The points with the smallest ϵ-goodness are located in the narrow passage. Each such point sees only a subset of F of volume approximately $3wW$; hence, $\epsilon \sim w/W$. A point near the top right corner of the left square sees this entire square; but only a subset of this square, of approximate volume wW, contains points that, each, see a set of volume $2wW$; hence, $\alpha \sim w/W$ and $\beta \approx w/W$. In the n-D version of this example, two hypercubes, each having volume W^n, are connected by a hyper-parallelepipedic passage that has size w along k dimensions ($k \in [1, n-1]$) and size W along the $n-k$ other dimensions. Each of the parameters ϵ, α, and β is on the order of $(w/W)^k$. The worst case happens when $k = n-1$, that is, when the passage is narrow along $n-1$ dimensions.

4.3 Path-clearance assumption

Another analysis of the basic PRM can be done by explicitly considering the "width" of the passages in F [13, 14]. Let q and q' be two configurations in the same component of F and τ be a free path connecting them. Let ℓ be the Euclidean length of τ and σ its distance to F's boundary. We call σ the *clearance* of the path.

Theorem 4. *Let ζ be a constant in $(0,1]$ and a be the constant $2^{-n}\mu(\mathcal{B}_1)/\mu(F)$ where $\mathcal{B}_1$ denotes the unit ball*

in $\mathbb{R}^n$. *If* s *is chosen such that:*

$$s \geq \frac{1}{a\sigma^n} \log \frac{2\ell}{\sigma\zeta},$$

then with probability at least $1 - \zeta$ `roadmap` *generates a roadmap in which one component contains two milestones* m *and* m' *such that* q *sees* m *and* q' *sees* m'.

This theorem is proven in [14]. Consistently with Theorem 3, it says that the probability that a roadmap fails to provide a path through a narrow passage decreases exponentially with the number of milestones. It also rightly suggests that the number of milestones may increase as $(2/\sigma)^n$. However, it is more conservative than Theorem 3. For instance, in the n-D version of the example shown in Figure 5, $\sigma = w/2$, even if the passage is narrow along only one dimension. While Theorem 4 suggests that the number of milestones needed increases as w^{-n}, Theorem 3 tells us that it only increases as w^{-k}, where $k < n$ is the number of dimensions along which the passage is narrow.

A variant of the path-clearance assumption that may yield a slightly tighter bound than Theorem 4 is the σ-complexity assumption proposed in [27].

5 Finding Narrow Passages

We first describe a new sampling strategy for finding narrow passages. Then we show experimental results with a simplified implementation of this strategy. The computation of penetration distances is studied in the next subsection. Finally, we discuss the new strategy in the context of the results of Section 4.

5.1 New algorithms

The new roadmap-construction algorithm, `new-roadmap`, starts by constructing a roadmap R' in a dilated free space F' obtained by allowing some penetration distance δ of the robot into the obstacles. Next, it "pushes" the milestones and links of R' that do not lie in F into F by performing local resampling operations. The outcome is a roadmap R in F. The rationale behind this two-stage strategy is that free space dilatation widens narrow passages and thus makes it easier to capture free space connectivity. Once constructed, R' provides pertinent information about which areas of F should be more densely sampled.

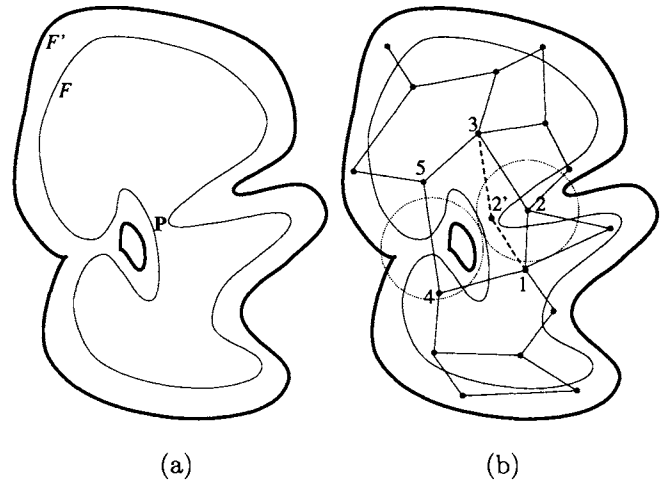


Figure 6: *Roadmap construction steps.*

The operations of `new-roadmap` are illustrated in Figure 6. Figure 6(a) depicts the free space F (thin contour) and the dilated free space F' (bold contour). Note the narrow passage **P** in F. This passage is wider in F', but a spurious passage has been created on the left. Figure 6(b) displays a roadmap R' (only a subset of the links are shown) and illustrates resampling operations. The procedure `new-roadmap` resamples a neigborhood of 2, which is not in F, until it finds a new milestone $2'$ in F through which it can connect the milestones 1 and 3. Similarly, `new-roadmap` resamples a neigborhood of the link between milestones 4 and 5, but this operation eventually fails to find a connection. Other resampling operations are performed for milestones and links that are not in F, but are not illustrated in the figure.

Figure 7 gives a more formal description of `new-roadmap`. This procedure is given a function U_m that maps any configuration m to a region of fixed size and geometry around m, which constitutes the resampling region for pushing a milestone m into F. It is also given a function U_l that maps any pair of configurations (m, m') to the resampling region used for pushing a link (m, m') into F. Step 1 builds the roadmap R' in F', while Steps 3 and 4 push this roadmap into F. Step 3

procedure `new-roadmap`:

1. Generate a roadmap $R' = (M', L')$ of size s' in the dilated free space F' by invoking `roadmap`.
2. Initiate a roadmap $R = (M, L)$ in the free space F by setting M and L to the empty set.
3. **for each** milestone m in M' **do:**
   (a) **if** m lies in F **then** add m to M.
   (b) **else repeat** x **times:**
      i. Pick a configuration q uniformly at random in $U_m(m)$.
      ii. **if** q lies in F **then** add q to M.
      until a configuration q has been added to M.
   (c) **if** a configuration has been added to M at any of the above steps, **then** denote this configuration by $p(m)$.
4. **for each** link (m, m') in L' **do:**
   (a) **if** $p(m)$ sees $p(m')$ **then** add $(p(m), p(m'))$ to L.
   (b) **else**
      i. Pick y configurations uniformly at random in $U_l(p(m), p(m'))$. Let Q be the subset of those configurations which are in F.
      ii. **if there exists** a sequence $q_1, ..., q_k$ of configurations in Q, **such that** $p(m)$ sees q_1, q_i sees q_{i+1} for $i = 1, ..., k-1$, and q_k sees $p(m')$, **then:**
      . Add every q_i, $i = 1, ..., k$ to M.
      . Add every (q_i, q_{i+1}), $i = 0, ..., k-1$ to L.

Figure 7: *The new roadmap-construction algorithm.*

pushes each milestone m of R' that is not already in F by picking up to x configurations at random in the region $U_m(m)$. The milestone of R obtained by pushing m is denoted by $p(m)$. If m was already in F, then $p(m) = m$. Step 4 pushes each link (m, m') of R' into F; this operation may lead to adding up to y milestones to R by resampling the region $U_l(p(m), p(m'))$.

The new query-processing algorithm uses `query` to connect the two query configurations q_b and q_e to two milestones m_b and m_e of R'. For $i = b, e$, if q_i is directly connected to m_i, *i.e.*, q_i sees m_i in F', then, if needed, the new algorithm pushes the link $(q_i, p(m_i))$ into F by

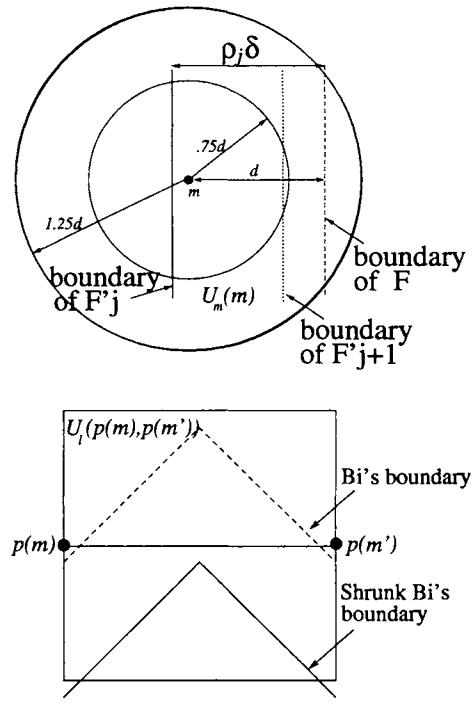


Figure 8: *The regions U_m and U_l.*

resampling the region $U_l(q_i, p(m_i))$. If, instead, q_i is connected to (*i.e.*, sees) an intermediate configuration $q \in F'$ that is itself connected to m_i, then q is first pushed to a configuration $p(q) \in F$ by resampling the region $U_m(q)$; next, both $(q_i, p(q))$ and $(p(q), p(m_i))$ are pushed into F by resampling the regions $U_l(q_i, p(q))$ and $U_l(p(q), p(m_i))$.

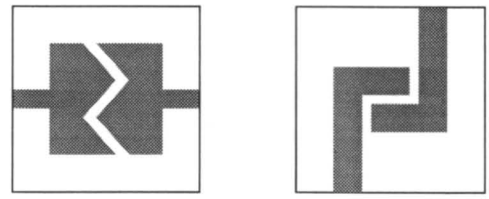
5.2 Implementation and results

We have written two programs that respectively implement the algorithms `roadmap` and `new-roadmap`. These programs share large portions of code to facilitate comparison. They both apply to polygonal free spaces. Like the planner in [13, 16], our implementation of `roadmap` only tries to connect two milestones by a link if they are closer than some predefined distance. In all the experiments reported below this distance was set to 0.5. To investigate the behavior of the new algorithm in higher-dimensional spaces, a version of `new-roadmap` has also been adapted to the case where F is a polytope in 6-D.

In the 2-D case, the free space F is input by defining several obstacles B_i, $i = 1, 2, ...$, in $\mathcal{C}$. Our programs access this representation by invoking a procedure $\texttt{cdist}(q, B_i)$ that returns the Euclidean distance between q and B_i's boundary. We use the notation "cdist" to stress the fact that the distance is computed in $\mathcal{C}$, which would not be the case in a more general implementation. This distance is positive if $q \notin B_i$, while it is negative if $q \in B_i$. Each obstacle B_i is a simple polygon, and no two obstacles B_i and B_j, $i \neq j$, intersect.

Our **new-roadmap** program uses a series of dilated free spaces, $F'_1, F'_2, ..., F'_r$, instead of a single one. The penetration distance for each F'_j $(j = 1, ..., r)$ is set equal to $\rho_j \delta$, where δ is the maximal penetration distance allowed and $1 \geq \rho_1 > \rho_2 > ... > \rho_r > 0$. In our experiments, r ranged between 1 and 5 and we set ρ_j to $(1/4)^{j-1}$ (hence, $\rho_1 = 1$). For convenience, we define F'_{r+1} to be F. The **new-roadmap** program constructs successive roadmaps $R'_1, ..., R'_r, R'_{r+1}$, where $R'_{r+1} = R$. Each milestone m of R'_j, $j \in \{1, ..., r\}$, that does not lie in F'_{j+1} is pushed into F'_{j+1} as follows. Let B_i be the obstacle in which m lies and let $d = -\texttt{cdist}(m, B_i)$. Up to x configurations are picked at random in the circular ring $U_m(m)$ shown in Figure 8 (top), which is made of all points that are distant from m by more than $.75d$ (if $j \neq r$) or d (if $j = r$) and less than $1.25d$, until one configuration is picked in F'_{j+1}. Note that the definition of U_m is related to the choice of ρ_j. For each link (m, m') of R'_j, if the line segment joining $p(m)$ to $p(m')$ does not fully lie in F'_{j+1}, then up to y additional milestones are added in the square $U_l(p(m), p(m'))$ shown in Figure 8 (bottom). In all the experiments reported below we set x to 25 and y to 10. The choices of U_m and U_l reflect ad hoc compromises. In particular, the choice of U_l allows $p(m)$ and $p(m')$ to lie near the boundary of an obstacle B_i on both sides of a 90dg corner (dashed line in Figure 8).

We experimented with **roadmap** and **new-roadmap** on several examples with narrow passages. In our experiments we only worried about the connectedness of the final roadmap. Figure 9 shows two examples, where $\mathcal{C}$ is the square $[0, 1]^2$. In both examples, we set r to 1, and the maximal penetration distance δ to 0.075. In



Example 1 Example 2

Figure 9: *Two examples.*

Algorithm	Example 1		Example 2	
	#mil.	#cdist	#mil.	#cdist
new-roadmap	65	16,812	112	11,035
roadmap	281	67,894	290	70,971

Figure 10: *Experimental results.*

Example 1, this penetration distance causes the creation of new passages between obstacles. Each *test* with one program consisted of performing a sequence of runs. For the first run in a test of **new-roadmap**, the size s' of the roadmap $R' = R'_1$ is set large enough so that the final roadmap R includes a path through the narrow passage. The following runs in the test are done with iteratively smaller values of s', until the connectivity of the final roadmap no longer reflects that of F. We call the run just before this last run the *breaking run* of the test. Each test of **roadmap** is similar. The value of s is first set large enough that the roadmap R includes a path through the passage. The following runs are done with iteratively smaller values of s, until the connectivity of R is incomplete. We did 10 tests for each example and each program, using 10 different seeds for the random number generator; we used the same 10 seeds for both programs. The table in Figure 10 shows the average numbers of milestones in the roadmap R generated in the breaking runs and the average numbers of times **cdist** was invoked in those runs. The running time is not significative and is not indicated. In both examples we see that **new-roadmap** captures the free space connectivity with, on the average, considerably fewer milestones than **roadmap**; the average number of invocations of **cdist** is also much smaller. Figure 11 displays distributions of milestones

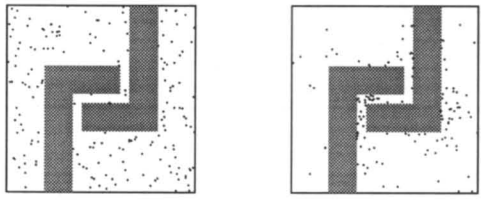


Figure 11: *Milestone distributions for Example 2.*

w/W	r	#mil. (initial)	#mil. (final)	#cdist
.1	1	33	95	9,320
.01	1	38	118	32,942
.001	4	34	200	79,931
.0001	4	35	193	72,944
.00001	5	34	191	82,958

Figure 12: *Results with the example of Figure 5.*

generated for Example 2 by a breaking run of `roadmap` (left) and a breaking run of `new-roadmap` (right).

In another series of experiments with `new-roadmap` we considered the example of Figure 5, with w/W successively set to .1, .01, .001, .0001, and .00001. $\mathcal{C}$ was defined as $[0,3] \times [0,1]$. The maximal penetration distance was set to 0.45. For each value of w/W, we did the same kind of tests as above. The table of Figure 12 displays the average results over 5 tests. We used $r = 1$ for $w/W = .1$ and .01, $r = 4$ for $w/W = .001$ and .0001, and $r = 5$ for $w/W = .00001$. In this table, we indicate the numbers of milestones in the initial roadmap (R'_1) and in the final one (R). Note that the initial and final numbers of milestones, as well as the number of invocations of `cdist`, remain approximatively constant when w/W decreases from .001 to .00001. (The average running time also stayed roughly constant.) When $w/W = .00001$, the average number of milestones in the final roadmap is 191; in the same free space, `roadmap` would have had to generate on the order of 1000 times as many milestones, and we believe that this factor would continue to grow if w/W was set smaller. Figure 13 shows the distribution of milestones in a final roadmap in the case where $w/W = .001$ (the passage is barely visible in the figure). Due to the resampling at Step 3(b), this distribution is denser near the obstacle boundary. Due to the resampling of Step 4(b), it is maximum near the two entrances of the passage.

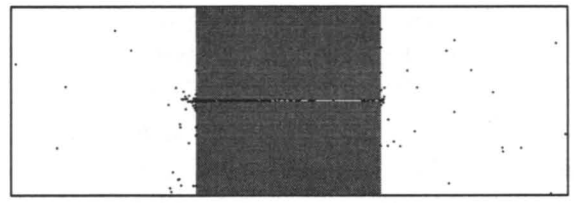


Figure 13: *Milestone distribution for example of Fig. 5.*

k	average #mil. (`new-roadmap`)	estimated #mil. (`roadmap`)
1	61	40
2	105	800
3	1656	16,000

Figure 14: *Results with 6-D example.*

We have adapted our `new-roadmap` program to handle the 6-D version of the example of Figure 5. We did experiments with $w/W = .05$ and $k = 1$, 2, and 3 (recall that k is the number of dimensions along which the passage is narrow). The second column of the table of Figure 14 shows the average numbers of milestones (over 5 runs) in roadmaps created by `new-roadmap` that are connected through the narrow passage. The third column contains an estimate of the numbers of milestones that `roadmap` would have to pick to build connected roadmaps with high probability.

5.3 Computation of penetration distance

The function `cdist` used in our implementation of `new-roadmap` computes the (penetration) distance in configuration space. But for path-planning problems involving objects of complex geometry and robots with many degrees of freedom, it is impractical to perform this computation in configuration space.

In Section 2 we introduced the function `dist`(q) that computes the Euclidean distance of the robot placed at q and the obstacles. To be more specific, let the robot be made of p rigid bodies $L_1, ..., L_p$. Let t_i be the shortest translation that L_i must undergo before it touches an obstacle. The distance `dist`(q) is

$\min_{1=1}^{p}\{t_i\}$. As we mentioned in Section 2, a number of techniques have been proposed to implement `dist` [7, 8, 12, 18, 19, 20, 22, 25].

To be usable by `new-roadmap`, the function `dist`(q) should be extended to compute the (negated) penetration distance of the robot into the obstacles at configurations where $q \notin F$. Then, $q \in F'$ if and only if `dist`$(q) > -\delta$. Consistently with the above definition of `dist`(q) when $q \in F$, we can define the penetration distance of the robot placed at $q \notin F$ as follows. Let t_i be the shortest translation that L_i must undergo before it does not collide with any obstacle. The penetration distance `dist`(q) is $\max_{i=1}^{p}\{t_i\}$. Techniques for computing this penetration distance are proposed in [6, 7, 17, 26] for convex polytopes and in [20] for nonconvex ones. Note that this distance does not correspond to any metric in $\mathcal{C}$. Hence, allowing a penetration distance δ of the robot into the obstacles does not result in a uniform dilatation of F.

Another penetration distance is defined in [21, 22]. It is based on the notion of the "growth distance" between two objects and it is computed by growing/shrinking objects about a seedpoint. The computation of this distance has been implemented for convex obstacles.

To experiment with our sampling strategy on realistic path-planning problems, we project to implement a new version of the `new-roadmap` that uses a function `dist` based on V-Clip [20].

5.4 Discussion

We now discuss several aspects of the new sampling strategy in light of the results presented in Section 4. We recognize that a more formal analysis will eventually be desirable. For simplification, we assume that the dilated free space F' is obtained by growing F by δ isotropically. As mentioned above, allowing some penetration distance δ of the robot into the obstacles does not entail a uniform dilatation of F by δ. But, for any given robot, there exists a positive constant h such that $F \subset F_h \subseteq F'$, where F_h is obtained by uniformly dilating F by $h \times \delta$. We also assume that the volume of F' is not much greater than that of F, which simply means that δ is relatively small. This makes sense since the only thing we want is to significantly widen those passages which are too narrow to be found by `roadmap`.

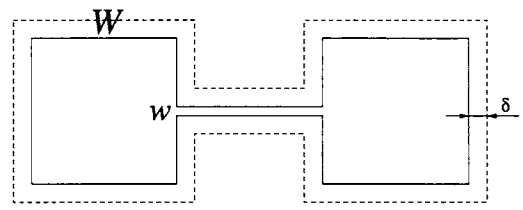


Figure 15: *Example of a dilated free space.*

- Consider the path-clearance assumption of Subsection 4.3. If two configurations q and q' are connected by a path of small clearance σ in F, the same path has clearance $\sigma + \delta$ in F'. This reduces the bound on the number of milestones given by Theorem 4 by a factor approximately equal to $1/(1+c)^n$, where $c = \delta/\sigma$. In the example of Figure 5, c can be made very large. Hence, a relatively small roadmap R' can reliably capture the connectivity through the narrow passages of F.

- Consider the example of Figure 5. Recall that this problem has an n-D version where the passage is narrow along k dimensions, with $k = 1, 2, ...,$ or $n-1$. In 2-D, F' is the region depicted with a dotted boundary in Figure 15 (assuming that F' does not extend beyond $\mathcal{C}$'s boundary). In n-D, F is $(\epsilon, \alpha, \beta)$-expansive for values of ϵ, α, and β that are on the order of $(w/W)^k$. In F' the values of these parameters are on the order of $[(w+\delta)/W]^k$. So, posing $c = \delta/w$, the ϵ-goodness and expansiveness of F' are greater than those of F by a factor of $(1+c)^k$. The higher k, the greater this factor, which means that the worst case for F (which occurs when $k = n-1$) also yields the best improvement. Again, this means that a relatively small roadmap R' can reliably capture the connectivity of F.

The above two arguments indicate that, when a passage gets narrower, the gain in number of milestones achieved by `new-roadmap` over `roadmap` is on the same order of magnitude as the number of additional milestones that `roadmap` must generate to keep the roadmaps connected. This indication is consistent with the observation that the number of milestones remains approximately constant in Figure 12.

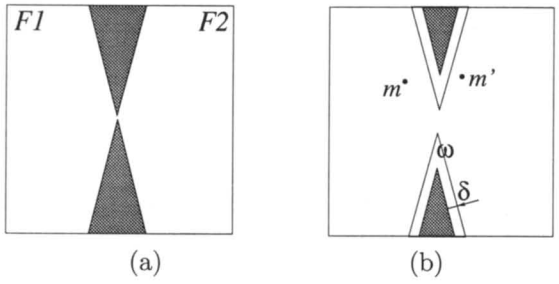


Figure 16: *A pathological case.*

• Every milestone m picked at Step 1 in F', but not in F, is pushed into F at Step 3(b) by resampling $U_m(m)$. This operation produces a milestone distribution in F that is denser near its boundary. This effect is accentuated further by Step 4(b). If F has poor ϵ-goodness or expansiveness, the regions responsible for that are necessarily close to its boundary. Therefore, the resampling operations at Steps 3(b) and 4(b) increase the density of milestones where they are they are likely to be the most useful.

• In general, there is no guarantee that F' is more ϵ-good and/or expansive than F. However, we can define the ϵ-goodness and expansiveness of F through F' by extending the visibility relation as follows: two configurations in F see each other *through* F' if the straight line segment joining them lies in F'. The ϵ-goodness of F through F' is greater than its ϵ-goodness. Let S be a subset of a component E of F. Let the lookout of S through F' be the set of all points in S that see a significant portion of $E \backslash S$ through F'. The expansiveness of F through F' is greater than its expansiveness. Consider the roadmap R'' that contains all the milestones of R' lying in F and all the links between these milestones. (While all the milestones of R'' lie in F, its links may lie in F'.) Theorems 1 and 3 indicate that R'' can capture the connectivity of F (through F') with fewer milestones than a roadmap generated by `roadmap` in F.

• R' may contain more links than desired, since there may exist connections between milestones through F' that do not exist in F. In that case Step 4 will fail to push such links into F. It will waste some time trying to establish impossible connections. But it is reasonable to expect that this will only happen in a small number of localized areas. It is to be compared to the uniformly denser sampling that `roadmap` would have to perform to find existing narrow passages.

• The above comment raises the opposite question. Can Steps 3 and 4 of `new-roadmap` push the other links into F reliably? For example, shrinking the two triangular obstacles in Figure 16(a) by a relatively small δ causes a big widening of the passage, as depicted in Figure 16(b). The two milestones m and m' in this figure see each other through F', but pushing the link between them into F would require resampling a region $U_l(m, m')$ that would grow arbitrarily large as the angle ω decreases. The choice of U_l in our implementation (Figure 8) rests on an ad hoc compromise. When $\mathcal{C}$ has many dimensions and penetration distances are computed in the robot's workspace, there seems to exist no easy way to optimally set U_l's size. The use of successive dilated free spaces, as in our implementation, is one way to reduce the risks of disconnecting a roadmap when U_l is too small. Another way would be to try to connect pushed milestones $p(m)$ to more milestones in R. Finally, one should also note that a typical roadmap contains redundant links; hence, if Step 4 fails to push a link into F because U_l is too small, it may successfully push another nearby link and still get an adequate roadmap R.

• One may be tempted to choose a large value for δ in order to widen narrow passages as much as possible. However, a large value of δ may also cause large portions of obstacles or entire obstacles to vanish. At one extreme, if the dilated free space F' becomes the entire configuration space, the roadmap R' does not provide pertinent information to `new-roadmap` on which regions should be more densely sampled. To illustrate, consider Example 2 in Figure 9. One could make the two obstacles very "skinny" and the passage between them arbitrarily narrow. Setting δ to a value greater than half the thickness of the obstacles eliminates these obstacles and makes F' useless. Alternatively setting δ to a smaller value only results in a small widening of the passage, which hence remains hard to find.

• In Subsection 5.1 we could have kept the query-processing algorithm unchanged. However, it is possi-

ble that by reducing the number of milestones in R, these milestones may not adequately cover F, even though they may adequately cover F through F'. For that reason, the new query-processing algorithm connects the query configurations to R' and, if needed, pushes the connections into F.

6 Conclusion

This paper investigates the effect of narrow passages on the performance of a PRM. Section 4 provides foundations for understanding the effect of narrow passages on probabilistic roadmaps. It proposes three geometric properties, ϵ-goodness, expansiveness, and path-clearance, to assess the complexity of a path-planning problem submitted to a PRM. Section 5 uses these properties to propose and motivate a new approach to effectively deal with such passages. It also presents encouraging experimental results with an implementation of this approach in a polygonal configuration space.

It is clear, however, that much additional work is needed to formally analyze the proposed strategy, to understand the role of key parameters (e.g., δ, U_m, U_l, x, y) and how they could be adjusted on-line, to efficiently compute penetration distances, and to conduct more realistic experimentation. The computation of penetration distances is likely to be the key implementation issue when the robot and the obstacles have complex geometry, since it is inherently more expensive than the computation of separation distances.

Acknowledgments

This research was supported in part by ARO MURI grant DAAH04-96-1-007. D. Hsu is currently supported by a Microsoft Fellowship. L. Kavraki is partially supported by NSF CAREER Award IRI-970228. The authors also thank Leo Guibas, who first pointed to us that increasing milestone density near the free space boundary could help finding narrow passages.

References

[1] N. Amato, O.B. Bayazit, L.K. Dale, C. Jones, and D. Vallejo. OBPRM: An Obstacle-Based PRM for 3D Workspaces. *These proceedings*, 1998.

[2] N. Amato and Y. Wu. A Randomized Roadmap Method for Path and Manipulation Planning. *Proc. IEEE Int. Conf. on Rob. and Aut.*, Minneapolis, MN, pp. 113-120, 1996.

[3] B. Baginski. Local Motion Planning for Manipulators Based on Shrinking and Growing Geometry Models. *Proc. IEEE Int. Conf. on Rob. and Aut.*, Minneapolis, MN, pp. 3303-3308, 1996.

[4] J. Barraquand, L.E. Kavraki, J.C. Latombe, T.Y. Li, R. Motwani, and P. Raghavan. A Random Sampling Scheme for Path Planning. *Int. J. of Rob. Res.*, 16(6):759-774, 1997.

[5] J. Barraquand and J.C. Latombe. Robot Motion Planning: A Distributed Representation Approach. *Int. J. of Rob. Res.*, 10(6):628-649, 1991.

[6] S.A. Cameron. Enhancing GJK: Computing Minimum and Penetration Distances Between Convex Polyhedra. *Proc. IEEE Int. Conf. on Rob. and Aut.*, Albuquerque, NM,pp. 3112-3117, 1997.

[7] D.P. Dobkin, J. Hershberger, D.G. Kirkpatrick, and S. Suri. Computing the Intersection Depth of Polyhedra. *Algorithmica*, 9:518-533, 1993.

[8] E.G. Gilbert, D.W.Johnson, and S.S. Keerthi. A Fast Procedure for Computing the Distance Between Complex Robots in Three-Dimensional Space. *IEEE Tr. Rob. and Aut.*, 4:193-203, 1988.

[9] P. Ferbach and J. Barraquand. A Method of Progressive Constraints for Manipulation Planning. *IEEE Tr. Rob. and Aut.*, 13(4):473-485, 1997.

[10] D. Hsu, J.C. Latombe, and R. Motwani. Path Planning in Expansive Configuration Spaces. Proc. *IEEE Int. Conf. pn Rob. and Aut.*, Albuquerque, NM, pp. 2719-2726, 1997. An extended version of this paper will appear in *Int. J. of Comp. Geom. and Applicatons.*

[11] T. Horsch, F. Schwarz, and H. Tolle. Motion Planning for Many Degrees of Freedom - Random Reflections at C-Space Obstacles. *Proc. IEEE Int. Conf. on Rob. and Aut.*, San Diego, CA, pp. 3318-3323, 1994.

[12] P. Jiménez, F. Thomas, and C. Torras. Collision Detection Algorithms for Motion Planning. *Robot Motion Planning and Control*, J.P. Laumond (ed.), Lecture Notes in Control and Info. Sc., 229, Springer, New York, NY, pp. 305-343, 1998.

[13] L. Kavraki. *Random Networks in Configuration Space for Fast Path Planning.* Ph.D. Thesis, Rep. No. STAN-CS-TR-95-1535, Comp. Sc. Dept., Stanford Univ., Stanford, CA, 1995.

[14] L. Kavraki, M. Kolountzakis, and J.C. Latombe. Analysis of Probabilistic Roadmaps for Path Planning. *Proc. IEEE Int. Conf. on Rob. and Aut.*, Minneapolis, MN, pp. 3020-3025, 1996.

[15] L. Kavraki, J.C. Latombe, R. Motwani, and P. Raghavan. Randomized Query Processing in Robot Motion Planning. *Proc. ACM SIGACT Symp. on Theory of Computing (STOC)*, Las Vegas, NV, pp. 353-362, 1995.

[16] L. Kavraki, P. Švestka, J.C. Latombe, and M. Overmars. Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces. *IEEE Tr. Rob. and Aut.*, 12(4):566-580, 1996.

[17] S.S. Keerthi and K. Shidharan. Measures of Intensity of Collision Between Convex Objects and Their Efficient Computation. *Proc. of Int. Conf. on Intelligent Robotics*, pp. 266-272, 1991.

[18] M. Lin and J.F. Canny. A Fast Algorithm for Incremental Distance Computation. *Proc. of the IEEE Int. Conf. on Rob. and Aut.*, Sacramento, CA, pp. 602-608, 1994.

[19] M. Lin, D. Manocha, J. Cohen, and S. Gottschalk. Collision detection: Algorithms and applications. *Algorithmic Foundations of Robotics*, Goldberg et al. (Eds), A K Peters, Ltd., pp. 129-141, 1995.

[20] B. Mirtich. *V-Clip: Fast and Robust Polyhedral Collision Detection.* Tech. Rep. TR97-05, Mitsubishi El. Res. Lab., Cambridge, MA, 1997.

[21] C.J. Ong. On the Quantification of Penetration between General Objects. *Int. J. of Rob. Res.*, 16(3):400-409, 1997.

[22] C.J. Ong and E.G. Gilbert. Growth Distances: New Measures for Object Separation and Penetration. *IEEE Tr. Rob. and Aut.*, 12(6):888-903, 1996.

[23] M. Overmars. *A random Approach to Motion Planning.* Tech. Rep. RUU-CS-92-32, Comp. Sc. Dept., TB Utrecht, The Netherlands, 1992.

[24] M. Overmars and P. Švestka. A Probabilistic Learning Approach to Motion Planning. *Algorithmic Foundations of Robotics*, K. Goldberg et al. (eds.), A.K. Peters, Wellesley, MA, pp. 19-37. 1995.

[25] S. Quinlan. Efficient Distance Computation Between Non-Convex Objects. *Proc. IEEE Int. Conf. on Rob. and Aut.*, San Diego, CA, pp. 3324-3330, 1994.

[26] K. Sridharan, H.E. Stephanou, K.C. Craig, and S.S. Keerthi. Distance Measures on Intersecting Objects and their Applications. *Inf. Proc. Letters*, 51:181-188, 1994.

[27] P. Švestka and M. Overmars. Probabilistic Path Planning. *Robot Motion Planning and Control*, J.P. Laumond (ed.), Lecture Notes in Control and Information Sciences, 229, Springer, New York, NY, pp. 255-304, 1998.

Taylor & Francis
Taylor & Francis Group
http://taylorandfrancis.com

OBPRM: An Obstacle-Based PRM for 3D Workspaces

Nancy M. Amato, *Texas A&M University, College Station, TX, USA*
O. Burchan Bayazit, *Texas A&M University, College Station, TX, USA*
Lucia K. Dale, *Texas A&M University, College Station, TX, USA*
Christopher Jones, *Texas A&M University, College Station, TX, USA*
Daniel Vallejo, *Texas A&M University, College Station, TX, USA*

*Recently, a new class of randomized path planning methods, known as Probabilistic Roadmap Methods (*PRMs*) have shown great potential for solving complicated high-dimensional problems.* PRMs *use randomization (usually during preprocessing) to construct a graph of representative paths in C-space (a* roadmap*) whose vertices correspond to collision-free configurations of the robot and in which two vertices are connected by an edge if a path between the two corresponding configurations can be found by a local planning method.*

This work describes and evaluates various node generation and connection strategies for one such PRM*, the obstacle-based probabilistic roadmap method (*OBPRM*), in cluttered 3-dimensional Workspaces. Various node generation strategies are evaluated in terms of their ability to produce nodes in difficult regions of C-space; our results include recommendations for selecting appropriate node generation strategies for different types of objects, and a default strategy for use when objects cannot be classified easily. We also propose and analyze a multi-stage strategy for connecting the roadmap nodes; the use of different local planners at different stages is shown to enhance the connectivity of the resulting roadmap significantly.*

1 Introduction

Automatic motion planning has application in many areas such as robotics, virtual reality systems, and computer-aided design. Although many different motion planning methods have been proposed, most are not used in practice since they are computationally infeasible except for some restricted cases, e.g., when the robot has very few degrees of freedom (dof) [12, 20]. Indeed, there is strong evidence that any complete planner (one that is guaranteed to find a solution or determine that none exists) will require time that is exponential in the number of dof of the robot [23]. For this reason, attention has focussed on randomized or probabilistic motion planning methods. Notable among these are randomized potential field methods (e.g., RPP [4]), which work very well when the configuration space (C-space) is relatively uncluttered, but unfortunately there also exist simple situations in which they are not successful [5, 17].

Recently, a new class of randomized motion planning methods has gained much attention (see, e.g., [1, 3, 11, 10, 16, 17, 19, 21, 22]). These methods, known as *probabilistic roadmap methods* (PRMs), use randomization (usually during preprocessing) to construct a graph in C-space (a *roadmap* [20]). Roadmap nodes correspond to collision-free configurations of the robot. Two nodes are connected by an edge if a path between the two corresponding configurations can be found by a local planning method. Queries are processed by connecting the initial and goal configurations to the roadmap, and then finding a path in the roadmap between these two connection points.

PRMs: Probabilistic Roadmap Methods
I. Preprocessing: Roadmap Construction
  1. Node Generation (find collision-free configurations)
  2. Connection (connect nodes to form roadmap)
     (repeat as desired)
II. Query Processing
  1. Connect start/goal to roadmap
  2. Find path in roadmap between connection nodes

PRMs have been shown to perform well in practice. In particular, after the roadmap is constructed during preprocessing, many difficult planning queries can be answered in fractions of seconds [3, 19]. Although PRMs

are particularly suitable when multiple queries will be answered in the same static environment, the general PRM strategy can be used to solve single queries by only constructing 'useful' portions of the roadmap [11, 22].

Node Generation. Node generation strategies are the methods used to select collision-free robot configurations to be used as nodes in the roadmap. A good node generation strategy will produce nodes that can be connected to form a roadmap that is representative of the connectivity and complexity of C-free. Ideally, the roadmap should contain nodes in every C-space crevice and corridor. However, guaranteeing this requires the costly computation of the constraint surfaces — which is what randomized methods seek to avoid.

The first PRMs [17, 19] use uniform sampling in C-space to generate roadmap candidate nodes (collision-free configurations are retained); roadmaps are enhanced by further sampling in 'difficult' regions. These methods perform well for general many-dof robots. However, their effectiveness decreases as the environments become more cluttered since uniform sampling of C-space is unlikely to yield configurations in narrow regions of C-space. In [10], a roadmap is first built in a dilated free-space, and then efforts are made to 'push' it into the true free-space by resampling. To obtain improved roadmaps in crowded situations, some PRMs use information about the environment to guide node generation. Examples include executing random reflections at C-obstacle surfaces [9], and a technique called geometric node adding [22] for generating configurations of non-articulated robots near Workspace obstacle boundaries.

Connection. After the collision-free roadmap candidate nodes are generated, they must be connected to form the roadmap. The basic idea is to attempt to connect selected pairs of roadmap nodes using some local planning method(s); each successful connection identifies an edge in the roadmap. To save space, the paths found in this stage are not recorded since they can be re-generated quickly when processing queries.

The methods by which a PRM determines which (and how many) nodes to attempt to connect, and the local planner(s) selected to make those connections can crucially impact both the quality of the resulting roadmap and the running time of the PRM. Indeed, even though most PRMs greatly limit the number of connections attempted (say, to ten for each node), they still typically spend more than 95% of their preprocessing time in the connection phase [3, 19].

The general strategy of PRMs is to first make as many of the 'easy' and 'cheap' connections as possible, and then to use more sophisticated techniques to improve the roadmap's quality. For example, the PRM of [17, 19] first tries to connect each node to the k (a parameter) closest nodes (as determined by some distance metric) using the common straight-line in C-space local planner, and then attempts to enhance the roadmap by sampling more nodes in identified 'difficult' regions and/or by using more sophisticated local planners such as RPP [4].

1.1 Our results

In this paper we consider an obstacle-based PRM (OBPRM) [3, 25] which samples points on or near C-obstacle surfaces. Even though the prototype implementation of OBPRM for planar articulated robots employed only the simplest node generation and connection strategies, it established that OBPRM was a promising method for planning in cluttered environments. Here, we describe several more sophisticated strategies for the node generation and connection phases, and provide an evaluation of a more mature implementation of the method for cluttered 3-dimensional Workspaces, typical, e.g., of mechanical designs [6]. The moving objects (robots) are rigid, non-articulated objects yielding six-dimensional C-spaces. Although we concentrate on OBPRM, we believe the techniques proposed here will be useful for PRMs in general, and for other motion planning approaches as well.

We propose and study various heuristics for generating nodes on or near C-obstacle surfaces and evaluate their effect on roadmap quality. The results of our study include recommendations for selecting appropriate combinations of node generation strategies for use with objects of certain general shapes and compositions (e.g., symmetrical with surfaces described by triangles

of roughly equivalent area). We identify a default strategy for use with unclassified objects.

We also propose a multi-stage connection strategy that is shown to significantly improve the connectivity of the resulting roadmaps, while still controlling the time needed to construct them. The first stage attempts to make the 'easy' connections using many invocations of the fastest local planning method(s), while the later stages make a decreasing number of connection attempts and use increasingly more powerful local planning methods. The last stage may add new roadmap nodes as it attempts to join different connected components of the previous roadmap.

We remark that our goal is to provide empirical evidence that certain node generation and connection strategies work well for OBPRM for certain types of problems. We seek empirical evidence due to the randomized nature of PRMs, which makes them difficult to analyze. Recently, a number of attempts have been made to theoretically explain the success of PRMs (see, e.g., [11, 15, 18]). However, these studies generally make simplifying assumptions regarding the nature of the C-space and/or the PRM components (e.g., local planner), and therefore unfortunately cannot be applied to OBPRM.

2 Preliminaries

The moving objects (robots) considered in this paper are rigid objects in three-space. We represent configurations using six-tuples $(x, y, z, \alpha, \beta, \gamma)$, where the first three coordinates define the position and the last three define the orientation. The orientation coordinates are represented in radians, normalized to $[0 - 1)$.

In addition to collision detection, all PRMs make heavy use of so-called *local planners* and *distance computations*. Local planners are simple, fast, deterministic methods used to make connections between roadmap nodes when building the roadmap, and to connect the start and goal to the roadmap during queries. Distance metrics are used to determine which pairs of nodes one should try to connect.

The local planners and distance metrics used in OBPRM are based on recommendations from [2]. The distance metric we use is scaled Euclidean distance in C-space (the scale places more or less weight on the position coordinates).

The local planners currently implemented in OBPRM are the common *straight-line* in C-space, three versions of a (parameterized) planner proposed in [2] called *rotate–at–s* ($0 \leq s \leq 1$), and some A^*-like methods (see, e.g., [7, 8, 13, 14, 24]). The suggested order to apply these planners was: first straight-line and rotate-at-$\frac{1}{2}$ (the most successful planners), next rotate-at-0 and rotate-at-1, and finally, the more expensive A^*-like planners, which try to fill in gaps left by the faster planners.

rotate-at-s. Briefly, when moving from c_1 to c_2, the rotate-at-s planner first translates from c_1 to an intermediate configuration c', rotates to a second intermediate configuration c'', and finally translates to c_2. The parameter s represents the fractional part of the translational distance between c_1 and c_2 that the robot travels from c_1 to c'. The straight-line planner is used to plan between each pair of configurations, that is, between (c_1, c'), between (c', c''), and between (c'', c_2).

The rotate–at–$\frac{1}{2}$ planner made more connections than the straight-line planner in nearly every situation studied, perhaps explained by the fact that it has a smaller swept volume. It is also fairly fast, taking about twice as long as the straight-line planner (significantly less than the A^* methods). The other two versions of rotate–at–s used here are $s = 0$ and $s = 1$, which although not as successful do make some connections the straight-line and rotate–at–$\frac{1}{2}$ do not.

A^*-like planners. The A^*-like planners used in OBPRM vary according to: (1) the number of neighboring configurations explored in each iteration, (2) the evaluation function used to select among the neighbors, and (3) the number of iterations they are allowed to run. Currently, OBPRM supports three different neighbor functions yielding three, nine, and fifteen neighbors. The first three neighbors, which are common to all functions, are the configurations in which (i) only the position coordinates, (ii) only the orientation coordinates, and (iii) all coordinates are incremented towards the goal. The additional six (or twelve) neighbors are

the configurations in which exactly one of the coordinates is incremented towards (or away from) the goal. The two evaluation functions used are (i) minimum *distance* from the goal, and (ii) maximum *clearance* from a workspace obstacle; both these functions are approximated using the center of mass of the relevant objects. The maximum number of iterations the planner is allowed to run is a multiple of the number of steps the straight-line planner would take; common values for this are 3, 6, 9, or 15. Note these planners are not truly A^* methods, but rather employ A^*-like strategies.

We denote the various versions of these planners by A^*−eval(nbrs,steps), e.g., A^*-clearance(9nbrs,6steps). Although significantly more expensive than the other planners, the A^*-like methods can sometimes make connections the others can not by 'feeling' their way along in tight places.

3 Node Generation in OBPRM

The prototype version of OBPRM [3, 25] for a many-dof articulated robot in a 2-dimensional Workspace used a simple strategy to generate nodes on contact surfaces. Briefly, for each obstacle X:

PROTOTYPE NODEGENERATION
1. c_{in} := colliding robot cfg with C-obstacle X
2. D := m random directions emanating out from c_{in}
3. **for** each $d \in D$
4. c_{out} := free cfg in direction d (if exists)
5. find contact cfg on (c_{in}, c_{out}) by binary search
6. **endfor**

This simple strategy was sufficient to establish the potential of the method. However, it is clear that more sophisticated node generation strategies are needed for more complex objects to produce a 'good' distribution of nodes in all the 'different' regions of C-free. Outlined below are some of the methods we've implemented and tested. Keeping in the spirit of OBPRM, all these methods employ information regarding the environment to guide node generation. Briefly, the methods are designed to generate three types of nodes: (i) *contact* configurations, (ii) *free* configurations (near contact surfaces), and (iii) sets of configurations (*shells*) surrounding C-obstacles.

Generating contact configurations. The node generation strategy used in the prototype version of OBPRM is attractive due to its simplicity and its efficiency (node generation typically accounted for 1-2% of preprocessing time). However, the distribution of the generated nodes is clearly very sensitive to both the shape of the C-obstacle and to the *seed* (origin c_{in} for the binary search). That is, no single seed will yield a good distribution of configurations on the surface of the C-obstacle if its shape is not roughly spherical, and even if the C-obstacle is spherically shaped, a seed removed from the C-obstacle's center will yield more configurations on the region of its surface closest to the seed.

Ideally, we would like a method that is as simple as the binary search technique, but which is less sensitive to the shape of the C-obstacle (which we wish to avoid computing explicitly). One way to achieve this is to use multiple seeds – so long as they are chosen in an intelligent manner. Indeed, the current version of OBPRM uses a different seed for (nearly) every node generated:

GENERATECONTACTCONFIGURATION
1. p_{rob} := point associated with robot
2. p_{obs} := point associated with obstacle of interest
3. c_{in} := translate robot so p_{rob} and p_{obs} coincide and rotate robot randomly until collision
4. d := random direction emanating out from c_{in}
5. c_{out} := free cfg in direction d (if exists)
6. find contact cfg on (c_{in}, c_{out}) by binary search

The types of configurations found in this manner depend upon the shapes of the robot and obstacle, and also on the selected points p_{rob} and p_{obj} (generally these points will be on the objects, e.g., a vertex). We have implemented and tested several methods for selecting points in an object (robot or obstacle) in OBPRM (see Table 1). There are, of course, many other possible variants (e.g., edges instead of triangles), but the ones we've implemented can be thought of as representative strategies.

Generating free configurations. Sometimes roadmap connectivity might be improved by generating some points in freespace – but near constraint surfaces. For example, such configurations may give local planners room to 'maneuver' around obstacle corners. It

POINT SELECTION STRATEGIES (NODE GENERATION)	
Strategy	*Explanation and Bias*
cM center of mass	The center of mass of the object vertices. Good for (roughly) spherical objects, and also useful to ensure that some points are generated in the vicinity of the center of mass of that object.
rV random vertex	A random object vertex. Biased towards those portions of the object that have more vertices – which are generally the more complex regions.
eV extreme vertex	Select one of 6 extreme vertices with maximal and minimal x, y, and z coordinates at random. Finds configurations distributed around the object, even in regions which do not have high descriptive complexity (which might be captured by the random vertex case).
rT random triangle	Select a triangle at random, and then randomly select a point in that triangle. Biased towards those portions of the object that have more triangles – which are often the regions where the object's shape (e.g., curvature) changes.
wT weighted triangle	Select a triangle with probability proportional to its area, and then randomly select a point in that triangle. Biased towards triangles with large area (largest parts of object).

Table 1: *Point selection strategies for node generation.*

turns out the above described seed generation strategies can easily be adapted to generate such free configurations. In particular, after selecting the two points p_{rob} and p_{obj} in the robot and obstacle (according to one of the methods described above), we translate the robot until these points coincide and then randomly rotate the robot to obtain a *free* configuration (rather than a configuration in collision).

Building Shells. The general idea behind OBPRM is that the roadmap should be densest around the C-obstacles since that is where planning is hard. However, since planning is very difficult near contact surfaces, we would like to include paths in the roadmap that leave some clearance between the robot and the obstacles. Roadmap nodes that might be contained in such paths can be found at low cost during node generation. In particular, we can retain some number s of the configurations found during the search (e.g., the s closest, or s uniformly spaced among those found). We refer to s as the number of *shells* generated for the object (as they will hopefully encase the object).

4 Roadmap Connection in OBPRM

The first PRMs attempt connections between each node and the k (e.g., 10) closest nodes to it (as determined by some distance metric) [19]. This is a natural approach to take since initially there is little to differentiate one configuration from another. In OBPRM, however, this strategy would result in most connection attempts being between nodes associated with the same C-obstacle which may be more difficult to connect and also might not be as useful, in terms of roadmap connectivity, as connections between different C-obstacles.

There are three general stages in OBPRM's roadmap connection strategy. The first stage (SIMPLE CONNECTION) attempts to make the 'easy' connections using many invocations (typically thousands) of the fastest local planning method(s). The next two stages (CONNECTING COMPONENTS and GROWING COMPONENTS) make a decreasing number of connection attempts and use increasingly more powerful local planning methods. The third stage may add new roadmap nodes as it attempts to join different connected components of the previous roadmap. The reason for the multi-stage strategy is that the faster local planners are not capable of making the crucial connections needed for a well connected roadmap, while the more powerful local planners are computationally too expensive to be used for the easier connections (which generally account for the majority of the roadmap edges).

Overviews of the strategies applied in each stage are described below. Within each stage, there are still sev-

eral choices to be made, such as, for example, which local planner(s) and distance metric(s) to use. Based on the findings in [2], the current implementation of OBPRM uses a scaled Euclidean distance metric and employs multiple local planners (see Section 2). We note, that it is likely that the best methods will vary according to the particular problem, and thus ideally the planner should adaptively select appropriate distance metrics and local planning methods.

Stage 1: Simple Connection. The first connection stage in OBPRM is similar to those used in the original PRMs. The main difference is that it takes advantage of the extra C-obstacle information associated with the nodes:

STAGE 1: SIMPLECONNECTION
1. let V_i be the set of nodes associated with C-obstacle i
2. let m be the number of C-obstacles
3. `for` each $v \in V$
4. `for` $i := 1, m$
5. Compute $C_{v,i}$, the closest `K1` nodes in V_i to v.
6. Try to connect v to each node in $C_{v,i}$.
7. `endfor`
8. `endfor`
9. ANALYZEROADMAP /*compute connected components*/

The input parameter `K1`, the number of connections attempted for each node, is typically quite small (e.g., 10). Currently, the only local planner OBPRM uses in this stage is the *straight-line* in C-space, which is our fastest planner. After the `K1` connections have been attempted, the connectivity of the resulting roadmap is analyzed, e.g., its connected components, and their sizes, are computed. If, as is usually the case, the roadmap does not consist of a single connected component, we proceed to the next connection stage.

Stage 2: Connecting Components. The goal of this stage is to make connections between different connected components of the stage one roadmap. In particular, the strategy is similar to stage one, except that now connections are attempted between nodes that belong to different connected components rather than different C-obstacles:

STAGE 2: CONNECTINGCOMPONENTS
1. $V_i = \{v | v \in \text{connected component } i\}$, $1 \leq i \leq m$
2. assume $|V_1| \leq |V_2| \leq \ldots \leq |V_m|$
3. `for` $i := 1, m$ /*connected components*/
4. `for` $j := i+1, m$ /*bigger connected components*/
5. `if` $|V_i| \leq$ `MAX2`
6. `then` ATTEMPTALL(V_i, V_j,`K2.1`)
7. `else` ATTEMPTK(V_i, V_j,`K2.2`)
8. `endfor`
9. `endfor`
10. ANALYZEROADMAP /*compute connected components*/

Each iteration of the inner for loop attempts to make a connection between two connected components V_i and V_j. If the size of the smaller component, V_i, is less than some constant `MAX2` (typically 10-30), then connection attempts are made between every node in V_i to the `K2.1` closest nodes in V_j. If, however, $|V_i| \geq$ `MAX2`, then connections are attempted for the `K2.2` closest pairs of nodes, one in V_i and one in V_j. In either case, the procedure returns as soon as one connection is made.

The **stage** 2 strategy implemented in OBPRM is actually a bit more sophisticated in that it consists of several substages, each one patterned after the procedure outlined above. They differ from each other in the values of the parameters `MAX2`, `K2.1`, and `K2.2`, and in the set of local planners used to make the connections. In particular, there are two significant substages:

stage 2.1: `K2.1` = 10, `K2.2` = 10, `MAX2` = 10. Local Planners: straight-line in C-space, rotate-at-$\frac{1}{2}$.

stage 2.2: `K2.1` = 10, `K2.2` = 8, `MAX2` = 10. Local Planners: straight-line in C-space, straight-line in C-space, rotate-at-$\frac{1}{2}$, A^*-clearance(15nbrs,9steps).

The connectivity of the roadmap resulting from each substage is updated before the next substage is entered.

Stage 3: Growing Components. The first two connection stages try to make as many connections as possible between nodes created in the initial node generation stage. As in **stage** 2, the main goal of **stage** 3 is to merge different connected components in the current roadmap. However, whereas in **stage** 2 we simply tried to make connections between existing nodes in the two connected components of interest, in **stage** 3 we also increase individual connected components by adding new nodes to them. The motivation is to 'grow'

the connected components so that it will be easier to make connections between them. This stage is similar to the 'enhancement' methods used in PRMs [19, 11].

Currently, two different **stage** 3 strategies are implemented in OBPRM. They may be used either independently, or in series.

Stage 3-fp: Expanding failed paths. The motivation behind this first method is that even when connection attempts between connected components fail, they may make some progress which might be useful. The general idea is to save the last valid node c_3 visited on a failed connection attempt between configurations c_1 and c_2, and to add this node to our roadmap:

```
STAGE 3-FP: EXPANDINGFAILEDPATHS
1.  V_i = {v|v ∈ connected component i}, 1 ≤ i ≤ m
2.  cm_i := the center-of-mass of V_i
3.  for i := 1, m /*connected components*/
4.     V_j := component with minimum dist(cm_i, cm_j)
5.     P_{i,j} := CLOSESTPAIRS(V_i, V_j, K3.1)
6.     for each (c_1, c_2) ∈ P_{i,j}
7.        connect(c_1, c_2, c_3) /*c_3 is terminus of failed path*/
8.        if connect fails
9.           then ATTEMPTALL(c_3, V_j, K3.2)
10.       if can't connect c_3 to V_j
11.          then ATTEMPTALL(Nbrs(c_3), V_j, K3.3)
12.       if can't connect any v ∈ Nbrs(c_3) to V_j
13.          then add c_3 to V_i
14.    endfor
15.    ANALYZEROADMAP /*compute connected components*/
16. endfor
```

This method uses the *center-of-mass* to select which connected components to try to connect (the center-of-mass of component V_i is the average of all the configurations in V_i). Next, for each pair (V_i, V_j) of connected components selected, it further selects the `K3.1` closest configuration pairs (c_1, c_2) in $V_i \times V_j$. If we cannot connect c_1 and c_2, we save the last valid node c_3 on this failed path. Then, we attempt to connect c_3 to the `K3.2` closest nodes in V_j. If that fails also, we attempt to connect each of the neighboring configurations $Nbrs(c_3)$ of c_3 to the `K3.2` closest configurations in V_j. If this too fails, then c_3 is added to the roadmap and the process repeats with the next closet pair (c_1', c_2') between V_i and V_j.

This whole process can be repeated until no further progress is obtained. Generally, `K3.1` and `K3.2` should be small constants; we have had success using `K3.1`= 15 and `K3.2` = 10. Our current implementation uses the A^*-clearance(15nbrs,9steps) local planner for the first two connection attempts (lines 7 and 9) and the rotate-at-s planners for the third attempt (line 11).

Stage 3-sc: Expanding small components. The motivation behind this method is that the configurations that are in 'difficult' regions of C-space are likely to be contained in the smaller connected components, and moreover, these small components may prove instrumental as 'bridges' between the larger connected components. The general idea is to expand the small components by generating more configurations near configurations already in that component.

```
STAGE 3-SC: EXPANDINGSMALLCOMPONENTS
1.  V_i = {v|v ∈ connected component i}, 1 ≤ i ≤ m
2.  for i := 1, m /*connected components*/
3.     if |V_i| < MAX3
4.        c := random configuration in V_i
5.        generate Nbrs(c)
6.        for each c' ∈ Nbrs(c)
7.           ATTEMPTALL(c', V_i, K3.3)
8.  endfor
9.  run CONNECTINGCOMPONENTS /* stage 2 */
```

As with the other methods, there are several parameters that must be selected appropriately: `MAX3` (the cut-off for small components) and `K3.3` (the number of configurations to try to connect with each neighbor c'). In addition, one must select the number and type of neighbors one would like to generate, and the local planners to use to attempt those connections. The current implementation uses `MAX3` = 10, `K3.3` = 10, and considers the fifteen neighbors checked by the A^* method. The straight-line planner is used to try to connect each $c' \in Nbrs(c)$ to c's connected component V_i (line 7), and A^*-clearance(15nbrs,9steps) is the planner used in the call to `ConnectingComponents` (the **stage** 2 connection strategy).

5 Evaluating OBPRM

The experiments described below were designed to evaluate the node generation and connection strategies

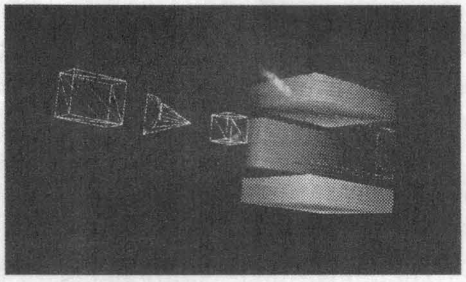


Figure 1: *Corridor 1.*

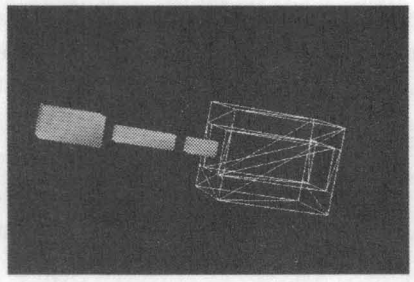


Figure 2: *Corridor 2.*

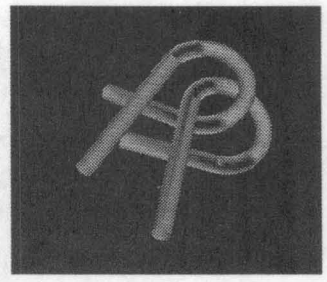


Figure 3: *Alpha puzzle.*

in cluttered Workspaces containing narrow C-space corridors. All programs were written in C, and run on SGI machines.

Environments. We considered two Workspace environments containing easily identifiable C-space corridors, and an interesting puzzle. (See Figures 1–3.)

Workspace *corridor 1* had dimension $2 \times 3 \times 5$, and had three different moving objects (robots): a $1 \times 1 \times 1$ cube (easy, rotates easily in corridor), a $1 \times 1.5 \times 2$ block (hard, cannot rotate freely), and a 'half-cone' that has roughly the same dimensions as the block (moderate).

Workspace *corridor 2* consisted of a tunnel of dimension $3 \times 8 \times 3$ contained in a bounding box of dimension $5 \times 28 \times 6$. Again, we considered three moving objects (robots): a small block of dimension $1 \times 2 \times 1$ (easy), a medium block of dimension $1 \times 4 \times 1$ block (moderate), and a large block of dimension $2 \times 4 \times 2$ block (hard).

We also considered an *alpha puzzle* environment containing two tubes twisted into an α shape (1008 triangles per tube). The objective is to separate the intertwined tubes by a sequence of rotations and translations. The puzzle can be made easier by scaling the obstacle tube in one dimension, which has the effect of shrinking or widening the gap between the two prongs of the α and simultaneously transforming the tube's cross-section from a circle to an increasingly thinner and longer ellipsoid.

5.1 Node generation

The two most important criteria for evaluating a node generation scheme are, first, its ability to generate nodes in difficult regions of C-space, and second, whether the generated nodes can be connected into a roadmap. We used the Corridor 1 environment to evaluate these criteria.

Generating nodes in difficult regions. The nature of the Corridor 1 environments made it easy to determine whether a generated configuration was in the corridor. In particular, using the seed generation strategy of interest, and a chosen number of shells (1, 3, or 5), we first attempted to generate the requested number of nodes (100 per obstacle, or 400 total for shell 1, and 1200 or 2000 total for shells 3 and 5, respectively). Then, each resulting node was classified as being `inside` (all vertices inside the corridor), `border` (some vertices inside and some outside), or `outside`. The values reported are averages of 10 runs.

As expected, it became increasingly difficult to generate `inside` configurations for the cube, the half-cone, and the block. However, the same did not hold for the `border` configurations – which are also very important since they represent the entrance to the corridor. In fact, for all shells, more border configurations were generated for the half-cone than for the cube or block. The reason for this is that the half-cone is longer than the cube and thinner than the block (and thus there are more border configurations possible for it).

Usually the best strategy for at least one of the objects was a combined strategy (i.e., using multiple strategies). This was true even though some of our strategies are more or less indistinguishable for the objects we studied, e.g., random versus weighted triangles for the cube, or extreme versus random vertices for all objects. We suspect this will only be magnified

when more complex objects are studied. However, we did see that certain strategies are preferred for certain types of objects — and these adhere to our expectations as listed in Table 1. In particular, **cM** is good for symmetrical objects like the cube, **eV** and **wT** cover important, and perhaps different, object features, and **rV** and **rT** help assure good coverage.

Our findings seemed to indicate that multiple shells are *not* useful for generating `inside` or `border` configurations. For the cubes the percentage of `inside/border` configurations was approximately 10% in the first shell and between 7% and 8% in the second and third shells; a similar relation existed for the half-cone and block. However, our experience showed that three shells did improve roadmap quality, and thus they should probably be retained since they come for free during node generation.

Generating connectable nodes. The connectivity of the resulting roadmap is the other crucial criterion for judging a node generation scheme. Since we were concerned with our ability to find paths *through* C-space corridors, we pruned the `outside` configurations and built `stage 1` roadmaps using only the `inside` and `border` configurations.[1]

Figure 4 shows, for the cube and half-cone, respectively, four `stage 1` roadmaps built using different seed generation strategies; all roadmaps were constructed using all local planners. Each connected component of a roadmap is represented by a bar showing the number of nodes in that component. Thus, in general, the fewer and the taller the bars of a roadmap, the better it is. Of course, it is also desirable for the large connected components to contain `inside` nodes (shown in black).

In general, the best roadmaps were obtained using the combined strategy **rV+wT**. It can be seen from the figures that using **all** the seed strategies yields roadmaps whose connectivity is approximately an average of the others. Thus, if one knows enough about the shapes of the objects, then some advantage can be gained by selecting node generation strategies suited to those objects. However, if this knowledge is not available, then roadmaps with reasonable connectivity can be obtained by using all the strategies simultaneously. In this case, it might be advisable to first build small roadmaps using the various strategies, and then to select appropriate strategies based on these 'test' roadmaps.

[1]Notice that the pruning may reduce connectivity since we may remove `outside` nodes that 'bridge' different connected components that pass through different C-space corridors corresponding to the one Workspace corridor.

5.2 Connection

The success of the roadmap connection phase depends on both the chosen local planning methods and on the the strategies used to select candidate nodes for connection.

Choosing Local Planners. To test the connectivity of the roadmaps resulting from different (combinations of) local planners, we used the Corridor 1 environments, generated sets of nodes (once), and analyzed the connected components in the `stage 1` roadmaps built using different (combinations of) planners. (See Figure 5.)

Each figure shows the connected components of the roadmap constructed using the different combinations of local planners. It is clear the connectivity of the resulting roadmap is improved by using multiple local planners, and that none of them is totally redundant. That is, there are some cases in which each local planner makes the crucial connections between different connected components. For example, the half-cone roadmaps in Figure 5 show that the straight-line and rotate-at-$\frac{1}{2}$ planners make different connections (Rdmp 3 vs Rdmps 1 and 2), and that the A^*-like planners (Rdmps 5 and 6) were needed to obtain one connected component.

While it is true that connectivity can be greatly enhanced by using multiple local planners, it should be remembered that this implies a larger cost for each connection. Some sample node generation and connection times for the `stage 1` (hybrid) roadmaps environment are shown in Table 2; note that the connection times for the Corridor 1 environments are only for the reduced

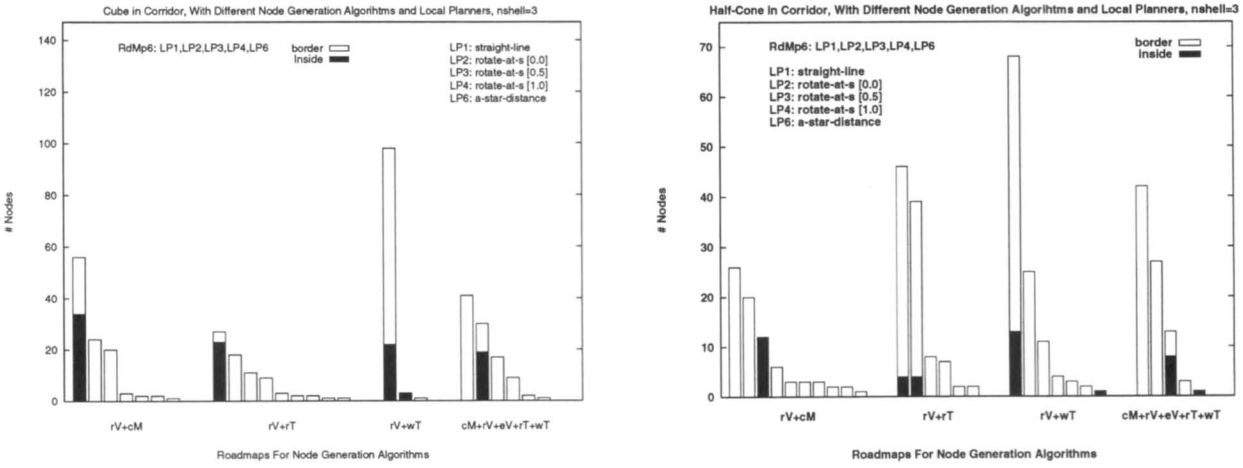


Figure 4: *Node Generation:* **stage 1** *roadmaps (all local planners), corridor 1, cube and half-cone, respectively.*

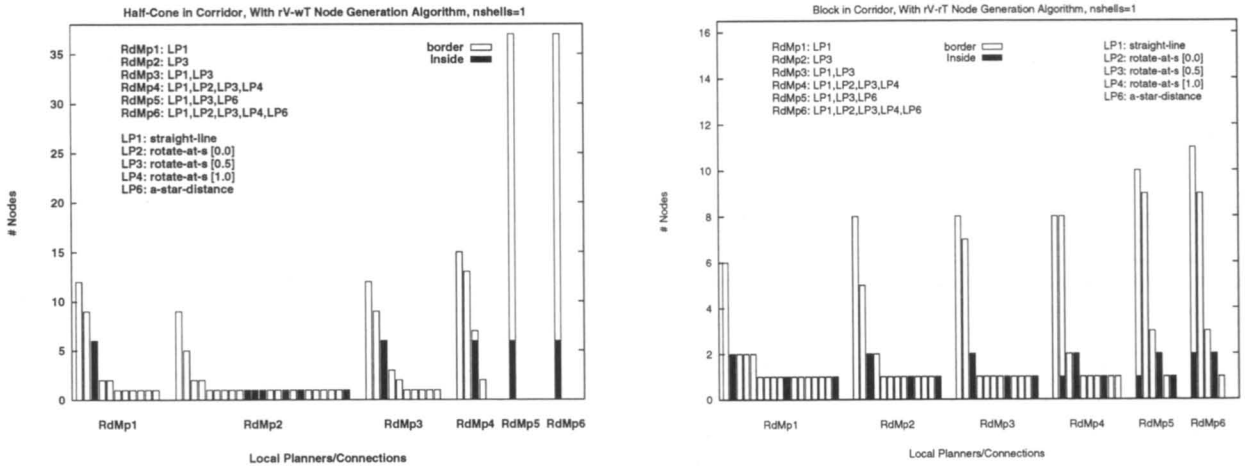


Figure 5: *Local Planners:* **stage 1** *roadmaps, corridor 1, half-cone and block, respectively.*

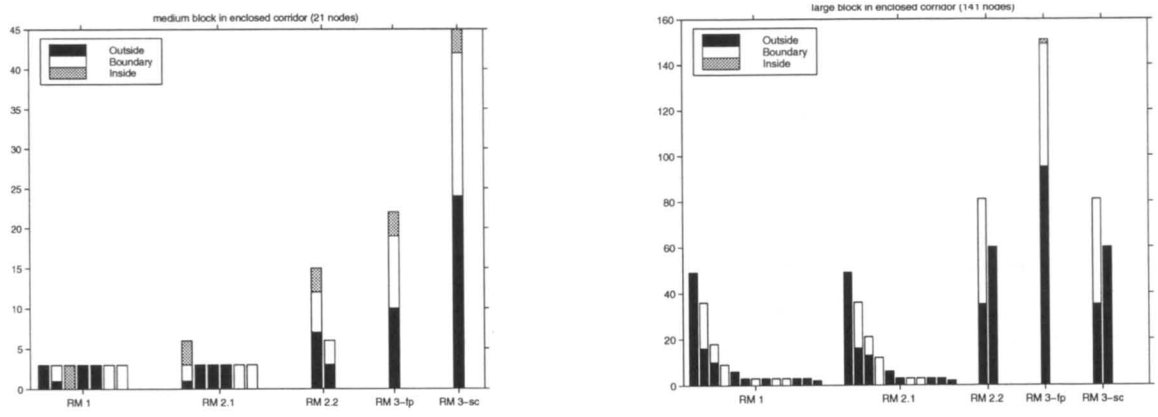


Figure 6: *Connection Stages:* **stage** *1, 2.1, 2.2, 3-fp, 3-sc roadmaps, corridor 2, medium and large block, respectively.*

Roadmap Construction, seed strategy 8 (all), no. shells = 3																
Envt	Generation			Roadmap Connection												
		#nodes	#nodes	Rdmp1		Rdmp2		Rdmp3		Rdmp4		Rdmp5		Rdmp6		
	sec	all	in+bd	sec	#cc	sec	#cc	sec	#cc	sec	#cc	sec	#cc	sec	#cc	
cube	10	872	100	25	21	46	19	49	15	75	7	175	7	199	6	
half-cone	22	834	86	31	12	58	18	61	9	105	6	232	6	269	5	
block	15	736	73	13	23	23	31	28	23	67	10	162	9	188	9	
alpha	2	15	n/a	3	11	4	14	6	10	18	9	30	7	41	7	

Table 2: *Preprocessing computation times and statistics for* `stage 1` *(hybrid) roadmaps.*

Multi-Stage Roadmap Connection (node generation: seed strategy 77, no. shells = 3)												
Envt	Generation		Roadmap Connection									
			Stage 1		Stage 2.1		Stage 2.2		Stage 3-fp		Stage 3-sc	
	sec	#nodes	sec	#cc	sec	#cc	sec	#cc	sec	#cc	sec	#cc
sm blk	17	327	63	12	29	6	257	2	1870	2	76	2
sm blk	47	839	159	20	120	7	645	2	4	1	0	1
md blk	7	21	3	7	14	6	487	2	27	1	96	1
md blk	25	128	32	14	34	12	797	3	330	1	331	1
md blk	36	183	42	19	101	11	709	2	5	1	70	1
lg blk	56	72	13	4	1	4	276	2	7020	2	249	2
lg blk	85	141	23	13	13	11	909	2	2273	1	169	2
lg blk	115	198	29	14	27	13	1409	2	3	1	32	1
lg blk	116	210	23	14	41	13	1114	2	14	1	93	2
alpha	15	591	156	27	31	21	3206	10	25832	7		
alpha	129	4949	1346	47	202	22	3440	17	29637	14		

Table 3: *Preprocessing computation times and statistics for multi-stage roadmap connection.*

set of `inside` and `border` nodes. As can be seen, the A^*-like planner has a large impact on the running time.

Multi-stage Connection. To evaluate our multi-stage connection strategy, we used the Corridor 2 environments. In each case, we first generated nodes, and then performed the various connection stages; the `stage 3` connection methods were each applied to the roadmap produced in `stage 2.2`. Some resulting roadmaps are shown in Figure 6. The roadmap labels correspond to the connection stage that produced it. In almost every case, there were clear connectivity benefits obtained from one stage to the next. For example, even though `stage 2.1` considers the same set of nodes and uses only the straight-line and rotate-at-$\frac{1}{2}$ local planners, it still usually produced a better connected roadmap. The first time the A^* planners are used is in `stage 2.2`. In every case studied, there was a significant improvement in connectivity between the `stage 2.1` and `stage 2.2` roadmaps. This was particularly true as the environments increased in difficulty.

By design, the Corridor 2 environment has a tendency to have two connected components, one on either side of the corridor. For the medium and large blocks, the more sophisticated `stage 3` strategies proved crucial for joining these components. It is interesting to note that the `stage 3` strategies sometimes discovered crucial **inside** nodes that enable them to connect the two components (e.g., for the large block).

Node generation and connection times for the various connection stages are shown in Table 3. As expected, `stage 1` and `stage 2.1` are generally the fastest since they employ only the faster local planners (straight-line and rotate-at-$\frac{1}{2}$). It can also be seen that the time spent in `stage 1` depends on the number of nodes, and the time spent in `stage 2.1` depends on the number of connected components. The times spent in `stage`

2.2 and stage 3 are generally significantly greater due to the A^*-like planners. Thus, while it is clear that these methods can be highly effective, they must be used with care in order for the resulting methods to be computationally feasible.

Alpha-puzzle. The hardest problem studied in this paper is the alpha-puzzle. This problem illustrates that different problems require different types of node generation strategies. A representative roadmap for this problem must include both configurations in which the two tubes are intertwined and separated. Through experimentation, we found that such configurations are most easily generated using a strategy of **rV+wT** for one tube and **cM** for the other for the *free nodes*; using this strategy, the generated nodes are split roughly 80/20 between intertwined and separated configurations. In contrast to the corridor environments, the contact configurations did not seem to be as useful here.

At present, OBPRM has succeeded in solving a version of the problem in which the obstacle tube is scaled by 1.2 in the z-direction (perpendicular to the tube's cross-section). This version was solved using a stage 1 roadmap. Although, it is a rather close fit, the gap between the prongs of the α in this version is large enough to enable the other tube to escape by 'sliding' out (rather than the complicated sequence of rotations and translations needed to solve the original version). Currently, we are working on this problem using our multi-stage connection strategy. Based on our preliminary results we believe we will be able to solve (at least) the version in which the obstacle tube is scaled by 1.1.

6 Conclusion

In this paper, we describe and evaluate several strategies for node generation and propose a multi-stage connection strategy for OBPRM in cluttered 3-dimensional Workspaces.

Our evaluation of various node generation strategies in terms of their ability to produce nodes in difficult regions of C-space, and in terms of the connectivity of the resulting roadmaps, yields recommendations for selecting appropriate combinations of node generation strategies for different types of objects (robot or obstacle). When the objects cannot be classified easily, we recommend a default strategy which is essentially a combination of all the others. While it will likely not be the best strategy in any given situation, it seems to give reasonable results for the problems we've studied. Finally, when the relative importance of the contact configurations and the free configurations cannot be determined *a priori*, we recommend generating roughly half of each.

We also show that roadmap connectivity can be greatly improved by using a multi-level connection strategy and multiple local planners. However, this improvement in connectivity does require significant computation. Although our study was aimed at OBPRM, we believe our techniques and conclusions will be useful for PRMs in general, and for other motion planning approaches as well.

Currently, we are working to make OBPRM adapt automatically to the environment (e.g., automatically selecting seed generation strategies), and are further refining the connection strategies described here. We are also planning to augment our implementation for planning contact tasks.

Acknowledgements

This research was supported in part by NSF CAREER Award CCR-9624315 (with REU Supplement), NSF Grant IRI-9619850, and by the Texas Higher Education Coordinating Board under Grant ARP-036327-017. O. Burchan Bayazit is supported in part by the Turkish Ministry of Education. Lucia K. Dale is supported in part by a GE Foundation Graduate Fellowship. Daniel Vallejo is on leave from Universidad de las Americas-Puebla, Mexico and is supported in part by a Fulbright-CONACYT scholarship.

We would like to thank the robotics group at Texas A&M, especially Jeff Trinkle and Li Han, for their suggestions regarding this work. We are also grateful to John Canny, Shane Chang, Lydia Kavraki, and Jean-Claude Latombe for useful discussions and comments. The alpha puzzle was designed by Boris Yamrom of the

Computer Graphics & Systems Group at GE's Corporate Research & Development Center. GE also provided us with *Product Vision* (their CAD animation package) which was used to produce the environment snapshots shown in this paper, and our collision detection routine is based on code developed at GE.

References

[1] J. M. Ahuactzin and K. Gupta. A motion planning based approach for inverse kinematics of redundant robots: The kinematic roadmap. In *Proc. IEEE Internat. Conf. Robot. Autom.*, pages 3609–3614, 1997.

[2] N. M. Amato, O. B. Bayazit, L. K. Dale, C. V. Jones, and D. Vallejo. Choosing good distance metrics and local planners for probabilistic roadmap methods. In *Proc. IEEE Internat. Conf. Robot. Autom.*, 1998. To appear.

[3] N. M. Amato and Y. Wu. A randomized roadmap method for path and manipulation planning. In *Proc. IEEE Internat. Conf. Robot. Autom.*, pages 113–120, Minneapolis, MN, April 1996.

[4] J. Barraquand and J.-C. Latombe. Robot motion planning: A distributed representation approach. *Internat. J. Robot. Res.*, 10(6):628–649, 1991.

[5] D. J. Challou, M. Gini, and V. Kumar. Parallel search algorithms for robot motion planning. In *Proc. IEEE Internat. Conf. Robot. Autom.*, volume 2, pages 46–51, 1993.

[6] H. Chang and T. Y. Li. Assembly maintainability study with motion planning. In *Proc. IEEE Internat. Conf. Robot. Autom.*, pages 1012–1019, 1995.

[7] P. C. Chen and Y. K. Hwang. SANDROS: A motion planner with performance proportional to task difficulty. In *Proc. IEEE Internat. Conf. Robot. Autom.*, pages 2346–2353, 1992.

[8] B. Glavina. Solving findpath by combination of directed and randomized search. In *Proc. IEEE Internat. Conf. Robot. Autom.*, pages 1718–1723, 1990.

[9] T. Horsch, F. Schwarz, and H. Tolle. Motion planning for many degrees of freedom – random reflections at c-space obstacles. In *Proc. IEEE Internat. Conf. Robot. Autom.*, pages 3318–3323, 1994.

[10] D. Hsu, L. Kavraki, J.-C. Latombe, R. Motwani, and S. Sorkin. On finding narrow passages with probabilistic roadmap planners. In *Proc. Intern. Workshop on Algorithmic Foundations of Robotics*, 1998. These proceedings.

[11] D. Hsu, J.-C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. In *Proc. IEEE Internat. Conf. Robot. Autom.*, pages 2719–2726, 1997.

[12] Y. K. Hwang and N. Ahuja. Gross motion planning – a survey. *ACM Computing Surveys*, 24(3):219–291, 1992.

[13] Y. K. Hwang and N. Ahuja. A potential field approach to path planning. *IEEE Trans. Robot. Automat.*, 8(1):23–32, 1992.

[14] Y. K. Hwang and P. C. Chen. A heuristic and complete planner for the classical mover's problem. In *Proc. IEEE Internat. Conf. Robot. Autom.*, pages 729–736, 1995.

[15] L. Kavraki, M. Kolountzakis, and J.-C. Latombe. Analysis of probabilistic roadmaps for path planning. In *Proc. IEEE Internat. Conf. Robot. Autom.*, volume 4, pages 3020–3025, 1996.

[16] L. Kavraki, F. Lamiraux, and C. Holleman. Towards planning for elastic objects. In *Proc. Intern. Workshop on Algorithmic Foundations of Robotics*, 1998. These proceedings.

[17] L. Kavraki and J. C. Latombe. Randomized preprocessing of configuration space for fast path planning. In *Proc. IEEE Internat. Conf. Robot. Autom.*, pages 2138–2145, 1994.

[18] L. Kavraki, J. C. Latombe, R. Motwani, and P. Raghavan. Randomized query preprocessing in robot path planning. In *Proc. ACM Symp. Theory of Computing*, pages 353–362, 1995.

[19] L. Kavraki, P. Svestka, J. C. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. Automat.*, 12(4):566–580, August 1996.

[20] J. C. Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, Boston, MA, 1991.

[21] M. Overmars. A random approach to path planning. Technical Report RUU-CS-92-32, Computer Science, Utrecht University, The Netherlands, 1992.

[22] M. Overmars and P. Svestka. A probabilistic learning approach to motion planning. In *Proc. Workshop on Algorithmic Foundations of Robotics*, pages 19–37, 1994.

[23] J. Reif. Complexity of the piano mover's problem and generalizations. In *Proc. IEEE Symp. Foundations of Computer Science*, pages 421–427, 1979.

[24] P. Watterberg, P. Xavier, and Y. Hwang. Path planning for everyday robotics with sandros. In *Proc. IEEE Internat. Conf. Robot. Autom.*, pages 1170–1175, 1997.

[25] Y. Wu. An obstacle-based probabilistic roadmap method for path planning. Master's thesis, Department of Computer Science, Texas A&M University, 1996.

Subdivision: A Technique for Modeling Curved Shape

Joe Warren, *Department of Computer Science, Rice University*

Subdivision is an iterative process that maps an initial coarse shape into finer and finer shapes. If the subdivision process is carefully chosen, then the limit of this process is a smooth, curved shape that roughly follows the initial coarse shape. Due to this correlation, subdivision has proven to be an increasingly popular method for representing curved shape. This paper first reviews two related modeling techniques: functions and fractals. The paper then gives a brief introduction to subdivision viewed as a fusion of these two techniques.

1 Introduction

Geometric modeling examines the fundamental problem: What is a good representation for complicated, curved shape? Although there are many different approaches, most methods fall into one of two categories: functions or fractals.

1.1 Functions

Functions map a parameter t to a function value $F(t)$. Plotting the graph of a smooth function F over finite range defines a shape,

$$x = F(t).$$

In figure 1, the function $sin(t)$ defines a wave-like shape.

A pair of functions can be combined to define a shape parametrically.

$$(x, y) = (F_x(t), F_y(t)).$$

For example in figure 2, the parameterization $(cos(t), sin(t))$ defines a circle. Alternatively, a curve

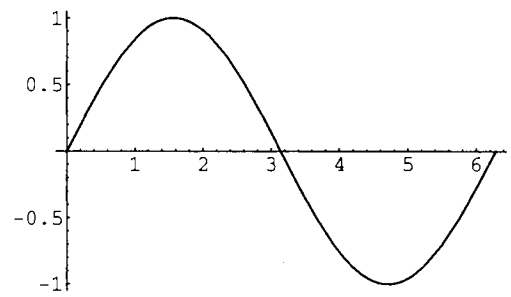


Figure 1: *A plot of $sin(t)$.*

can be defined implicitly as the zero contour of a bivariate variate function F,

$$F(x, y) = 0.$$

For example, a circle is also the solution to the equation $x^2 + y^2 = 1$.

To use functions effectively in modeling shape, we must address several important questions. First, what space of functions lead to a rich, but computationally tractable space of shapes? As we shall see, polynomial functions are one answer. Second, how can the shape defined by these functions be related to their representation? For this question, an appropriate choice of basis leads to an elegant representation.

1.2 Fractals

The construction of fractals requires two ingredients, a space of geometric shapes G and a transformation S that maps a shape in G to new shape in G. Given an

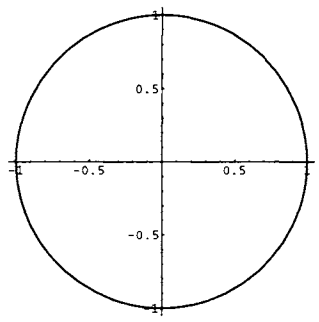


Figure 2: *A circle defined parametrically using cos(t) and sin(t).*

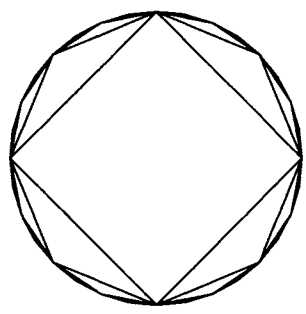


Figure 3: *A circle defined as the limit of 2^k-gons.*

initial shape g_0, let

$$g_{k+1} = S(g_k),$$

define an infinite sequence of shapes. The fractal associated with initial shape g_0 is the limit of the shape g_k as $k \to \infty$. For example, consider the function S that maps to a regular 2^k-gon to a regular 2^{k+1}-gon. The limit of these polygons as k goes to infinity is a circle. Figure 3 illustrates this process.

One particularly appealing type of fractal are those produced by iterated affine transformations. For this fractal, possible shapes in G are simply point sets. Possible transformations S are sets of affine transformations. If these transformations are restricted to contractive transformations (i.e. affine transformation

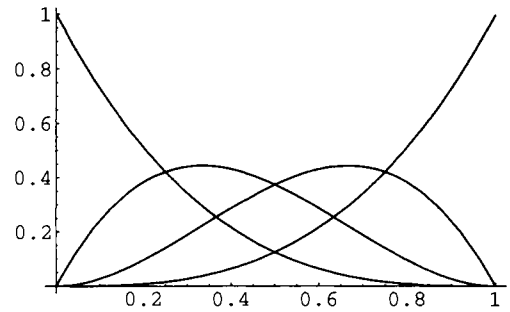


Figure 4: *The cubic Bernstein basis functions.*

whose scaling components are less than one), the fractal process is guaranteed to produce a well-defined limit shape.

2 Functions: Polynomials

Polynomial functions have a simple computational representation. Analytical and numerical properties of polynomials are well-understood. The key question is to relate the representation of the polynomial to the shape of its graph. As the next section shows, the answer is to choose the correct basis. (For a more complete introduction, see Farin [6].)

2.1 Polynomials over an interval

The Bernstein basis is one example of a basis that provides a geometric connection between the representation for a polynomial (its coefficients) and the shape of its graph. If we restrict our attention to the unit interval $[0, 1]$, then the Bernstein basis functions of degree n satisfy

$$B_i^n(t) = \frac{n!}{i!(n-i)!}(1-t)^{(n-i)}t^i.$$

These basis functions form a sequence of bump functions over the unit interval. The ith basis function reaches its maximum at $t = \frac{i}{n}$. Figure 4 shows the graphs of the cubic Bernstein basis functions on the interval $[0, 1]$.

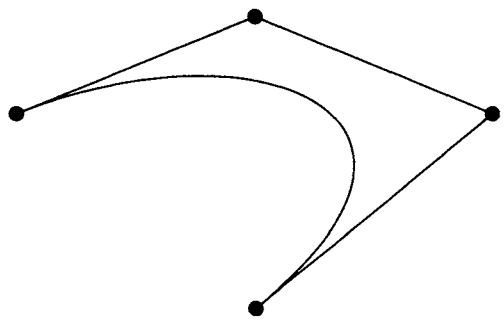


Figure 5: *A control polygon and its associated Bezier curve.*

A shape is specified by a coarse approximation that consists of a sequence of points, $p_0, p_1, ..., p_n$. Connecting these points with line segments yields a control polygon $\mathbf{p}$ [1]. The Bezier curve $F(t)$ associated with this control polygon $\mathbf{p}$ is the parametric curve form satisfying

$$F(t) = \sum_{i=0}^{n} p_i B_i^n(t).$$

Figure 5 shows an example of a control polygon and its associated Bezier curve. The Bezier curve has many nice properties. In particular, the curve

- Follows its control polygon,
- Interpolates the endpoints of its control polygon,
- Lies in the convex hull of its control polygon,
- Is independent of coordinate system,
- Is variation diminishing. (Wiggles no more than its control polygon does.)

One drawback of Bezier curves is that the entire curve is defined by a single polynomial function. If the number of control points is small, then this property poses no problem. However, if the number of control points is large, then manipulating the underlying polynomial function can become expensive. The solution to this problem is to use a piecewise polynomial representation.

[1] Boldface variables denote vectors.

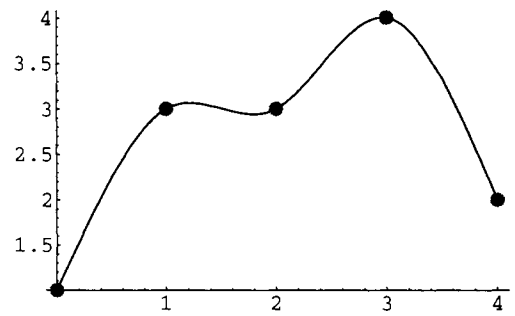


Figure 6: *A natural cubic spline.*

2.2 Piecewise polynomials over several intervals

Consider the variational problem of minimizing the expression

$$\int_{t_0}^{t_m} (\frac{\partial^2 F(t)}{\partial t^2})^2 dt,$$

over all "smooth" functions $F(t)$ such that $F(t_i) = v_i$ at a set of knots $T = \{t_0, ..., t_m\}$. The resulting minimizing functions are the C^2 piecewise cubic functions, the natural cubic splines. These functions interpolate the values v_i without any extraneous "wiggles". Figure 6 depicts the natural cubic spline interpolating the values $\mathbf{v} = (1, 3, 3, 4, 2)$ at the parameter values $T = \{0, 1, 2, 3, 4\}$, respectively.

The solution to this variational problem satisfies the differential equation

$$\frac{\partial^4 F(t)}{\partial t^4} = 0 \quad \forall t \notin T.$$

Due to this equation, it is clear that the minimizing functions must be piecewise cubic polynomials. One particularly convenient basis for this space is cubic B-spline basis. These basis functions are the piecewise C^2 cubics of minimal support. The basis functions are supported over 3 consecutive intervals. Figure 7 depicts the five basis functions for the previous example.

As in the case of Bezier curves, a B-spline curve $F(t)$ is a linear combination of a set of control

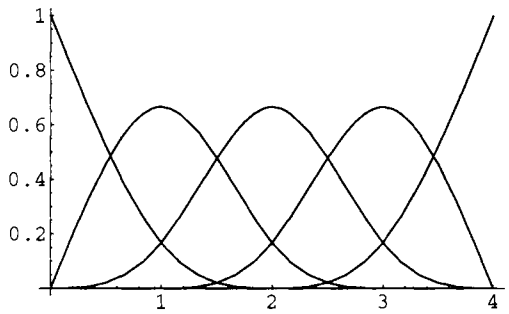


Figure 7: *B-spline basis functions.*

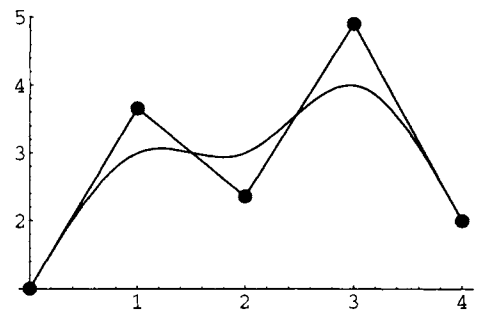


Figure 8: *The B-spline representation for the example.*

points $\mathbf{p} = (p_0, ..., p_n)$ times B-spline basis functions $N_0(t), ..., N_n(t)$,

$$F(t) = \sum_{i=0}^{n} p_i * N_i(t).$$

(Note that the B-spline basis functions depend on the knot set T.)

The B-spline basis functions are not interpolating. To satisfy the interpolation conditions, we must find a set of initial control points $\mathbf{p}$ such that $F(t)$ satisfies the interpolation conditions. If C is the matrix that consists of the values of the B-spline basis functions at the various knots T, then the control polygon $\mathbf{p}$ whose corresponding B-spline curve interpolates the values $\mathbf{v}$ at the knots of T satisfies

$$\mathbf{p} = C^{-1}\mathbf{v}.$$

Continuing our previous example,

$$C = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ \frac{1}{6} & \frac{2}{3} & \frac{1}{6} & 0 & 0 \\ 0 & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} & 0 \\ 0 & 0 & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

Figure 8 shows the resulting control polygon and its associated B-spline curve. B-spline curves also have numerous nice properties. The curve

- Follows its control polygon,
- Interpolates the endpoints of its control polygon,
- Lies in the convex hull of its control polygon,
- Is independent of coordinate system,
- Is variation diminishing.

3 Fractals: Iterated Affine Transformations

Affine transformations of the plane include common transformations such as rotation, translation, and scaling. These transformations can be represented by a 3 by 3 matrix. This matrix can be viewed as mapping the three vertices of a triangle $p_0p_1p_2$ to the three vertices of a new triangle $\hat{p}_0\hat{p}_1\hat{p}_2$. If the rows of this transformation matrix sum to one, then this transformation is independent of the particular coordinate system. For example, the transformation matrix

$$\begin{pmatrix} 1 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} \end{pmatrix} \begin{pmatrix} p_0 \\ p_1 \\ p_2 \end{pmatrix} = \begin{pmatrix} \hat{p}_0 \\ \hat{p}_1 \\ \hat{p}_2 \end{pmatrix}$$

affinely maps the white triangle into the black triangle in figure 9.

Each step of an iterated affine transformation consists of applying a set of affine transformations S to a set of triangles g_k to defied a new set of triangles g_{k+1},

$$g_{k+1} = S(g_k).$$

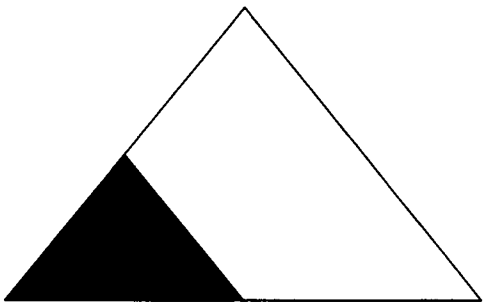


Figure 9: *An affine transformation.*

The limit of the g_k as $k \to \infty$ is the associated fractal. (See [1] for more details on convergence of this process.) Figures 10, 11 and 12 depict several steps in this process for three well-known shapes. The fractal of figure 12 is actually a quadratic Bezier curve. This observation hints at the fact that polynomials can be cast as fractals.

4 Subdivision: Functions as Fractals

Subdivision recasts polynomial functions into the framework of fractals. The geometry is simple: polygons and polyhedra. The transformations are matrices.

4.1 Subdivision matrices

B-splines relate a control polygon $\mathbf{p}_0$ and a knot set T_0 to a spline $F(t)$. Due to its piecewise polynomial nature, the same function $F(t)$ can also be expressed as new control polygon $\mathbf{p}_1$ with respect to a denser knot set T_1. The control polygons $\mathbf{p}_0$ and $\mathbf{p}_1$ are related by the change of basis (subdivision) matrix S_0

$$\mathbf{p}_1 = S_0 \mathbf{p}_0.$$

For example, if $T_0 = \{0,1,2,3,4\}$ and $T_1 = \{0, \frac{1}{2}, 1, \frac{3}{2}, 2, \frac{5}{2}, 3, \frac{7}{2}, 1\}$, then the subdivision matrix S_0 for cubic B-splines has the form

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{8} & \frac{3}{4} & \frac{1}{8} & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{8} & \frac{3}{4} & \frac{1}{8} & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{8} & \frac{3}{4} & \frac{1}{8} \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

If the knots sets T_k are 2^{-k} dilates of the integers, then the associated subdivision matrices S_k degenerate to a single common subdivision matrix S. Each column of S is the shift of a fundamental sequence. For cubic B-splines, this sequence is a multiple of the binomial sequence $(1,4,6,4,1)$. Due to this binomial structure, the action of the matrices S on $\mathbf{p}$ can be expressed as discrete convolution.

4.2 The fractal process

Given an initial set of control points $\mathbf{p}_0$, the subdivision matrices S_k defined an increasingly dense sequence of control polygons $\mathbf{p}_k$,

$$\mathbf{p}_{k+1} = S_k \mathbf{p}_k.$$

If $F_k(t)$ is the piecewise linear function associated with the control polygon $\mathbf{p}_k$, then the limit of this process is the function $F(t)$ such that

$$\lim_{k \to \infty} F_k(t) = F(t).$$

Thus, this fractal process associates a function $F(t)$ with an initial set of coefficients $\mathbf{p}_0$.

In our running example, the initial knot set T_0 was $\{0,1,2,3,4\}$. If the subsequent knot sets T_k are restrictions of 2^{-k} dilates of the integers to the interval $[0,4]$, then the two steps of subdivision (as shown in figure 13) results in a control polygon $\mathbf{p}_2$ that is indistinguishable from the actual cubic B-spline $F(t)$.

4.3 Properties of subdivision schemes

Given an arbitrary sequence of subdivision matrices S_k, the basis functions associated with this subdivision

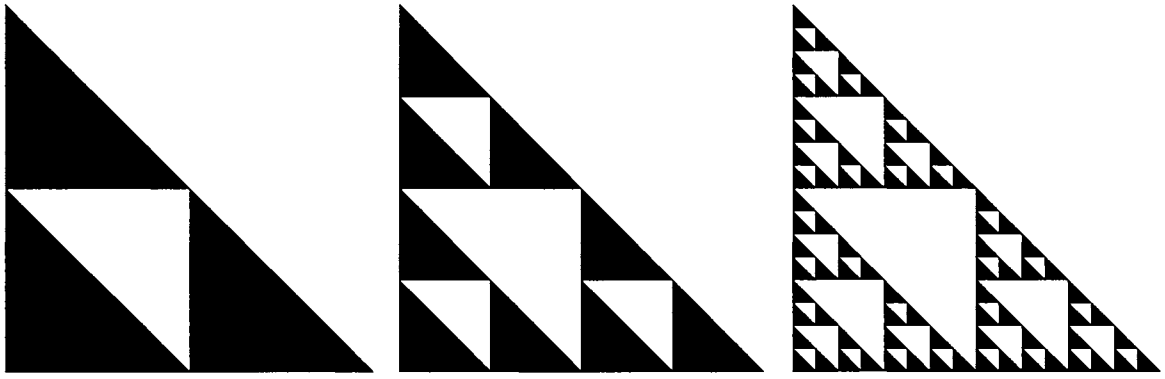


Figure 10: *The Serpenski triangle.*

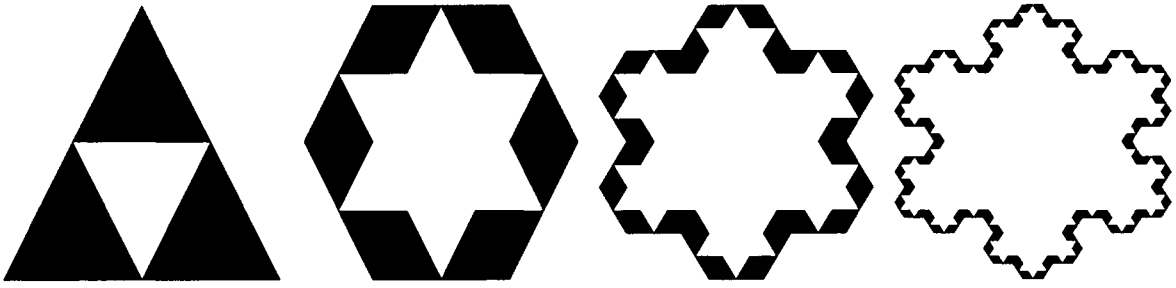


Figure 11: *The Koch curve.*

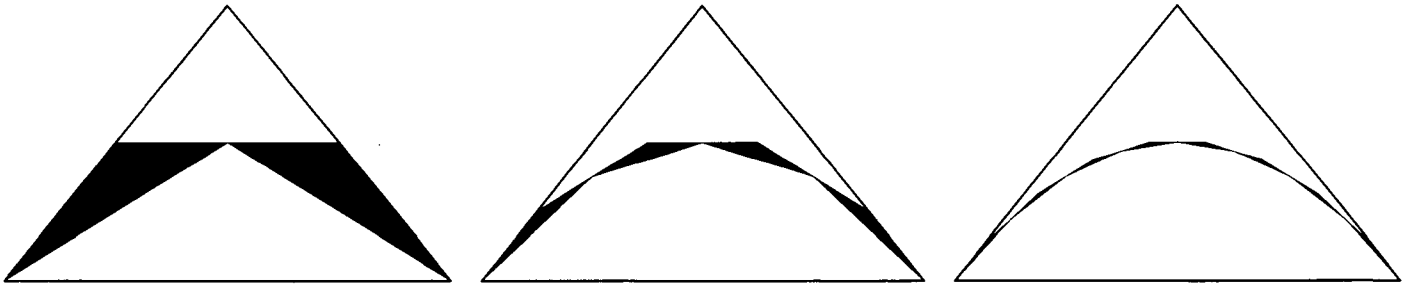


Figure 12: *A quadratic Bezier curve as a fractal.*

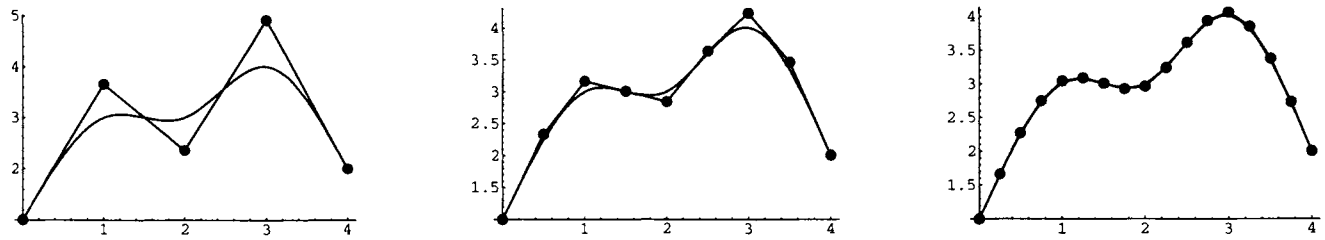


Figure 13: *Subdivision of a cubic B-spline.*

scheme can be computed by setting the initial coefficient vector $\mathbf{p}_0$ to be a basis vector. If the rows of S_k are locally supported, then this basis function is also locally supported. If the rows of S_k sum to one and are non-negative, then subdivision scheme is coordinate independent and satisfies the convex hull property.

For arbitrary S_k, no closed form expression for the limit function $F(t)$ may exist. However, it is still possible to compute many important quantities associated with $F(t)$ directly from $\mathbf{p}_0$ and the subdivision matrices S_k. These quantities include:

- The value of $F(t)$ at a knot of T_k,
- The value of derivatives of $F(t)$ at a knot of T_k,
- The inner products of two limit functions $F(t)$ and $G(t)$ defined using the S_k.

4.4 Research trends

Most research on subdivision addresses one of two main problems:

Convergence/smoothness of schemes
Given a sequence of subdivision matrices S_k, do the associated functions $F_k(t)$ converge to a limit function? If so, does the limit function have continuous derivatives?

For uniform knot sequences, the subdivision matrices S_k are independent of k. The subdivision scheme satisfies $\mathbf{p}_{k+1} = S\mathbf{p}_k$. Spectral analysis can be used to analyze the behavior of the limit process [3]. If the knot sequences T_k vary non-uniformly, then a pure matrix approach can be used to compute subdivision schemes for the various derivatives associated with the scheme [4]. Testing for convergence of these derivatives then involves verifying that the differences of these derivatives converge uniformly to zero.

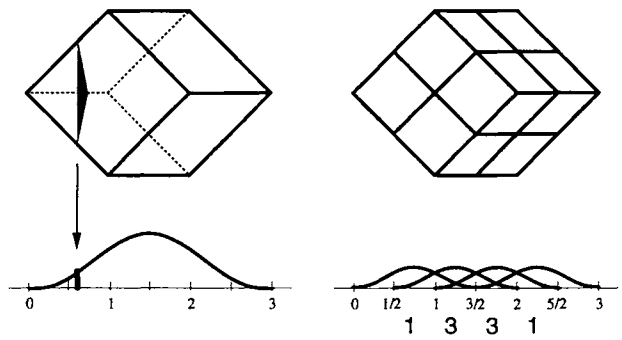


Figure 14: *Subdivision of a quadratic B-spline basis function.*

Construction of "interesting" schemes
Smooth, locally supported polynomial functions can be constructed as the projection of cross-sectional volumes of d-dimensional cubes. Splitting these cubes into 2^d subcubes yields a subdivision scheme associated with these functions. Figure 14 depicts a quadratic B-spline basis function as the cross-sectional area of 3-cube projected onto the line. Splitting this cube into 8 subcubes yields a subdivision formula for quadratic B-splines. If the cross-sectional volumes are projected into two or more dimensions, the resulting splines are known as box splines [4].

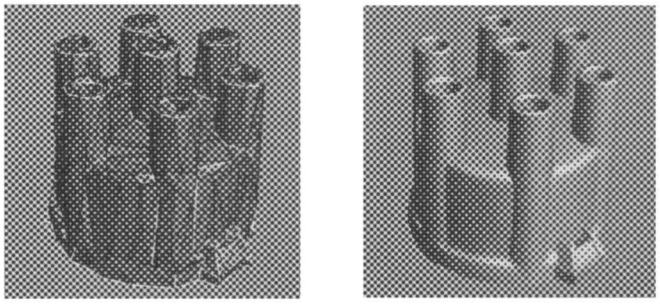


Figure 15: *A polyhedron and its corresponding subdivision surface.*

Box splines require the underlying knot sets to be uniform. In two dimensions, this restriction forces triangular surfaces to have six triangles meeting at every vertex (or four squares at every vertex). Unfortunately, only surfaces of genus one satisfy this restriction. Several modified versions of box splines exists that allow for the existence of vertices that do not satisfy these valence restrictions (extraordinary vertices) [5, 2]. The resulting subdivision scheme produce C^1 parametric subdivision surfaces of arbitrary genus [8]. Recent methods even allow for the controlled incorporation of tangent plane discontinuities into the limit surface [7]. (See figure 15 for an example).

Finally, polynomial functions are not the only functions that possess subdivision schemes. The solution to various variational problems can be expressed as the limit of fairly simple subdivision schemes. For example, thin-plate splines, a favorite tool for scattered data interpolation, can be defined via subdivision [9].

5 Summary

Fractals and functions are two important techniques for representing curved shape. Subdivision represents a fusion of these ideas in which functions are represented as fractals. The beauty of subdivision is that very simple rules can be use to define a wide range of curved shapes (including many well-known functional shapes). Applying the fractal process with these rules provides a fast method for linear approximation of the final, curved limit shape. This linear approximation can then be used as the input to most algorithms for rendering, processing and analyzing the curved shape.

References

[1] M. Barnsley. *Fractals Everywhere.* Academic Press, 1993.

[2] E. Catmull and J. Clark. Recursively generated b-spline surfaces on arbitrary topological meshes. *Computer-aided Design*, 10:350–355, 1978.

[3] A. Cavaretta, W. Dahmen, and C Micchelli. *Stationary Subdivision*, volume 453 of *Memoirs of the AMS.* AMS, 1991.

[4] C. de Boor, K. Höllig, and S. Riemenschneider. *Box Splines.* Springer Verlag, 1993.

[5] D. Doo and M. Sabin. Behavior of recursive subdivision of surfaces near extraordinary points. *Computer-aided Design*, 10:356–360, 1978.

[6] Gerald Farin. *Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide.* Academic Press Inc., New York, 1988.

[7] H. Hoppe, T. DeRose, T. Duchamp, M. Halstead, H. Jin, J. McDonald, J. Schweitzer, and W. Stuetzle. Piecewise smooth surface reconstruction. *Computer Graphics*, 28:295–302, 1994.

[8] U. Reif. A unified approach to the analysis of subdivision schemes. *Computer Aided Geometric Design*, 12:153–174, 1995.

[9] Joe Warren. Subdivision schemes for variational problems. submitted to *Computer-aided Geometric Design*, 1997.

Spherical Shell: A Higher Order Bounding Volume for Fast Proximity Queries

Shankar Krishnan, *AT&T Research Labs, Florham Park, NJ, USA*
Amol Pattekar, *University of North Carolina, Chapel Hill, NC, USA*
Ming C. Lin, *University of North Carolina, Chapel Hill, NC, USA*
Dinesh Manocha, *University of North Carolina, Chapel Hill, NC, USA*

Hierarchical data structures have been widely used to design efficient algorithms for interference detection for robot motion planning and physically-based modeling applications. Most of the hierarchies involve use of bounding volumes which enclose the underlying geometry. These bounding volumes are used to test for interference or compute distance bounds between the underlying geometry. The efficiency of a hierarchy is directly proportional to the choice of a bounding volume. In this paper, we introduce spherical shells, a higher order bounding volume for fast proximity queries. Each shell corresponds to a portion of the volume between two concentric spheres. We present algorithms to compute tight fitting shells and fast overlap between two shells. Moreover, we show that spherical shells provide local cubic convergence *to the underlying geometry. As a result, in many cases they provide faster algorithms for interference detection as compared to earlier methods. We also describe an implementation and compare it with other hierarchies.*

1 Introduction

The problems of interference detection and distance computation between static or dynamic objects are fundamental to robotics and computer simulated environments. Given two or more geometric models, composed of polygonal or curved boundaries, the goal is to check whether they overlap or to compute the minimal distance between them. As an integral part of robot motion planning or collision avoidance, the geometric reasoning system that detects potential contacts and determines the amount of separation between the moving object(s) and the obstacles in the workspace serves as a good indicator for fast proximity queries.

These problems have been well studied in the literature and numerous solutions have been proposed. Especially in the case of convex polytopes or objects that can be represented as union of convex polytopes, efficient algorithms have been presented in [16, 29, 7, 4]. They work quite well for polygonized manipulators in a simple simulated environment. However, in different applications using motion planning techniques, including virtual prototyping [10], haptic rendering [35] and molecular modeling [15], models composed of tens or hundred of thousands of primitives (e.g. polygons, spline patches) are very common. Furthermore, in many CAD applications, the input models come as unstructured models or polygons soups [30] and it is very hard to decompose them into convex pieces. As a result, most planning algorithms use hierarchical techniques for proximity queries. Some of the simplest hierarchies are based on the use or variants of spheres or axis-aligned bounding boxes [34, 18]. These bounding volumes are used for simple rejection tests and work well when the objects are far away.

However, in several instances such as those encountered in maintainability study systems, objects are in close proximity of each other. As a result, a *very significant* fraction of the motion planning algorithm based on potential field approaches [25, 22] or global methods [9], is spent on collision detection or distance computation routines. According to Latombe [26], in many cases, 90% of the overall time of a path planner based on potential field techniques is spent on distance computation. This can correspond to tens of CPU hours for assembly maintainability study on large CAD models [10].

Recently, Gottschalk et al. [17] have proposed the use of oriented bounding boxes (OBBs) for fast inter-

ference detection. In terms of shape approximation, OBBs provide local quadratic convergence to the underlying geometry, as compared to linear convergence for hierarchies based on sphere or axis-aligned bounding boxes. Consequently, for many close proximity situations, higher order bounding volumes like OBBs offer better overall performance for proximity queries.

Main contribution: In this paper, we introduce hierarchies based on *spherical shells* and use them for proximity queries. A spherical shell corresponds to a portion of the volume between two concentric spheres and encloses the underlying geometry. We present efficient algorithms to compute such volumes and fast overlap tests between two shells. The overlap test is only slower by a small factor as compared to other bounding volumes, e.g. 2 or 3 times in case of OBBs. Their main advantage comes from the fact that they provide local *cubic* convergence to the underlying geometry. By "cubic convergence" we mean that the bounding volume approximates the surface accurately up to the second order (if the surface is expressed as a Taylor series). Therefore, there are fewer "false positives" in terms of overlap tests between the bounding volumes and the underlying primitives. This results in improved overall performance for proximity queries between two objects in close proximity configurations or between two objects with high-curvature surfaces. In practice, the local cubic convergence is restricted to elliptic (unlike hyperbolic) regions of the surface.

Organization: The rest of the paper is organized in the following manner. We survey related work in Section 2 and give an overview of our approach in Section 3. An efficient algorithm to compute the spherical shell is presented in Section 4. We analyze the asymptotic performance of a spherical shell as a bounding volume in Section 5. Section 6 highlights a fast overlap test between two shells. Finally, in Section 7 we describe our implementation and compare its performance with other bounding volumes. We conclude in Section 8.

2 Related Work

The problem of interference detection has been widely studied in computational geometry, robotics and computer simulated environments. Most of the earlier work in robotics and in computational geometry has focussed on collision detection between convex polytopes. Good theoretical and practical approaches based on linear complexity of the linear programming problem are known as well [32, 38]. Using the Minkowski difference and convex optimization techniques, algorithms with expected linear time performance are also given in [16] to compute the distance between convex polytopes and keep track of closest features. Successive spherical approximations (SSA) [4] were introduced in the rapid detection of collisions in a dynamic environment like a robot being tested for collisions with obstacles along a specified path. The SSA representation is similar to the spherical shells presented in this paper. However, there are a number of significant differences in the two approaches. In the SSA approach, the geometry of the underlying surface is restricted to be topologically closed polygonal objects and the hierarchy computation is restricted to exactly four levels. This restriction causes the collision detection algorithm to be inefficient if the object contains a large number of faces. Further, the collision detection scheme of the SSA approach is very conservative and handles only the trivial accepts and rejects. The complicated cases are resolved further down the hierarchy. Last, but not the least, the parameters required to represent the bounding volume are essentially different (the SSA bounding volume has a pyramidal structure, whereas we use a conical representation).

Using hierarchical representations, an $O(log^2 n)$ algorithm is given in [12] for polytope-polytope overlap problem, where n is the number of vertices. As for curved objects, many algorithms are presented for models whose trajectory can be expressed as a closed form function of time [19]. These methods use either subdivision methods, interval arithmetic or bounds on derivatives and constrained minimization. However, they are fairly slow in practice and not adequate for most of online planning algorithms or interactive dynamic simulation.

In applications involving rigid motion, geometric coherence has been utilized to design algorithms for convex polyhedra based on local features [2, 29, 28]. Local

properties have been used in the earlier motion planning algorithms by [13, 31] when two objects come into contact. These algorithms utilize the spatial and temporal coherence between successive instances and work well in practice.

Most environments consists of multiple objects and performing $O(n^2)$ pairwise interference detection becomes a bottleneck for large n. Algorithms of complexity $O(nlog^2n+m)$ have been presented for spheres in [20] and rectangular bounding boxes in [14], where m corresponds to the number of overlaps. More recently, Cohen et al. have presented algorithms and a system, I-COLLIDE, based on spatial and temporal coherence, for large environments composed of multiple moving objects [11].

A number of hierarchies have been used for collision detection between general polygonal models. Typical examples of bounding volumes include axis-aligned boxes (of which cubes are a special case) and spheres, and they are chosen for to the simplicity of finding collision between two such volumes. Hierarchical structures used for collision detection include cone trees, k-d trees and octrees [36], sphere trees [21, 34], trees based on S-bounds [6] etc. Other spatial representations are based on BSPs and its extensions to multi-space partitions [40], spatial representations based on space-time bounds or four-dimensional testing [1, 5, 8, 21] and many more. All of these hierarchical methods do very well in performing "rejection tests", whenever two objects are far apart. However, when the two objects are in close proximity and can have multiple contacts, these algorithms either use subdivision techniques or check very large number of bounding volume pairs for potential contacts. In such cases, their performance slows down considerably and they become a major bottleneck in the simulation. Many researchers have proposed hierarchical data structures based on other bounding volumes. More recently, Gottschalk et al. [17], have presented a fast algorithm and a system, called RAPID, for interference detection based on oriented bounding boxes. Barequet et al. [3] have also used oriented bounding boxes for computing hierarchical representations of surfaces in 3D and collision detection. Klosowski et al. [23] have used bounding volumes corresponding to k-DOPs for fast collision detection.

3 Overview

Any interference detection algorithm based on hierarchical data structures involves computing a hierarchical decomposition of the model and bounding volumes corresponding to each node of the tree. At runtime, the bounding volumes are transformed based on objects' motion and used for performing a quick overlap test. The total cost for interference detection is given by the following *cost* equation:

$$T = N_u \times C_u + N_v \times C_v + N_p \times C_p, \qquad (1)$$

where

T: total cost function for interference detection,
N_u: no. of bounding volumes updated due to object's motion,
C_u: cost of updating a bounding volume,
N_v: no. of bounding volume pair overlap tests
C_v: cost of testing two bounding volumes for overlap,
N_p: no. of primitive pairs tested for interference,
C_p: cost of testing two primitives for interference,

This equation differs from the cost function presented in [17]. Here we include a separate term for updating each hierarchy based on objects' motion.

For simpler hierarchies based on spheres or axis-aligned bounding boxes, C_v is a small constant (6 to 10 arithmetic operations). For higher order bounding volumes, C_v can be much higher. For example, for OBBs, the average value of C_v is about 100 arithmetic operations [17]. Since a higher order bounding volume provides a tighter fit to the enclosed geometry, N_v and N_p are relatively lower for them. The overall performance of different hierarchies varies with the geometric models and their configurations.

Given a large model composed of polygonal primitives, we present algorithms to compute a tight fitting shell around them. Initially, each primitive is decomposed into triangles. Each shell is represented by the

center of the spheres, inner and outer radius, an axis vector and the solid angle subtended at the center. We use a top down approach to build the hierarchy and decompose the model by dividing the shell volume into two halves and checking for triangles in each half. Each leaf node corresponds to a triangle. Given two hierarchies composed of spherical shells, the algorithm recursively checks two bounding shells for overlap. If they overlap, the algorithm traverses down the tree and checks the children node. Eventually, it checks the triangles at the leaf node for overlap.

In Section 5, we show that spherical shells provide a local cubic convergence to the enclosed geometry. Most of the earlier work in the literature has focussed on hierarchies providing linear or quadratic convergence only. As a result, for many geometric objects in close proximity, N_v and N_p for hierarchies based on spherical shells are much lower (as compared to other hierarchies) and this results in a faster interference detection algorithm. The same argument can be extended to distance computation algorithms.

4 Building the Shells

In this section, we define a shell and present methods to compute shells for arbitrary polygonal models. Consider two concentric spheres. Let V_1 be the portion of space outside the smaller sphere but inside the larger sphere. Now, suppose there is a cone whose apex coincides with the common center of the two spheres. Let V_2 be the collection of points in the interior of the cone. Then, the volume in the intersection of volumes V_1 and V_2 is called a *spherical shell.* Without loss of generality, we assume that the common center of the two spheres and the apex of the cone lie at the origin. Then, mathematically we can define a shell as follows: Let $r_{\max}$ and $r_{\min}$ be the radii of the larger and smaller spheres respectively. Let $\mathbf{A} = (\mathbf{A_x}, \mathbf{A_y}, \mathbf{A_z})$ be a unit vector representing the axis of the cone. Let α be the half-angle of the cone. Then, the shell is defined as a set of all points (x, y, z) satisfying:

$$\begin{aligned} r_{\min}^2 \leq x^2 + y^2 + z^2 \leq r_{\max}^2 \\ \frac{xA_x + yA_y + zA_z}{\sqrt{x^2 + y^2 + z^2}} \geq \cos\alpha \end{aligned} \tag{2}$$

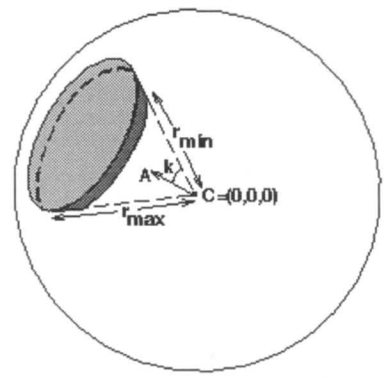


Figure 1: *Spherical shell.*

Figure 1 illustrates a spherical shell.

Thus, a shell is completely determined by the following parameters:

- $C(x, y, z)$, the center,
- $r_{\min}$ and $r_{\max}$, the radii,
- $\mathbf{A}$, a unit vector representing the axis,
- k, the cosine of the half-angle.

In the following section, we describe an algorithm to compute a shell for a given geometry. We assume that the geometry is represented in the form of a set of triangles. No adjacency information is required. As a result, the algorithm can handle all polygonal models. We refer to such models as *triangle soups.* We shall now describe the process of building a shell around a triangle soup.

4.1 Determining the center

The center of the shell is chosen as the center of a good spherical approximation to the given polygonal model. In our case, we choose this approximation to be the least squares fit sphere for all the vertices of the input model. We formulate a set of equations for this least squared problem and obtain a solution using Singular Value Decomposition (SVD) [27]. We now present the SVD formulation.

The general equation of a sphere with center (c_x, c_y, c_z) and radius r is $(x - c_x)^2 + (y - c_y)^2 + (z -$

$c_z)^2 - r^2 = 0$. Substituting $\frac{r^2 - c_x^2 - c_y^2 - c_z^2}{2}$ by d and rearranging the above equation, we get:

$$c_x x + c_y y + c_z z + d = \frac{x^2 + y^2 + z^2}{2}$$

In our case, c_x, c_y, c_z and d are unknowns. Each of m points (x_i, y_i, z_i) $(i = 1, 2, \ldots, m)$ forms a row of a *linear system.* This results in a $m \times 4$ matrix which is an over-determined system. The result obtained by SVD gives us the vector (c_x, c_y, c_z, d). It is trivial to determine the center and radius of the sphere from this vector.

A concern here is the size of the matrix on which we perform the SVD. Say it is an $m \times n$ matrix. For our problem, n is equal to 4, but m equals three times the number of triangles in the input model. Since the cost of performing an SVD is $O(m^2 n)$, for soups containing a large number of triangles, the computational cost incurred can be very high. In order that the computation of shells be fast, we need to ensure that the size of this matrix remains fairly small even if the number of triangles in the soup increases. For this, we perform a spatial subdivision of the vertices in the triangle soup into an octree hierarchy. Subdivision is performed until the octree has approximately N leaves, each with roughly equal number of triangles, where N is a predetermined constant. We compute the mean of vertices belonging to each leaf. This leaves us with a set of approximately N points. We use these to compute the best fit sphere. In our implementation, we choose $N \approx 100$. Since N is approximately a constant, the cost incurred in the SVD computations is a constant. As a result, the overall cost of computing the center is linear in terms of number of vertices. We choose the center of this best fit sphere to be the center of the shell. In practice, this approach has resulted in fairly satisfactory shell fits.

4.2 Computing $r_{\mathbf{max}}$

After computing the center, we want to compute the radius of the outer sphere. By the definition of a shell, the entire geometry should lie inside the outer sphere. In other words, all vertices of the geometry should lie inside the sphere. Therefore, we compute the distance of each vertex from the center of the shell and choose the maximum of these distances to be the r_{max}.

4.3 Computing $r_{\mathbf{min}}$

From the definition of a shell, the entire geometry should lie outside the inner sphere. However, the exterior of a sphere is not a convex volume. As a result, looking only at the vertices of the geometry to determine the radius of the inner sphere does not guarantee that the entire geometry lies outside this sphere. Therefore, we calculate the minimum distance of each triangle in the model from the center of the shell and choose the minimum of these distances to be the r_{min}.

4.4 Computing the axis

Consider the set of unit vectors from the center of the shell pointing towards the centroids of the triangles in the geometry. We compute a weighted mean of these unit vectors, where the weights are the areas of respective triangles. The vector so obtained is normalized and is treated as the axis of the shell. The intuition behind using area weighted mean is as follows: suppose some section of the geometry has a much finer tessellation than the rest. If we were to use an unweighted mean, then the axis would tend to be oriented towards the direction of the finer tessellation. Using an area weighted mean prevents this bias.

4.5 Computing the angle

For each vertex, we compute the cosine of the angle between the axis and the vector joining the center of the shell to the vertex. Since we require the entire geometry to lie inside the cone, we choose the minimum of all these cosines (i.e. the maximum of all these half-angles). If the half-angle is less than 90°, then the inside of the cone is a convex volume and thus, looking at only the vertices is sufficient to guarantee that the whole geometry lies in the interior of the cone. However, if the half-angle is greater than 90°, then the inside of the cone becomes a non-convex volume. Now, looking at only the vertices is no longer sufficient since portions of the geometry can lie outside the cone even

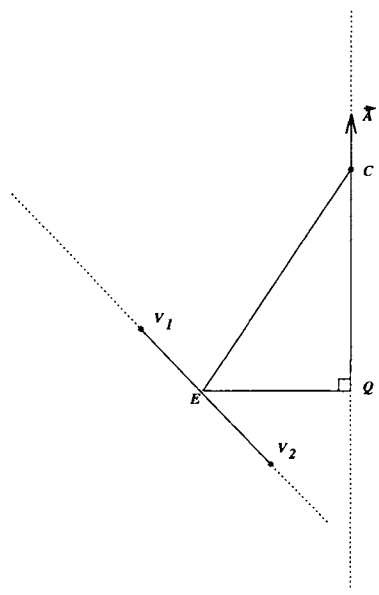


Figure 2: *Computing correct shell angle.*

if all its vertices lie inside the cone. In order to take care of this problem, we use the following algorithm:

Consider Figure 2. Here, $\mathbf{C}$ is the center of the shell, $\mathbf{A}$ is the axis and $\mathbf{V_1V_2}$ is an edge of some triangle in the given geometry. Consider a point $\mathbf{E}$ on the edge $\mathbf{V_1V_2}$. $\mathbf{Q}$ is the point of intersection of a line passing through $\mathbf{C}$ and parallel to $\mathbf{A}$ with a line passing through $\mathbf{E}$ and perpendicular to $\mathbf{A}$. We want $\mathbf{E}$ to be such that the angle that the line $\mathbf{CE}$ forms with the axis $\mathbf{A}$ is maximized. In other words, we want to minimize angle $\mathbf{ECQ}$, subject to the constraint that the point $\mathbf{E}$ lies on edge $\mathbf{V_1V_2}$.
We represent $\mathbf{E}$ parametrically as:

$$\mathbf{e} = \mathbf{V_1} + \mathbf{t}(\mathbf{V_2} - \mathbf{V_1}) \quad \cdots \quad \mathbf{0} \leq \mathbf{t} \leq \mathbf{1} \tag{3}$$

In order to minimize angle $\mathbf{ECQ}$ (this angle is less than 90°), we should to minimize $\frac{|\mathbf{EQ}|}{|\mathbf{CQ}|}$. Instead, lets minimize $\frac{|\mathbf{EQ}|^2}{|\mathbf{CQ}|^2}$. We can write the position vector of point $\mathbf{Q}$ as:

$$\mathbf{q} = \mathbf{c} + \mathbf{ka}$$

where $\mathbf{c}$ is the position vector of the center and $\mathbf{a}$ is the axis vector. Since $\mathbf{EQ} \perp \mathbf{CQ}$, we have

$$\begin{aligned} &(\mathbf{e} - \mathbf{q}) \cdot (\mathbf{c} - \mathbf{q}) = \mathbf{0} \\ \therefore \quad &(\mathbf{e} - \mathbf{c} - \mathbf{ka}) \cdot (-\mathbf{ka}) = \mathbf{0} \end{aligned} \tag{4}$$

This leaves us with 3 cases:

Case 1: $k = 0$

All such points $\mathbf{E}$ for which this holds have already been taken care of in our original algorithm. So, we ignore this case.

Case 2: $\mathbf{e} - \mathbf{c} = \mathbf{ka}$

This vector equation forms a system of three scalar equations in two unknowns, (t, k). We solve this system if it is feasible. If the solution (t, k) lies between $((0,1), (-\infty, 0)))$, then we set the half-angle to 180°. Otherwise, we ignore this case.

Case 3: $(\mathbf{e} \cdot \mathbf{a}) - (\mathbf{c} \cdot \mathbf{a}) - \mathbf{k}(\mathbf{a} \cdot \mathbf{a}) = \mathbf{0}$

Here $\cdot$ denotes dot product. We use this equation to substitute for k in:

$$f(t) = \frac{|\mathbf{EQ}|^2}{|\mathbf{CQ}|^2}$$

This gives us an equation in one unknown t, which can be written in the form:

$$f(t) = \frac{k_1 t^2 + k_2 t + k_3}{k_4 t^2 + k_5 t + k_6}$$

where $k_1, \ldots, k_6$ are known constants. Since we want to minimize $f(t)$, we solve for $f'(t) = 0$. This gives us a quadratic in t and in general, it will have two roots. If none of the roots lies in the interval $[0, 1]$, then we ignore this case. If one or both roots lie in $[0, 1]$, then we compute the half-angle subtended by these points and update our half-angle if required (this takes care of turning points that maximize our objective function).

This procedure is repeated for each edge in the geometry. Thus, we have looked at all cases where the point limiting the half-angle of the cone lies on the edges of the geometry. There is a possibility that a point in

the interior can limit the cone angle. It can be observed that such a case will arise only if a ray originating through the center of the shell in the negative direction of the axis intersects one of the triangles in the geometry. This case can be checked by ray shooting and if it occurs, we set the half-angle to 180°.

The complexity of fitting a shell to a soup of triangles (without any topological information) as described here is determined by the complexity of each step - determining center, minimum radius, maximum radius, shell axis and shell angle. Determining the center involves an octree decomposition followed by SVD. In our case, SVD takes constant time since the matrix size is fixed. The octree decomposition is performed by first placing a tight axis aligned bounding box around the entire point set belonging to an octree node and then partitioning the point set based on the position of points inside this bounding box. As a result, even in the worst case, the time complexity of the octree decomposition would be $O(n^2)$. In practice, however, it would be extremely rare to find a point set corresponding to a surface mesh that would exhibit this worst case complexity. In most cases, we observe the average case complexity of $O(n \log n)$. All the other steps in the shell building algorithm take linear time in the number of triangles.

5 Convergence Analysis

In this section, we show that the bounding volume of spherical shells approximates a smooth surface up to the second order term. In other words, the errors in the approximations decrease at a cubic rate.

5.1 Surface approximations using power series

Consider a 2-manifold $\mathbf{M} \subset \mathcal{R}^3$ and a point $p \in \mathbf{M}$ as shown in Figure 3. Without loss of generality, we make the following assumptions about p and $\mathbf{M}$.

- p is the origin,
- the tangent plane of $\mathbf{M}$ at p ($T_p(\mathbf{M})$) is the $z = 0$ plane, and

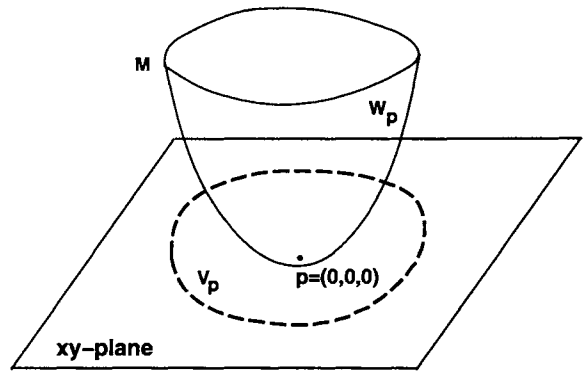


Figure 3: *Quadratic approximation of a surface.*

- the two principal directions of $\mathbf{M}$ at p are the coordinate axes $e_1 = (1,0,0)$ and $e_2 = (0,1,0)$.

It is easily seen that these conditions can be achieved by a simple rigid transformation of $\mathbf{M}$. To proceed further, we make use of a result from classical differential geometry which we state here without proof.

Theorem 1 *[33] There exists a small neighborhood W_p of $p \in \mathbf{M}$ such that the map $\pi : (x,y,z) \Rightarrow (x,y)$ is a one-to-one map with its image being an open set $V_p \subset \mathcal{R}^2$. Moreover the map π is a diffeomorphism.*

The fact that π is a diffeomorphism implies that π^{-1} exists in that neighborhood and that π and π^{-1} are smooth mappings. Therefore, we can approximate the surface in the neighborhood of p (W_p) as:

$$W_p = \{(x, y, F(x,y)) : (x,y) \in V_p\}$$

Here, $F(x,y)$ intuitively represents the surface as a height function. Further, the tangent plane of $W_p \subset \mathbf{M}$ is given by the basis vectors $\frac{\partial W_p}{\partial x} = (1, 0, \frac{\partial F}{\partial x}(0,0))$ and $\frac{\partial W_p}{\partial y} = (0, 1, \frac{\partial F}{\partial y}(0,0))$. Since, we assumed that the tangent plane at p is the $z = 0$ plane, $\frac{\partial F}{\partial x}(0,0) = \frac{\partial F}{\partial y}(0,0) = 0$.

Shape operator is a familiar concept in differential geometry. Essentially, the shape operator at a point $p \in \mathbf{M}$ (denoted by $S_p(\mathbf{M})$) is a linear operator that maps an element of $T_p(\mathbf{M})$ to another element in $T_p(\mathbf{M})$. If v_{p1} and v_{p2} are a set of basis vectors for $T_p(\mathbf{M})$, $S_p(av_{p1} + bv_{p2}) = cv_{p1} + dv_{p2}$ (a, b, c and d are

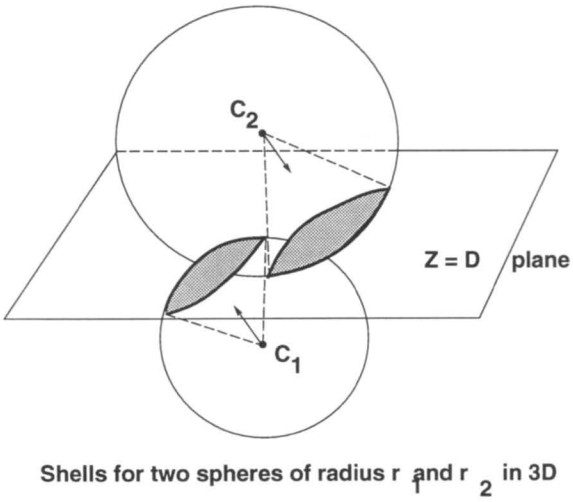


Figure 4: *Shells intersecting at specific radii.*

real valued scalars). For the special case of a vector v being a principal direction, $S_p v = Kv$, where K is the principal curvature. In our particular case, the tangent plane is spanned by e_1 and e_2. The shape operator applied to the vectors e_1 and e_2 are given by:

$$S_p e_1 = \frac{\partial^2 F}{\partial x^2}(p)e_1 + \frac{\partial^2 F}{\partial x \partial y}(p)e_2$$
$$S_p e_2 = \frac{\partial^2 F}{\partial x \partial y}(p)e_1 + \frac{\partial^2 F}{\partial y^2}(p)e_2 \qquad (5)$$

Since e_1 and e_2 are the principal directions, we can conclude that $\frac{\partial^2 F}{\partial x \partial y}(0,0) = 0$ and that $\frac{\partial^2 F}{\partial x^2}(0,0)$ and $\frac{\partial^2 F}{\partial y^2}(0,0)$ are the principal curvatures (denoted by k_1 and k_2 respectively).

We now use Taylor's formula to expand $F(x,y)$ around the origin $(0,0)$. Thus:

$$\begin{aligned} F(x,y) &= F(0,0) + x\frac{\partial F}{\partial x}(0,0) + y\frac{\partial F}{\partial y}(0,0) \\ &\quad + \frac{1}{2}(x^2\frac{\partial^2 F}{\partial x^2} + 2xy\frac{\partial^2 F}{\partial x \partial y} + y^2\frac{\partial^2 F}{\partial y^2}) \\ &\quad + \text{higher order terms} \\ &= \frac{1}{2}(k_1 x^2 + k_2 y^2) + \text{higher order terms} \end{aligned}$$

For smooth surfaces with low curvature variations, bounding volumes like shells approximate the surface

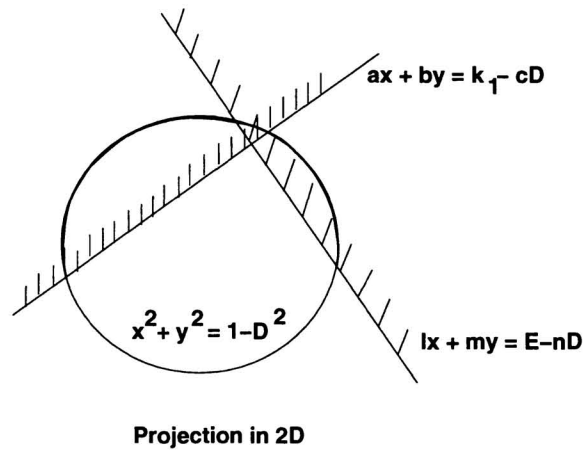


Figure 5: *Reduction of overlap test to the plane.*

up to the second order adequately. In fact, $\frac{1}{k_1}$ and $\frac{1}{k_2}$ provide bounds for the minimum and maximum radii of the shells. A caveat is in order here. The approximation of the surface we derived above is approximated well with spherical shells only if the principal curvatures have the same sign (*i.e.,* the surface is elliptic). For hyperbolic surfaces or surfaces with high curvature variations, these bounding volumes may not exhibit cubic convergence.

6 Overlap Tests between Shells

In this section, we briefly discuss the overlap test between two shells. Before we start, some notation has to be introduced. Without loss of generality, the two shells S_1 and S_2 are determined by the following parameters:

- **shells:** S_1, S_2
- **centers:** $C_1 = (0,0,0)$, $C_2 = (0,0,d)$
- **axis vectors:** (a,b,c), (l,m,n)
- **cosines of half angle:** k_1, k_2
- **radii:** $[r_{1n}, r_{1x}]$, $[r_{2n}, r_{2x}]$

We start by answering a simpler test. Let us define the set of points in the shell volume that are at a constant

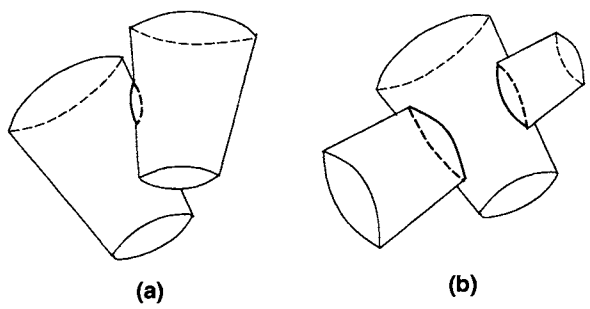


Figure 6: *Orientation of shells that do not involve extreme radii.*

distance r from the center as the *shell restricted to* r. A simpler test is to see if the shell S_1 restricted to r_1 intersects the shell S_2 restricted to r_2 (see Figure 4). Given the above parameters, the two spheres intersect along the plane $z = D'$, where $D' = \frac{d^2+r_1^2-r_2^2}{2d}$. Any intersection of the two restricted shells can only occur on this plane. Let us now write the algebraic constraints for the two restricted shells. The first of the two equations is the sphere equation and the second is the constraint for it to lie within the cone. This is obtained by taking a simple dot product. For S_1,

$$\begin{aligned} x^2 + y^2 + z^2 &= r_1^2 \\ ax + by + cz &\geq r_1 k_1 \end{aligned} \tag{6}$$

and for S_2,

$$\begin{aligned} x^2 + y^2 + (z-d)^2 &= r_2^2 \\ lx + my + n(z-d) &\geq r_2 k_2 \end{aligned} \tag{7}$$

Subtracting the two sphere equations, we get $z = D'$. The equation of the intersecting circle becomes $x^2 + y^2 = r_1^2 - D'^2$. Therefore, the two restricted shells intersect if there is a point (x, y, D') that satisfies

$$\begin{aligned} x^2 + y^2 &= r_1^2 - D'^2 \\ ax + by &\geq r_1 k_1 - cD' \\ lx + my &\geq r_2 k_2 + nd - nD' \end{aligned} \tag{8}$$

Scaling all the equations in (8) by a factor $\frac{1}{r_1}$ and replacing D'/r_1 by D and $(r_2 k_2 + nd)/r_1$ by E, we get

$$\begin{aligned} x^2 + y^2 &= 1 - D^2 \\ ax + by &\geq k_1 - cD \\ lx + my &\geq E - nD \end{aligned} \tag{9}$$

Essentially, the test boils down to checking if there exists some point on the circle which lies in the positive half-space of both lines (see Figure 5). To check for satisfaction of (9), let us parameterize x and y in the circle by

$$x = \sqrt{1-D^2}\frac{1-t^2}{1+t^2}, \qquad y = \sqrt{1-D^2}\frac{2t}{1+t^2}$$

Substituting this in the other two inequalities and simplifying, we get

$$\left(\frac{k_1 - cD}{\sqrt{1-D^2}} + a\right)t^2 - 2bt + \left(\frac{k_1 - cD}{\sqrt{1-D^2}} - a\right) \leq 0 \tag{10}$$

and

$$\left(\frac{E - nD}{\sqrt{1-D^2}} + l\right)t^2 - 2mt + \left(\frac{E - nD}{\sqrt{1-D^2}} - l\right) \leq 0 \tag{11}$$

This test can be performed easily by computing the roots of each of these equations (if they exist in the real domain) and checking if the appropriate intervals where the signs of the equations are non-positive have an intersection. However, solving the quadratic equations involves a couple of square roots. Square root is a relatively expensive operation as compared to other arithmetic operations like the additions and multiplications. If we can avoid square roots in our computation, it would really speed up our overlap test. It is fairly simple to still carry out the test without computing their actual roots.

The discussion so far has concentrated on the case when both shells are restricted (r_1 and r_2 are fixed). However, in our case, r_1 and r_2 range from $[r_{1n}, r_{1x}]$ and $[r_{2n}, r_{2x}]$ respectively. This introduces two variables to the system of constraints and makes the test a lot more difficult. We can reduce the number of variables in our system to one if we observe that when two

shells overlap, at least one of the extreme radii (r_{1n}, r_{1x}, r_{2n} or r_{2x}) of the shells is also involved except for the cases shown in Figure 6. So our test for overlap between two shells reduces to four overlap tests between a shell restricted to one of its extreme radii and another shell. The tests described previously carries through to this case as well. However, the signed expression we are evaluating changes to a (quartic) polynomial in a given range. We use Sturm sequences [39] to evaluate the signs of these polynomials without actually performing root computation. We omit many of the details here. But a detailed version of the test can be found in [24].

The overlap test may involve one square root computation (to compute distance between the centers of the shells) and up to 700 arithmetic operations, in the worst case. However, in a number of cases, the algorithm performs early rejects and early accepts. We ran a simulation by generating randomly positioned and oriented shells with limits on the distance between their centers (to prevent too many trivial accepts or rejects). In this simulation, our overlap test took 300−400 arithmetic operations on an average.

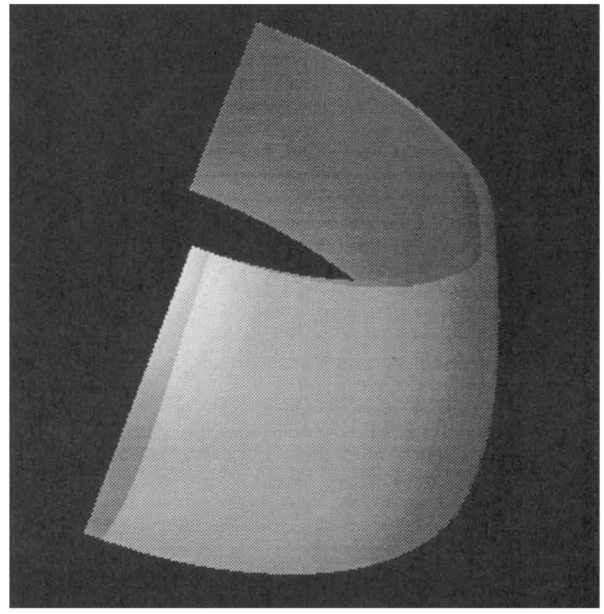


Figure 7: *Two highly tessellated bicubic patches used in the simulation. The patches are placed in parallel close proximity positions and we compare the performance of different bounding volumes on such configurations.*

7 Implementation and Performance

The software for the collision detection library has been written in C++. The main data structure is a "shell" which holds all the required parameters for a shell. Given a model, we initially build a hierarchy of shells which can be used for performing fast collision detection. We first construct a top-level shell enclosing all the triangles in the given soup. Further levels are generated by subdividing the geometry and fitting individual shells. While building the top-level shell, we keep track of the point p which limits the shell angle. Let v be the vector from p perpendicular to the axis vector. To build the shells at the next level, we consider the plane with normal v and containing both, the axis of the shell and also the center. We use this plane as a separating plane to partition the triangles in the given soup. The partitioning is based on the centroids of the triangles. If a centroid lies on the separating plane, then we arbitrarily assign the triangle to one of the partitions. This leaves us with two triangle soups. We build shells for these two soups and proceed recursively until we have one triangle per soup. Thus, we get a shell tree for each input model. In our shell building code, we use the SVD routines from the public domain FORTRAN libraries LAPACK and EISPACK. The method we have described to subdivide the geometry is purely heuristic in nature. However, without any connectivity information in the underlying model, it is very difficult to provide a provably better subdivision strategy.

In the current implementation, tree building is static, in the sense that we compute the entire shell tree for a given object before we start making any collision queries on it. However, in the future, we would like to look at building shell trees dynamically in a lazy manner. This would reduce the memory usage significantly.

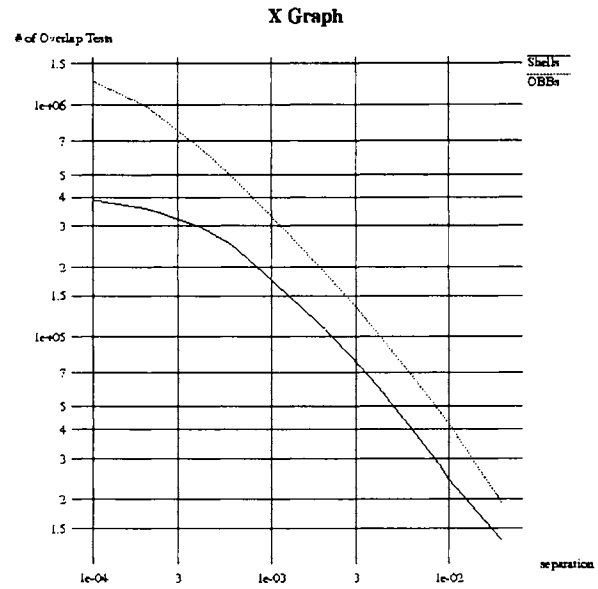


Figure 8: *OBBs (upper curve) and Shells (lower curve) for parallel close proximity.*

7.1 Parallel close proximity test

We have tested our system on some models. Of particular interest is the case of parallel close proximity since that is where higher order convergence plays a major role. In one of our simulations, we used two identical models, each of which consisted of two patches from the side of the Utah teapot. This model is shown in Figure 7. The patches have slowly varying curvature and are convex everywhere. This is admittedly a good case for the shells, but we wanted to study the empirical convergence of such cases. As a result, from the analysis presented in the previous sections, shells should provide cubic convergence for these models. Each of these models consisted of 50,176 triangles. In each frame of the simulation, we displaced one model slightly relative to the other and performed a collision query. The simulation consisted of several such frames, each with a different displacement between the models. We performed the same simulation using both shells and OBBs, and for each collision query, the number of overlap tests required were counted. The results obtained have been summarized in Table 1.

S.No.	Displacement	Shells	OBBs
1	0.1	2,277	2,063
2	0.05	4,957	5,571
3	0.02	13,353	19,333
4	0.009	27,795	47,007
5	0.006	41,257	70,091
6	0.003	77,545	133,189
7	0.0009	190,105	360,163
8	0.0006	248,607	490,057
9	0.0004	297,677	651,353
10	0.0002	357,333	982,415
11	0.0001	389,175	1,256,949

Table 1: *Overlap tests required for different displacements.*

Figure 8 shows a log-log plot of the number of collision queries required for both types of hierarchies as a function of the displacement between models. The upper curve is for OBBs while the lower one is for Shells. The differing slopes of the two curves imply that shells require asymptotically fewer overlap tests than OBBs, as a function of the displacement between the models in our experiment. Therefore, shells have a higher convergence than OBBs. As a result, for large models composed of thousands of polygons in close proximity positions, hierarchies based on spherical shells provide faster interference detection algorithms.

7.2 Volume test

The experiment conducted here was to measure the order of convergence empirically. Given a model composed of triangles, we constructed a hierarchy of spherical shells and OBBs. We use the same subdivision scheme to partition the model as we go down the hierarchy. Each node in the tree was subdivided into four children. The volume test was performed by accumulating the volume of all the spherical shells (or OBBs) occurring at a particular level of the tree. The ratio of the volumes at successive levels gives us an insight into the empirical convergence.

The model we used to conduct the volume test is shown in Figure 9. It consists of 131,072 triangles. We built two hierarchies: one with spherical shells and another with OBBs. For a shell with inner and outer radii r_1 and r_2, respectively, and cosine of half angle k,

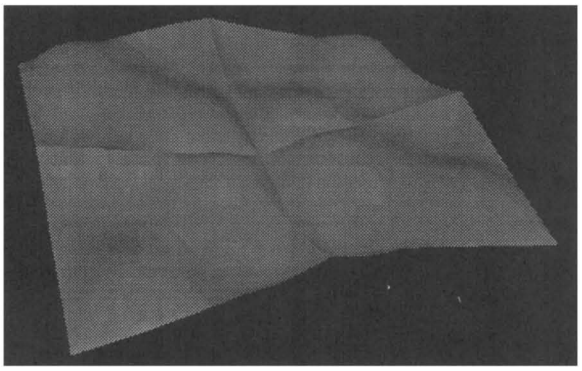


Figure 9: *Model consisting of around 131,000 triangles used in the volume test.*

Level No.	Shells		OBBs	
	Volume	Ratio	Volume	Ratio
1	237.37	–	83.432	–
2	35.1976	6.74	28.9779	2.88
3	5.31472	6.62	7.62836	3.80
4	0.969967	5.48	1.87269	4.07
5	0.250267	3.88	0.442474	4.23

Table 2: *Bounding volume ratios at various levels.*

its volume is given by the formula $\frac{2\pi(r_2^3-r_1^3)(1-k)}{3}$. The volume computation for an OBB is quite trivial. We build the tree to 5 levels (all leaves had 512 triangles each). The results are summarized in Table 2. We observe that the ratios of volumes at the higher levels is much higher for spherical shells than for OBBs. We expect a ratio of around 8 for ideal cubic convergence. We were unable to find a case for which spherical shells reached this ratio. But we were able to achieve superquadratic convergence. The ratios fall to about 4 near the lower levels. We believe that the reason for this has to do with the relatively flat nature of the geometry at these levels.

7.3 Performance on interlaced tori

We tested the performance of our system with a set of five finely tessellated *interlaced tori*. Each torus moves in a direction opposite to its adjacent torus. On collision, the direction of motion of both the tori are reversed. We ran this experiment for 200 frames. The simulations were repeated by varying the tessellations of the tori. The results are summarized in Table 3. The columns, in order, represent the number of triangles per torus, number of calls made to the collision detection system, total time taken to construct the entire hierarchy, total and average time required to answer the collision query. In this simulation, the average number of collisions reported were 55, the average collision query time was 42 microseconds, and the average number of calls made to the collision detection system was 303,506. All the timings presented here were measured on an SGI-Onyx with an R10000 processor, 195 MHz clock and 128 MB main memory.

Triangles per torus	Collision calls	Time (in micro secs.)		
		Tree const.	Coll. Query	Avg. Coll. Query
450	232469	2553	8.01e+6	34.46
882	482771	5148	2.36e+7	49.06
1152	167086	8116	8.61e+6	51.53
1458	207463	8842	9.00e+6	43.38
1800	362056	11971	1.43e+7	39.54
2178	369196	13049	1.40e+7	38.10

Table 3: *Performance of collision detection algorithm.*

8 Conclusion

In this paper, we have introduced a new bounding volume called spherical shells. We present efficient algorithms to build such shells for general polygonal models and for performing fast overlap tests between shells. The shells provide local cubic convergence to the underlying geometry. We compare its performance to other hierarchies and have shown that for many close proximity situations, the hierarchies based on spherical shells provide faster interference detection algorithms.

We are currently working on a public domain interference detection system based on spherical shells. Besides close proximity configuration, it would also be useful for highly tessellated models or spline models [24]. We also plan to use them for fast distance computation between geometric models.

Acknowledgements

This research was supported in part by a Sloan fellowship, ARO Contract P-34982-MA, NSF grant CCR-9319957, NSF grant CCR-9625217, ONR Young Investigator Award and Intel.

References

[1] A. Garica-Alonso, N. Serrano, and J. Flaquer. Solving the collision detection problem. *IEEE Computer Graphics and Applications*, 13(3):36–43, 1994.

[2] D. Baraff. Curved surfaces and coherence for non-penetrating rigid body simulation. *ACM Computer Graphics*, 24(4):19–28, 1990.

[3] G. Barequet, B. Chazelle, L. Guibas, J. Mitchell, and A. Tal. Boxtree: A hierarchical representation of surfaces in 3d. In *Proc. of Eurographics'96*, 1996.

[4] S. Bonner and R.B. Kelley. A representation scheme for rapid 3-d collision detection. *IEEE International Symposium on Intelligent Control*, pages 320–325, 1988.

[5] S. Cameron. Collision detection by four-dimensional intersection testing. *Proceedings of International Conference on Robotics and Automation*, pages 291–302, 1990.

[6] S. Cameron. Approximation hierarchies and s-bounds. In *Proceedings. Symposium on Solid Modeling Foundations and CAD/CAM Applications*, pages 129–137, Austin, TX, 1991.

[7] S. Cameron. Enhancing gjk: Computing minimum and penetration distance between convex polyhedra. *Proceedings of International Conference on Robotics and Automation*, 1997.

[8] J.F. Canny. Collision detection for moving polyhedra. *IEEE Trans. PAMI*, 8:200–209, 1986.

[9] J.F. Canny and M.C. Lin. An opportunistic global path planner. *Proceedings of International Conference on Robotics and Automation*, pages 1554–1559, 1990.

[10] H. Chang and T. Li. Assembly maintainability study with motion planning. In *Proceedings of International Conference on Robotics and Automation*, 1995.

[11] J. Cohen, M. Lin, D. Manocha, and M. Ponamgi. I-collide: An interactive and exact collision detection system for large-scale environments. In *Proc. of ACM Interactive 3D Graphics Conference*, pages 189–196, 1995.

[12] D.P. Dobkin and D.G. Kirkpatrick. Determining the separation of preprocessed polyhedra – A unified approach. In *Proc. 17th Internat. Colloq. Automata Lang. Program.*, volume 443 of *Lecture Notes Comput. Sci.*, pages 400–413. Springer-Verlag, 1990.

[13] B.R. Donald. Motion planning with six degrees of freedom. Master's thesis, MIT Artificial Intelligence Lab., 1984. AI-TR-791.

[14] H. Edelsbrunner. A new approach to rectangle intersections, part i. *Int. J. of Comput. Math*, 13:209–219, 1983.

[15] P. Finn, L. Kavraki, J.-C. Latombe, R. Motwani, C. Shelton, S. Venkatasubramanian, and A. Yao. Rapid: randomized pharmacophore identification for drug design. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 324–333, 1997.

[16] E.G. Gilbert, D.W. Johnson, and S.S. Keerthi. A fast procedure for computing the distance between objects in three-dimensional space. *IEEE J. Robotics and Automation*, vol RA-4:193–203, 1988.

[17] S. Gottschalk, M. Lin, and D. Manocha. Obb-tree: A hierarchical structure for rapid interference detection. In *Proc. of ACM Siggraph'96*, pages 171–180, 1996.

[18] M. Held, J. Klosowski, and J. Mitchell. Evaluation of collision detection methods for virtual reality flythroughs. In *Canadian Conference on Computational Geometry*, 1995.

[19] B.V. Herzen, A.H. Barr, and H.R. Zatz. Geometric collisions for time-dependent parametric surfaces. *Computer Graphics*, 24(4):39–48, 1990.

[20] J. Hopcroft, J. Schwartz, and M. Sharir. Efficient detection of intersections among spheres. *The International Journal of Robotics Research*, 2(4):77–80, 1983.

[21] P.M. Hubbard. Interactive collision detection. In *Proceedings of IEEE Symposium on Research Frontiers in Virtual Reality*, October 1993.

[22] L.E. Kavraki, J.-C. Latombe, R. Motwani, and P. Raghavan. Randomized query processing in robot path planning. In *Proc. 27th Annu. ACM Sympos. Theory Comput.*, pages 353–362, 1995.

[23] J. Klosowski, M. Held, J. Mitchell, H. Sowizral, and K. Zikan. Efficient collision detection using bounding volume hierarchies of k-dops. In *Siggraph'96 Visual Proceedings*, 1996.

[24] S. Krishnan, M. Gopi, M. Lin, D. Manocha and A. Pattekar. Efficient and Accurate Contact Determination between Spline Models us ing Spherical Shells. Technical report, Department of Computer Science, University of North Carolina, 1998.

[25] J.-C. Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, 1991.

[26] J.-C. Latombe. Personal Communication, 1995.

[27] C. Lawson. *Solving Least Squares Problems.* Prentice-Hall, 1974.

[28] M. Lin. *Efficient Collision Detection for Animation and Robotics.* PhD thesis, Department of Electrical Engineering and Computer Science, University of California, Berkeley, December 1993.

[29] M. Lin and J.F. Canny. Efficient algorithms for incremental distance computation. In *IEEE Conference on Robotics and Automation*, pages 1008–1014, 1991.

[30] M. Lin, D. Manocha, J. Cohen, and S. Gottschalk. Collision detection: Algorithms and applications. In *Algorithms for Robot Motion and Manipulation*, pages 129–142. A K Peters, 1996.

[31] T. Lozano-Pérez and M. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Comm. ACM*, 22(10):560–570, 1979.

[32] N. Megiddo. Linear-time algorithms for linear programming in r^3 and related problems. *SIAM J. Computing*, 12:pp. 759–776, 1983.

[33] B. O'Neill. *Elementary Differential Geometry.* Academic Press, London, UK, 1966.

[34] S. Quinlan. Efficient distance computation between non-convex objects. In *Proceedings of International Conference on Robotics and Automation*, pages 3324–3329, 1994.

[35] D. Ruspini, K. Kolarov, and O. Khatib. The haptic display of complex graphical environments. *Proc. of ACM SIGGRAPH*, pages 345–352, 1997.

[36] H. Samet. *Spatial Data Structures: Quadtree, Octrees and Other Hierarchical Methods.* Addison Wesley, 1989.

[37] T. Sederberg, S. White, and A. Zundel. Fat arcs: A bounding region with cubic convergence. *Computer Aided Geometric Design*, 6:205–218, 1989.

[38] R. Seidel. Linear programming and convex hulls made easy. In *Proc. 6th Ann. ACM Conf. on Computational Geometry*, pages 211–215, Berkeley, California, 1990.

[39] J. Stoer and R. Bulirsch. *Introduction to Numerical Analysis.* Springer-Verlag, 1993.

[40] W. Bouma and G. Vanecek. Collision detection and analysis in a physically based simulation. *Proceedings Eurographics workshop on animation and simulation*, pages 191–203, 1991.

[41] D. Zhu and J.-C. Latombe. New heuristic algorithms for efficient hierarchical path planning. *IEEE Trans. Robot. Autom.*, 7:9–20, 1991.

Kinetic Data Structures: A State of the Art Report

Leonidas J. Guibas, *Stanford University, Stanford, CA, USA*

Suppose we are simulating a collection of continuously moving bodies, rigid or deformable, whose instantaneous motion follows known laws. As the simulation proceeds, we are interested in maintaining certain quantities of interest (for example, the separation of the closest pair of objects), or detecting certain discrete events (for example, collisions) which may, in turn, alter the motion laws of the objects. In this paper we present a general framework for addressing such problems and the tools for designing and analyzing relevant algorithms, which we call kinetic data structures. *We discuss kinetic data structures for a variety of fundamental geometric problems, such as the maintenance of convex hulls, Voronoi and Delaunay diagrams, closest pairs, and intersection and visibility problems. We also briefly address the issues that arise in implementing such structures robustly and efficiently. The resulting techniques satisfy three desirable properties: (1) they exploit the continuity of the motion of the objects to gain efficiency, (2) the number of events processed by the algorithms is close to the minimum necessary in the worst case, and (3) any object may change its 'flight plan' at any moment with a low cost update to the simulation data structures. For computer applications dealing with motion in the physical world, kinetic data structures lead to simulation performance unattainable by other means. In addition, they raise fundamentally new combinatorial and algorithmic questions whose study may prove fruitful for other disciplines as well.*

1 Introduction

Motion is ubiquitous in the world around us. Computer disciplines, such as computer graphics, robotics, and vision, that deal with modeling the physical world must in particular deal with the modeling of motion. This may involve estimating motion parameters from sensor data, local and global motion planning and control, collision checking, physics-based simulation, character animation, rendering, etc. Unlike the modeling of shape, which has been well studied in the above disciplines over the past three decades, the analysis and modeling of motion in a cohesive fashion is still in its infancy. A key difficulty is that in modeling motion we rarely have all the motion data available at once. Motion takes place over the time dimension, and the motion parameters of a system of moving objects, whether real or virtual, can change drastically because of events and interactions between the objects that are hard to predict far into the future. Useful motion models must incorporate this basic on-line character of the problem.

In the work presented here, we will be interested in the simulation of a complex system of multiple moving objects and the efficient maintenance of various attributes of this system as the motion evolves. For example, we may be interested in maintaining (the separation of) the closest pair of objects, because we want to detect collisions between them. We may want to maintain a binary space partition (BSP) of the space containing the objects, in order to quickly obtain visibility information for rendering. Or we may want to maintain a minimum spanning tree (MST) connecting the moving objects, as a way of linking them through a wireless communications network. Closest pairs, BSPs, and MSTs are all concepts whose fundamental importance is well-known to Computational Geometry, and on which extensive research has been done. The novelty in our setting is that the defining objects are not static but in *continuous motion.* We mean 'continuous'

both in the sense of non-stop, as well as in the mathematical sense — objects do not jump from one part of space to another. We propose to study a new class of algorithmic techniques, which we call *kinetic data structures* (or KDSs for short), that exploit this continuity or coherence of the motion for maintaining these geometric structures of interest more efficiently than any known alternatives. We will refer to the attribute being maintained as the *configuration function* of the system (or CF for short). Note that each of the above CFs has a combinatorial description (e.g., the graph structure of the MST) that changes only at discrete times, when certain events occur (but the separation of the closest pair, or the total cost of the MST are themselves continuous functions of time).

Now, in principle, the continuous motion of each object can be approximated, after a discrete sampling of time, by deleting and reinserting it in a new position at each time step. But this incremental updating method, which is widely used in practice, benefits from continuity only indirectly and can be used only when the number of moving objects is small, because of its inherent inefficiency. In particular, the trouble with any fixed-rate sampling of the evolving system is that it either oversamples the system (wasting resources), or undersamples it (possibly missing critical periods) — since the events of interest to the configuration function typically occur in highly irregular patterns over time. The aim of our new technique is to take advantage of the coherence present in continuous motions so as to process a minimal number of combinatorial events, yet still maintain the CF correctly. In this respect, the way of analyzing our data structures is related to the *dynamic computational geometry* framework introduced by Atallah [10] in order to study the number of combinatorially distinct configurations of a given kind (e.g., convex hull or closest pair) that arise during known simple motions of the geometric objects. However, unlike Atallah's setting in which motions are predetermined, our data structures do not require us to know the full motion of the objects in advance. Thus they are much better suited to real-world situations in which objects can change their motion on-line because of interactions with each other, external impulses, etc.

Though our emphasis will be on exploiting the continuous motion of the present objects, we must also deal with new objects entering and old objects exiting the simulation. Such discrete changes can be handled by using dynamic data structures which are well-developed in Computational Geometry — these must then be combined with our kinetic structures. The need for handling discrete changes arises as well in *composing* kinetic data structures with each other. For example, a kinetic client of a kinetic convex hull algorithm for moving points will see a combinatorial change to the convex hull as a discrete update (a new point enters the hull, or an old point leaves).

The remaining subsections of the paper are as follows. Section 2 introduces the key idea on which our development of good KDSs is based: that of *animating proofs through time*. The motion model and qualities of good KDSs are discussed in sections 3 and 4 respectively. The following four sections survey the currently known results and discuss several ideas towards the future development of KDSs for important configuration functions in geometry, including classical problems in arrangements, intersections, visibility, etc., all not yet studied in the kinetic setting. Section 9 returns to the dynamic computational geometry framework and discusses new approaches for bounds on CF combinatorial changes under various probabilistic models. Section 10 discusses the issues that arise in implementing efficient schedulers for the event queues that arise in kinetic simulations and Section 11 explores various additional topics related to the practical implementation of KDSs. Finally Section 12 concludes by discussing the theoretical and applied significance of this work.

Although the work presented here is focussed primarily on rigid motions in a geometric context, it is worth remarking the the ideas of kinetic data structures extend to any context in which a continuous attribute evolves through a sequence of discrete events. Thus the conceptual framework we will develop is equally applicable to problems involving deforming objects, or even to completely non-geometric problems, such as the maintenance of shortest paths in a graph with continuously varying edge costs.

2 Animating Proofs Through Time

We will maintain a configuration function by performing an event-driven simulation of the motion. Such methods are based on an event queue which stores events, each with an associated event time. The simulation always proceeds to the next event in the queue; the event is removed from the queue and processed. This in turn may cause additional events to be scheduled or old events to be descheduled. And then this cycle is repeated all over — a situation not unlike that of sweep algorithms in Computational Geometry.

The critical events for us are those that change the combinatorial description of the configuration function. For example, if we are maintaining the closest pair of n moving points in the plane, the instant when the current closest pair AB is replaced by CD as the closest pair, constitutes an event that we must know about and have scheduled in the priority queue. Since discovering and scheduling these events is not easy, we will often find it advantageous to consider a somewhat larger set of events that is easier to maintain. We call the events which change the configuration function *external*, in order to distinguish them from the remaining events we process, which we call *internal*. Any correct method for maintaining the CF must process all external events; the internal ones are there for the convenience of the particular kinetic data structure we are developing.

How are we to find a superset of the set of the external events that is not too large and at the same time is easy to maintain? The key insight of our approach is that we can do so by maintaining through time an evolving *proof of correctness* of the value of the configuration function of interest. Such a proof will consist of a number of elementary conditions on the moving data, which we call *certificates*, that altogether imply (prove) the correctness of the current value of the CF[1]. Each certificate has an earliest failure time, and this failure is scheduled as an event in the event queue. Clearly, as long as no certificate fails, the combinatorial description of the CF cannot change. When a certificate fails, it is the job of the kinetic data structure to repair and update this proof. It may be that the CF has not changed (AB is still the closest pair), but the proof needs to be updated (internal event). Or it may be that both the CF and its proof need to change at such an event (external event).

Since the motion of the objects is continuous, it will be possible to repair a single certificate failure in a well-chosen proof with a relatively modest cost. If this failure is an external event, then the proof will guide us on how to update the configuration function as well. Finding proofs (i.e., certificate sets) that evolve gracefully in time as certificate failures happen is exactly the art of designing good kinetic data structures. Given a configuration function, we can generate a first cut at a proof as follows. Conceptually speaking, we stop the motion, run our favorite static algorithm for computing the value of the CF, and then collect all the tests the algorithm performed together with their outcomes. If this static algorithm is correct, then the outcomes of all these tests is a set of certificates that proves the correctness of the value of the CF. Most proofs generated this way will not animate well, or they will require considerable massaging before they yield good kinetic structures. We will present several examples of this process in later sections of this paper.

Since certificates will generally correspond to tests performed by a static algorithm for the CF, they invariably involve only a small and constant number of the moving objects. For reasons that will become clearer in Sections 3 and 4, throughout this work we will aim to only use certificates that involve a constant number of objects each, as well as to minimize the number of certificates that any one moving object is involved in.

To summarize then, a kinetic data structure maintains a CF by updating over time a proof of correctness of the CF. We saw earlier the difficulties with any scheme for maintaining the CF by doing a fixed-rate sampling of the evolving system. By using the idea of 'proof animation' we are able to do a more intelligent, uneven sampling that is better adapted to the CF, so as to avoid wasteful computation when the CF does not change, while doing the necessary update at the exact times when it does.

[1] In this paper we will often use the word 'proof' to refer to such a certificate set.

3 Motion Models

What information about the moving objects does a kinetic data structure need in order to do its work? The interface happens in the calculation of the certificate failure times. In the simplest setting, each moving object follows a publicly posted *flight plan* specifying its short-term motion. If we look at the evolving system over a period of time when these flight plans or motion laws do not change, then we can calculate for each currently valid certificate its failure time. In a typical case, the flight plans will be polynomial or rational trajectories, and the certificate itself will be a simple algebraic inequality, so the certificate failure time will be the next largest real root of some low-degree polynomial.

Of course the flight plan of an object can change. A flight plan *update* can occur because of interactions between the object and other moving objects, the environment, etc. For example, a collision between two moving objects will in general result in updates to the flight plans of both objects. But an object can also change its motion law because of its own free will. In order for the simulation to proceed correctly, whenever a flight plan update happens, the KDS must be informed about it. Events that change the flight plans of objects (e.g., collisions) must themselves be scheduled in the event queue before they happen. Thus a KDS for the MST of a set of moving and elastically bouncing balls may need to also implement a closest pair/collision detection KDS as a way to know all the flight plan updates. Or, as we already remarked, when composing KDSs, events that change the output of the first structure can generate flight plan updates for the other.

When a flight plan update for an object happens, all the certificates that involve that object must have their failure times recalculated and their positions in the event queue updated. This makes it desirable to keep the number of certificates involving any particular object small.

Although the simple motion model presented so far may be adequate for certain idealized virtual reality simulations, it is clearly insufficient when tracking moving objects in the real world. It can then be the case that the flight plans only give us partial information about the object motions, or that there are no posted flight plans at all. Instead the KDS has to try to predict the current motion of each object by extrapolating from its past motion. With this incomplete knowledge, the KDS clearly cannot calculate exact failure times for certificates. The best it can do is to calculate for each certificate a 'last time' till which the KDS can be sure that the certificate is still valid, given the partial flight plans and perhaps some *a priori* bounds on the velocity and other motion parameters of each object. Events can be scheduled for those times and the certificate then be re-examined using any updated position/motion information that may have become available in the meantime.

In the real-world context, this raises the issue of having the kinetic algorithm use sensing so as to acquire better motion information about the objects in order to resolve the status of certificates. This then becomes akin to the model of 'data in motion' introduced by Kahan [33]. What is the best way for a kinetic algorithm to use sensing in order to proceed with the simulation? This raises several research issues that have not been adequately explored yet.

Several other motion models incorporating uncertainty and limiting the ability of the KDS to examine the evolving system are worth discussing, but in order not to delay further the presentation of concrete problems, we will omit this topic. In the remaining sections we will assume that we have posted flight plans with full motion information and that all flight plan updates are events scheduled by our KDS or others in the event queue. We will also assume that each motion plan has fixed complexity independent of the number of moving objects[2]; this implies that the cost of calculating a certificate failure time is $O(1)$.

4 Analysis of Kinetic Data Structures

So far we have concentrated on the issue of correctness for a kinetic data structure. The challenges were how

[2]This assumption excludes certain kinds of simulations, such as classical n-body simulations [29], from our simplest framework.

not to miss any external events affecting the CF, and how to stay informed of all the flight plan updates. But how shall we evaluate kinetic structures? Storage and time are clearly important, but the on-line nature of the events raises some additional issues akin to those of on-line algorithms. Criteria for measuring the performance of kinetic data structures is the topic of this Section.

Though the exact nature of the flight plans is not relevant to correctness issues, it clearly affects performance. When analyzing kinetic structures we will generally assume that all trajectories followed by the moving objects are *pseudo-algebraic* splines. By this we mean that each trajectory consists of a discrete set of arcs, and the transitions between the arcs correspond to flight plan updates. The arcs themselves must be what we call pseudo-algebraic with respect to the KDS. By this we mean that all certificates used by the KDS can switch from TRUE to FALSE at most a bounded number of times, for any motions following the given arcs. This is a condition akin to the Davenport-Schinzel condition [41] commonly used to limit the complexity of curve interactions in Computational Geometry. In particular, algebraic arcs of bounded degree are always pseudo-algebraic. On occasion, we will make much stronger assumptions, such as in the usual scenario of linear, constant velocity, motions.

The performance bounds we give depend on n, the number of rigidly moving objects or object parts. Clearly the number of flight plan updates also affects the complexity of the simulation. Exactly how to incorporate that into our framework is a topic for further research. For the KDS analyses in the following sections we will assume that there are no flight plan updates, except in specific situations where we are composing kinetic structures. We will call a quantity 'small' if it is of the order of $O(\text{polylog}(n))$, or even $O(n^\epsilon)$ for some arbitrarily small $\epsilon > 0$.

We propose four major criteria for evaluating these structures.

responsiveness Most obviously, a KDS is good if the cost of processing a certificate failure is small. The responsiveness of a KDS is the worst-case amount of time needed to update its proof after a certificate failure. This may require discovering if the value of the CF has changed and what the new value is, as well as updating the certificate set by removing old certificates that are no longer part of the proof, and adding new certificates that are part of the new proof. All these certificate changes need to be reflected in the event queue as well. We call a KDS *responsive* if the the worst-case amount of time needed for such a proof update is small (in the technical sense just defined).

efficiency A second key performance measure for a KDS is the worst-case number number of events processed. Our aim will be to develop kinetic data structures for which the total number of events processed by the structure in the worst case is asymptotically of the same order as, or only slightly larger than, the number of external events in the worst case (technically, we require that the ratio of total events to external events is small). This is reasonable, as the number of external events is a lower bound on the cost of any algorithm for maintaining the desired configuration function. A KDS meeting this condition will be called *efficient*.

Note that in this definition we are comparing events over two different motions: the ones maximizing the number of events processed by the KDS, and the ones maximizing the number of external events. In a more refined setting, we call this measure *weak efficiency*. A weakly efficient KDS only guarantees that it never processes many more events than the worst-case number of external events over all allowed motions. We define *strong efficiency* to mean that the worst-case ratio of total events processed to external events (taken over all allowed motions) is small — this is akin to the *competitive ratio* of on-line algorithms [21] and is perhaps a more satisfactory notion of efficiency. Unfortunately so far we have found only few strongly efficient KDSs, and those under highly restrictive motion assumptions. We need to expand our repertory of strongly efficient structures.

locality We already remarked a number of times that it is important to keep low the maximum number of certificates in which any one object appears, in order to allow efficient flight plan updates. The locality of a

KDS is the maximum number of certificates in which any one object can ever appear. If that number is small we call the KDS *local*. The existence of local KDSs is an intriguing question for several CFs.

compactness The size of the KDS is the maximum number of certificates ever present in a proof — it also reflects the size of the event queue that needs to be maintained. We call a KDS *compact* if the maximum number of certificates ever present in a proof is of the order of n times a small (in the technical sense) quantity (i.e., if it is nearly linear).

Note that locality implies compactness, but responsiveness and efficiency are unrelated. There exist efficient but unresponsive structures, as well as inefficient but responsive ones[3].

5 An Example KDS

To make the issues above more concrete, and since we do not have the space to present in detail kinetic solutions to the more substantial problems discussed in the sequel, let us consider the following simple 1-d situation. Given a set of points moving continuously along the y-axis, we are interested in knowing at all times which is the topmost point (the largest, if we think of the points as numbers). If two points meet, we allow them to pass each other without interaction. Suppose further that we know that the points are moving with constant velocities (but possibly a different one each), starting from an arbitrary initial configuration.

If we draw the trajectories of the points in the ty-plane (where the t axis is horizontal and denotes time), then our problem is nothing but computing the upper envelope of a set of straight lines in the plane (or at least the part of it that is after the initial time t_0). This upper envelope computation can be trivially done in $O(n \log n)$ time with a divide-and-conquer algorithm (this bound holds even if points can appear and disappear at arbitrary times, but then it is not trivial [32]). In the worst case, the number of times during the motion that the topmost point changes is $\Theta(n)$. Thus we have a method for computing the configuration function of interest in time that is is only a logarithmic factor higher than the maximum number of changes in the configuration function itself.

For our purposes, however, this solution is unsatisfactory, because it is based on knowing in advance the full motions of the points: a flight plan update will force a recomputation from scratch of all future maxima. In [12] we show that various other simple solutions also suffer from drawbacks as kinetic structures. Maintaining a sorted list of the points is easy and yields a structure which is responsive, local, and compact, but unfortunately not efficient. Maintaining a heap is only slightly more complicated, and this yields a responsive, efficient, local, and compact structure. But in this case the proof of efficiency is not easy [13], and to have such a simple KDS be so hard to analyze is discouraging.

Let us also consider the following fourth solution to the kinetic maximum maintenance problem, which we call a *kinetic tournament*. The idea is to use a simple divide-and-conquer strategy. The algorithm partitions the points into two approximately equal-sized groups (arbitrarily), and recursively maintains the maximum of each group. A final comparison at the top level yields the global winner. This tournament structure generates $\Theta(n)$ certificates, each asserting an inequality between the leaders of two subtournaments. When one of these comparisons changes, the new winner has to be propagated up the tournament tree to its proper level. Clearly the update cost is $O(\log n)$, and locality is $O(\log n)$ as well.

If our points move with constant velocities, how many events will our kinetic tournament have to process? The key insight to answering this question is to realize that the kinetic tournament is implementing a divide-and-conquer algorithm for the computation of the upper envelope of n straight lines in the ty-plane (the point trajectories). For example, the comparisons performed over time at the top level for declaring the final winner are exactly those needed to merge the upper

[3] In our early work on KDSs we have focussed on problems for which compact and local KDSs exist, as the next few sections show. We expect that the development of KDSs for problems where the minimum proof size is superlinear to be considerably more challenging and to require revisions in our locality desiderata.

envelopes of the two subgroups of the lines. The overall cost of the merge is easily seen to be $O(n)$ and it follows that this upper envelope computation has a worst case cost satisfying of $C(n) = O(n \log n)$. The number of kinetic tournament events (re-schedulings, etc.) is proportional to the number of times the identity of one of the contestants at a node of the tournament tree changes. Each such identity change corresponds to an intersection in one of the sub-envelopes computed by the divide-and-conquer algorithm, and hence is counted by the $O(n \log n)$ bound on $C(n)$. Therefore the kinetic tournament accomplishes our goal of maintaining on-line the maximum of a set of moving points, and it is a responsive, efficient, compact, and local KDS.

This maximum problem is one of the rare ones for which we know of a strongly efficient KDS, but in a special case only. We present this as our fifth and final solution. If point y_i is moving as $t_i(t) = \alpha_i t + \beta_i$, then let us map it to the static vector (α_i, β_i) in the parameter plane with axes α and β. Clearly the maximum y_i at time t is the vector among these whose dot product with $(t, 1)$ is maximum. Thus it is enough to precompute the convex hull of the points in the (α, β)-plane corresponding to the tips of these vectors, and then simply track the extremum around the convex hull with a rotating tangent whose slope is $(-1, t)$. The cost of detecting each maximum change is constant. This structure is kinetic, because if flight plan update happens, it can be accommodated by updating the convex after deleting the vector corresponding to the old motion parameters and inserting the vector corresponding to the new ones. Such updates to the convex hull can be handled efficiently, in time $O(\log^2 n)$ [38].

6 Extent Problems

The first class of kinetic problems we will investigate deals with the *extent* of a set of moving objects. By extent we mean various representations of how spread out the objects are in space. We include under extent classical geometric configuration functions such as the convex hull, diameter, width, various types of bounding boxes, minimum enclosing sphere, etc. In the context of moving bodies these configurations are important for knowing if the objects have entered any forbidden regions (e.g., aircraft flying in forbidden airspace), clipping (e.g, speeding up rendering by ignoring objects outside the view port), detecting various exceptional conditions (e.g., some object gets so far away that it is out of communication range), etc.

6.1 Convex hulls and upper envelopes

We have developed a responsive, efficient (in the weak sense), local, and compact algorithm for points moving in the plane with algebraic trajectories of bounded degree [12]. Our algorithm works in a divide-and-conquer fashion, like the kinetic tournament of Section 5. It is actually easier to explain the algorithm in the dual setting, where the problem of maintaining the upper convex hull of n moving points in the plane dualizes to the problem of maintaining the upper envelope of n moving lines. Again, we partition the lines into two roughly equal groups which we call red and blue, and then recursively maintain the upper envelope of each group. The goal of the root node of the recursion is to maintain the purple upper envelope of all the lines, given the recursive maintenance of the red and blue envelopes. A certificate structure for this problem can be derived by considering how we can merge the red and blue envelopes into the purple envelope in a static setting. This can be done by a standard sweep-line algorithm and requires two types of tests: x-tests in order to decide the x-ordering of vertices of the red and blue envelopes, and y-tests that compare whether a red or blue vertex is above or below its blue or red *contender* edge (see [12] for details). Altogether these x- and y-certificates, from all levels of the recursion, form a proof of correctness of the current convex hull.

Unfortunately this set of certificates is not local: as many as $\Omega(n)$ blue vertices might be above the same red contender edge (or vice versa). If that red edge undergoes a flight plan update, a large number of certificates will need to have their failure times recomputed. In [12] we show how to modify this certificate set and make it local by adding an additional type of test, a *slope* or s-certificate between pairs of lines. It is then quite straightforward to check that when one of these x-, y-, or s-certificate fails at a node of the recursion

tree, it is possible to update the local upper envelope and corresponding certificate set in $O(1)$ time, and the same holds for all higher levels of the tree to which this change needs to be propagated. From these considerations it easily follows that the structure is local, compact, and responsive. Figure 1 shows a simple example of this process. How about efficiency? Even with straight lines motions, we can show that the convex hull can change combinatorially $\Omega(n^2)$ times [5]. Because of our algebraic trajectory assumption, it is clear that the number of events corresponding to s-certificate failures is $O(n^2)$, but the corresponding counts for y- and x-certificates are $O(n^3)$ and $O(n^4)$ respectively. However, by considering the surfaces swept by the lines over time and invoking recent theorems of Combinatorial Geometry about the complexity of upper envelopes [31], as well as the overlay of envelopes [8], we can prove a near-quadratic bound on the number of the y- and x-events as well. Thus the algorithm is efficient ($O(n^{2+\epsilon})$ events in the worst-case), but the proof of this fact requires substantial machinery.

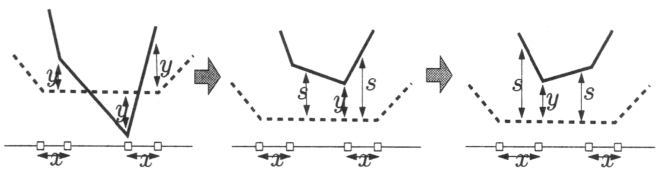


Figure 1: *A set of consecutive kinetic events.*

Efficient kinetic data structures for convex hulls and lower envelopes in higher dimensions $d > 2$ are not yet fully developed. We expect this to pose challenges, even for $d = 3$. To apply the divide-and-conquer strategy presented above to 3-d upper envelopes of moving planes we will have to maintain in the xy-plane the overlay of subdivisions corresponding to the projections of the red and blue upper subenvelopes. In Section 8 we will see how to maintain a planar subdivision induced by moving segments in the plane in the context of binary space partitions (BSPs). However, those ideas might not work as well in the current context, as the moving elements of the subdivision are themselves derived moving objects (e.g., the projections of lines that are the intersections of the primary moving planes). Also, the overlay subdivision can have $\Theta(n^2)$ complexity, even though each of the blue, red, and purple subdivisions has of course linear complexity. Thus we must find a way to maintain (the relevant parts of) the overlay subdivision implicitly, unless we are willing to consider non-compact structures with $\Theta(n^2)$ certificates. In addition, the trick we used to ensure locality for y-certificates in 2-d extends in the 3-d case only in part, unfortunately. It does not work when we assert that a convex envelope lies below a triangle — in this case all the rim vertices of the envelope seem to require independent z-certificates. Thus both sublinear locality and compactness remain open issues.

We expect progress on these fundamental structures to enable progress on other higher-dimensional kinetic problems as well.

6.2 Diameter, width, minimum spanning circle, etc.

Once we can maintain the convex hull of moving points in 2-d, we can also solve a number of other related extent problems. In [5] we show how to maintain the diameter, width, and various flavors of bounding boxes (minimum area, minimum perimeter) with the same overall kinetic performance as the convex hull. These applications show the power of composing kinetic data structures: once we have the convex hull we can maintain antipodal pairs of vertices of the convex hull and compare their distances or separation using a kinetic tournament. We also show that these kinetic algorithms are efficient, as each of the respective configuration functions can undergo $\Omega(n^2)$ changes even for points moving linearly in the plane. It will be quite interesting to consider the same problems in higher dimensions, either based on convex hull techniques, or by other more direct methods, as was done for diameter and width in the static case [18]. The kinetic maintenance of collections of bounding boxes, such as those used in OBB-trees [28] is also very interesting.

Not all 2-d extent problems can be solved this easily. Consider the problem of maintaining the minimum spanning circle (MSC) of a set of n moving

points in the plane. It is possible to prove in various ways that the maximum number of times the MSC can change combinatorially is nearly cubic (under the pseudo-algebraicity assumption), but we do not know if this bound is tight. In a space-time diagram the MSC provides a certain kind of envelope of the point trajectories. This makes us hope that some of the vertex charging schemes for lower envelopes, that were recently so successfully used to bound the complexity of lower envelopes of algebraic surfaces [40], might be applicable to this case as well. The key will be to find a way to relate vertices on the envelope (i.e., time instances when the MSC is defined by four cocircular points) to vertices inside the envelope, but not too deeply.

In terms of a kinetic data structure for the MSC, we can use the observation that the MSC is the smallest circumcircle of any of the triangles in the furthest point Delaunay triangulation of the point set (which is always a triangulation of the convex hull). If we can maintain the convex hull and the furthest point Delaunay triangulation of the point set, then we can derive an elegant kinetic structure for the MSC as follows.

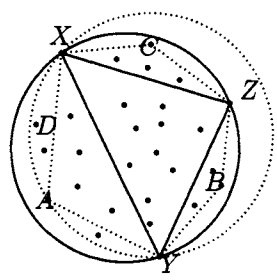


Figure 2: *Maintaining the MSC.*

Consider the dual tree of the furthest point Delaunay triangulation, and root it at the triangle whose circumcircle is the MSC. To be concrete, let this triangle be XYZ, and let the three other triangles that are the children of the root have as their third vertex the points A, B, and C respectively. We claim that as long as the convex hull and the Delaunay triangulation do not change, the MSC also cannot change. The only way for the MSC to change combinatorially is to have one of the points A, B, or C exit the current MSC, and this is a Delaunay event. A point such as D in the figure cannot escape from the MSC without first escaping from the circumcircle of the Delaunay triangle AXY — and this will also change the Delaunay triangulation. This observation creates a tree of *InCircle* Delaunay certificates of linear size, which together with the certificates needed for the convex hull, proves the correctness of the MSC. When one of the certificates fails, this MSC proof tree can be updated by simple tree rotations. Unfortunately, we do not know if this KDS is efficient. A near-cubic upper bound on the number of total events processed can be proved — the dominant term is the number of Delaunay triangulation changes, for which only an upper bound of the form $O(n^2\lambda_s(n))$ is known (see below).

6.3 Range searching

It is worth noticing that the diameter, width, MSC, etc. are all extent measures defined by a small and fixed number of the moving points. This is also the case for the canonical pieces used in various *range searching* geometric structures, including Willard partition trees [24], conjugation trees [24], and many of the newer structures for simplex range searching based on cuttings and shallow cuttings [36]. The efficient maintenance of range searching structures under continuous motion of the objects defining them is another area that needs to be developed.

7 Proximity and Intersection Problems

Proximity and intersection are fundamental modalities in describing events that affect the evolution of a system of objects in motion. Collisions between objects, i.e. transitions from non-intersection to intersection, invariably affect the flight plans of the objects involved. Proximity to other objects will often cause a moving object to change its flight in order to get closer or further away from one of the moving objects, and so on. For complex objects it is often advantageous to approximate them with simpler bounding volumes and then check for intersection between these bounding volumes

first. Thus we may want to track which pairs of bounding volumes of objects intersect, as such pairs may need to be checked for collision by a more refined algorithm. In this section we present some preliminary results and research plans for problems in this area.

7.1 Voronoi diagrams and Delaunay triangulations

The central role of Voronoi diagrams and Delaunay triangulations is well-established in Computational Geometry. From the kinetic point of view, there is both good and bad news regarding these diagrams. Let us focus on the Delaunay triangulation — these diagrams are equivalent and the events that change one combinatorially are exactly the events that change the other as well. Maintaining the Delaunay triangulation of points moving in a low-dimensional Euclidean space is surprisingly straightforward. We describe the situation for $d = 2$; analogous statements hold in higher dimensions. The key insight that helps is an old theorem of Delaunay himself: a triangulation which is 'locally Delaunay' is globally Delaunay [22]. By locally Delaunay we mean that every edge of the triangulation passes the *InCircle* test — in other words the circumcircle of the triangle on one side of the edge does not contain the third vertex of the triangle on the other side. This set of local conditions gives us a compact set of certificates for the Delaunay triangulation; their failure times become the events to be scheduled in the event queue. When a failure event happens, a 'flip operation' [24] replaces the bad edge by a good edge, and all is well. Thus a KDS for Delaunay is immediate and is clearly responsive, strongly efficient, and compact structure, though of course it is not local (a vertex can have high degree).

That was the good news. The bad news is that despite work on this problem over several decades, we still do not know a tight bound on the number of combinatorial changes in the Delaunay triangulation when the points move pseudo-algebraically, or even linearly. Currently the best known upper bound is $O(n^d \lambda_s(n))$ [30] for some constant s, and the best known lower bound is $\Omega(n^d)$. This gap makes it hard to judge the worst-case efficiency of algorithms that utilize Delaunay or Voronoi KDSs. There is some hope that we can apply recent techniques used to prove a subcubic bound on the complexity of the union of n unit cylinders in $\mathcal{R}^3$ [1] to improve the upper bound for $d = 2$.

7.2 Closest pair problems

Tracking the closest pair among a set of moving objects gives us a way to detect collisions among them: clearly the next colliding pair must be the closest pair among all the objects for a small time interval before the collision. Note also that the closest pair of n moving objects can change at most a near-quadratic number of times — we simply plot the separation of each pair as a function of time and take the lower envelope of these functions. Though Voronoi and Delaunay diagrams contain within them closest-pair information (for example, a kinetic tournament on the Delaunay edges will easily maintain the closest pair), the near-cubic bound on events for these diagrams suggests that other approaches may be significantly more efficient.

We have been able to develop two different kinetic data structures for maintaining the closest pair among n moving points. One structure is for $d = 2$ [12] (though extensions to higher dimensions should be possible), while the other works for all dimensions [15]. Both structures are based on a common insight: to find the closest pair is is sufficient to examine a linear number of point pairs defined by partitioning the space around each point into a fixed number of congruent cones having the point as their apex. These cones have a central axis and each point keeps track of some of its nearest neighbors in each cone. This of course is an old idea going back to A. Yao [16], but in our case the nearest neighbors are defined not in terms of Euclidean distance, but in terms of distance from the apex to the projections of the points on the cone axis — maintaining these neighbors is a much easier 1-d problem. A packing lemma implies that if we have fine enough cones and select enough nearest neighbors in each cone, then we cannot miss the true Euclidean closet pair. For example, in the case $d = 2$, our first approach uses three 60° cones and one neighbor per cone, while the second approach use four 90° cones and three neighbors per cone. It turns out that

point membership in these cones, as well as the 1-d nearest neighbors used by these algorithms, can all be obtained from keeping track of the sorted order in the projections of the moving points along a fixed (but of size roughly 2^d) set of directions in the space. Thus these methods clearly process a quadratic set of events in the worst case. Keeping track of which points lie in which cones is accomplished by using a kinetic multidimensional range search tree [15]. With some further insights that we do have not the space to discuss here, these ideas lead to closest pair KDSs which are responsive, efficient, local, and compact.

Several variations on the closest pair problem are interesting and open, even for $d = 2$. First, we do not have a good KDS for the bichromatic closest pair problem [2]. Second, though our closest pair methods extend to the problem of keeping track of the closest among n moving equal radius balls, they do not work when the radii are widely different. The reason is that the key packing lemma we use fails in that case. Of course a Delaunay triangulation corresponding to a weighted Voronoi diagram can still be used in this case [17]. But the issue of whether there is a KDS with a near-quadratic number of events for this problem remains open. Furthermore, it is not clear if it is possible to have a local structure at all. A large ball with many small balls near its boundary suggests that sublinear locality may not be possible — though we have been unable to prove any hard lower bound.

Finally note that our closest pair algorithms have the undesirable property that they process events corresponding to order reversals on the projections of the points along a number of axes. Many such events might be irrelevant, as the points involved are actually very far away in space. It seems intuitively desirable to develop kinetic structures where events always correspond to interactions between points that are sufficiently close. We can accomplish this by partitioning space into bins and processing events only among points in each bin, but at some point the cost of transferring points from bin to bin also becomes significant [35].

7.3 Minimum spanning trees

In a number of applications the moving objects are observers and it is important to keep communication among them for effective global motion scheduling, so as to accomplish a task (e.g., explore a building to find a target). One way of doing so is to keep a set of point-to-point wireless communication links between pairs of observers. Among all possible such communication networks, the minimum spanning tree (MST), i.e. the set of links connecting all the observers together with the minimum total Euclidean length, has many desirable properties. How can we efficiently maintain the MST of n points moving around in space?

We do not yet have a good kinetic solution for this problem. We have structures which are responsive, compact, and local, but not necessarily efficient [15]. Our structures maintain either the MST under a polyhedral distance metric, or an approximate Euclidean MST (guaranteed to be a $1 + \epsilon$ approximation of the true Euclidean MST). These structures make use of the kinetic multidimensional range search tree mentioned earlier. Our algorithms exploit again the idea of maintaining a sparse graph among the points which is guaranteed to contain all the MST edges, and then keeping track of the MST of this graph. Note that the MST can change either because the underlying sparse graph changes combinatorially, or because the weight relations among the edges of the graph change over time. We have recently [7] been able to obtain a subquadratic algorithm ($O(n^{11/6}\text{polylog}(n))$ currently, but probably better) for maintaining the MST of a (fixed) planar graph with edge weights varying in a pseudoline fashion (pseudo-algebraic of degree 1). This new algorithm is based on recursive graph partitioning using cycle separators and leads to a responsive, local, and compact KDS. The analysis uses tools similar to those exploited by Dey in his recent breakthrough result on the complexity of a level in an arrangement of lines (and pseudolines) [23]. As in the case of Voronoi/Delaunay, we are somewhat hampered by the lack of sharp bounds on the combinatorial number of changes to the MST in both the geometric and graph settings, even under linear point or weight motions (for these cases the

currently best known upper bounds are $O(n^3)$ for the Euclidean MST of n linearly moving points [34], and $O(kn^{1/3})$ for the MST of a graph with k vertices and n edges whose weights change linearly, as follows from the technique of [23]).

7.4 Incidences and many faces problems

Not every proximity problem deals with points or balls. In fact mixed proximity problems between points and lines or curves raise many interesting kinetic issues of their own. For example, consider the following problem. In the plane we are given n moving points and n moving lines. We wish to track the closest point-line pair (or perhaps just report all point line incidences during the motion). Clearly there can be a near-quadratic number of combinatorial changes to the configuration function. But a local and compact structure may not be possible. The standard many faces construction [25], as well as Erickson's lower bounds on the static version of this problem [27], suggest that a natural certificate set size for this problem is $\Theta(n^{4/3})$. By using ϵ-net techniques we can give a kinetic data structure that processes a near-quadratic number of events and has a proof size of $O(n^{5/3})$ [4]. It would be quite interesting to do better.

Several variations are also worth investigating. These include maintaining the face or faces of the arrangement of the lines containing one or several points, maintaining the zone of another line, and many other kinetic versions of classical arrangement problems. Of course, if we are willing to maintain the full arrangement of the moving lines, these problems become rather easy. The arrangement itself can be maintained by using a vertical decomposition of it — the certificates will simply assert that each trapezoid is well formed. The proof size will be $\Theta(n^2)$ and the worst-case locality $\Theta(n)$ (this is optimal for this problem). An easy counting argument can be used to show that the number of events processed by this kinetic structure will be near-cubic (in space-time we sum the squares of the complexities of the cells of that 3-d arrangement).

7.5 Maintaining intersecting pairs

We remarked earlier on the use of bounding volumes for efficient collision detection among moving objects, a method well-established in graphics and robotics [11]. Suppose for concreteness that we are in a 2-d situation and that these bounding volumes are disks. The bounding disks of two objects may intersect, even though the objects themselves do not. One way to approach the problem of collision detection is to maintain the intersecting pairs of disks. Pairs of objects whose disks intersect have to be checked for collision by a more sophisticated algorithm. When a new pair of disks starts intersecting, the corresponding pair of objects is added to the list of those to be thus checked. Similarly, when a pair of disks stops intersecting, the object pair is removed.

This is a more challenging problem than maintaining the closest pair of disks, as in the latter we can assume that all the disks are disjoint. Note that, since for our purposes it is sufficient to just report the changes in intersection status, we do not need to maintain an explicit list of intersecting pairs. This is fortunate, because there can be $\Theta(n^2)$ intersecting pairs among the n disks. One approach for discovering the changes in the intersecting pairs is to maintain a *hitting set* of points for the intersections, i.e., a set of point such that every intersecting pair contains one of these points. The arguments of [42] give an upper bound of $O(n^{5/3})$ and a lower bound of $O(n^{4/3})$ on the size of a minimum hitting set. We can also assume, without loss of generality, that the hitting set points lie in what are called *maximal* cells of the arrangement of the disks (these are cells covered by more disks than any of their neighbors — they are always intersections of a number of the circles and thus convex). A hitting set can be maintained by (1) tracking the evolution and eventual death of these maximal cells and (2) discovering the creation of new maximal cells, which always arise as a new intersection between two disks. The death of a maximal cell implies that a pair of disks stops intersecting, and that a maximal cell has fissioned into two. The set of intersections that before was hit by a single point will now require two points to be hit (unless one or both of

the new maximal cells is a single disk contained in no other, in which case it need not be hit). The tracking of a single maximal cell is the maintenance of the intersection a set of disks and this is akin to the intersection of half-planes or the convex hull problem we have already discussed in Section 6. However, the discovery of new maximal cells will require some new insights for a good KDS.

A different view of this problem that may be helpful can be obtained by lifting the disks with center (x, y) and radius r to the point in 3-d $(x, y, x^2+y^2-r^2)$. If we imagine that the paraboloid of revolution $z = x^2+y^2$is an opaque surface, then the condition that two disks intersect is exactly that the corresponding 3-d points are mutually visible, despite the paraboloid obstruction. Thus as these points fly around, we are interested in detecting visibility changes among the point pairs. We also note that this intersection maintenance question makes sense for other bounding volumes, including axis-aligned boxes — for which it may be easier.

8 Visibility Problems

The use of visibility information is essential in rendering animations involving very large large models (e.g., architectural walkthroughs) as well as in planning motions related to visibility tasks (model or environment acquisition by one or more mobile robots with cameras, guarding or art gallery type problems, pursuit-evasion algorithms, etc.). In the context of kinetic data structures, these applications pose problems such as the maintenance of the portions of the environment currently visible from one or multiple moving observers, the maintenance of the mutually visible pairs of observers, etc.

8.1 Binary space partitions

Binary space partitions (or BSPs) were one of the earlier structures used in computer graphics to obtain visibility information. The application of BSPs to visibility determination is based on the principle that an environment can be described as a collection of clusters, which can be separated from each other in such a way that a visibility ordering can be defined between the clusters. The partition of the clusters is done by means of cutting the underlying space into two halfspaces by a plane. The crucial observation is that clusters that lie in the same halfspace as a given viewpoint can obscure, but never be obscured by, the clusters on the other side — which then provides a visibility order that can be used to correctly render a scene. A recursive application of the partition operation yields a binary space partition tree from which a visibility ordering can be determined by performing a depth-first search in the tree. There have been a few attempts to update BSPs when the objects defining them move [37, 45, 20]. All these prior efforts, however, reduce to deleting moving objects from their earlier positions and reinserting them in their current positions after some time interval has elapsed. Such approaches suffer from all the problems discussed in Section 1.

We have recently been able to obtain kinetic data structures for disjoint moving segments in the plane [6], and disjoint moving triangles in space [3]. These methods are based on defining a BSP by cuts along the given objects or parallel to a particular axis; the cuts are generated according to a random ordering of the objects. The resulting BSP has expected size $O(n \log n)$ and depth $O(\log n)$ in 2-d, and expected size $O(n \log^2 n+k)$ and depth $O(\log n)$ in 3-d, where k denotes the number of intersections between pairs of edges in the projections of the triangles on the xy-plane. As this latter bound suggests, our 3-d structure is based on a kinetic 2-d structure for maintaining the BSP of set of moving and possibly intersecting segments. These BSPs behave very well from the kinetic point of view: we can detect when the current BSP structure becomes invalid due to object motion and update the tree at a cost of $O(\log n)$ per event for the 2-d BSP, and $O(\log^2 n)$ for the 3-d BSP. In the easier to explain 2-d algorithm the events correspond to times when certain critical trapezoids, called *transient* trapezoids [6], collapse by having their two parallel sides coincide. The event counts are $O(n^2)$ and $O(n^2\lambda_s(n))$ in the 2-d and 3-d cases respectively. These structures are responsive and strongly efficient. Their size is bounded by the BSP size, but they are not local.

Many interesting issues remain open. To obtain the above bounds we need to assume that there is no correlation between the motions of the objects and the fixed random ordering used by the algorithm. An adversary could, of course, design motions aimed at eventually making these BSPs bad, but then the algorithm could respond by evolving over time the ordering through random transpositions and making the corresponding updates to the tree. An even more fundamental open question has to do with BSP optimality, a notoriously difficult problem. Our BSPs above are strongly efficient, but could it not be that there is some other BSP that undergoes many fewer events for the same motions? Can we prove that all 2-d BSPs of a certain type must undergo $\Omega(n^2)$ transitions for some pseudo-algebraic motion of the n disjoint line segments in the plane?

8.2 Visibility set

For very large environments, even the fastest hardware depth-buffers cannot render a scene at interactive frame rates. Thus techniques have been developed for computing efficiently a relatively small superset of the potentially visible polygons, called the *visibility set*, and then throwing only those to the depth buffer [43]. A number of approaches have been proposed for maintaining this visibility set as the observer moves [19, 9] in a 2-d environment. Accomodating both observer and environmental motion is clearly an area where further research needs to be done.

9 Probabilistic Event Bounds

Given the rather high worst-case event counts we saw in earlier sections, it makes sense to ask about the number of changes these configuration functions undergo when we consider average, or 'typical' object motions. Defining typical object motions is of course very application dependent. Nevertheless, we have initiated an investigation of how configuration functions change for very simple random point motions in the plane.

Assume n points choose independently a random origin and destination in the unit square and then, in the time interval $[0, 1]$, they move with constant velocity along a straight line segment from their origin to their destination. This will be our model of a random motion in the plane. In [46] it is shown that the convex hull of the points changes $\Theta(\log^2 n)$ times in expectation; the corresponding bounds for closest pair and Delaunay triangulation are $\Theta(n)$ and $\Theta(n^{3/2})$ respectively. The methods used to obtain these results mimic those that have been successfully applied to the same questions in the static setting [39]. For instance, in the case of the convex hull, we compute the conditional probability that three points are on the convex hull given that they are collinear along a line ℓ at time t. We then compute the joint probability density on the product space of lines and times, and use linearity of expectation to complete the calculation. Such probabilistic analyses give simple answers to questions that are much harder to tackle in the worst case. Some preliminary calculations suggest that, in d dimensions and for a similar probabilistic model, the Delaunay triangulation changes $\Theta(n^{1+1/d})$ times and that the closest pair changes $\Theta(n^{2/d})$ times in expectation.

The expected number of events processed by a KDS can also be computed in this model. For example, the convex hull structure we have presented in Section 6 processes $\Theta(n)$ events in expectation, while our 2-d closest point structures process $\Theta(n^2)$ events even in expectation, as the latter have to maintain sorted orderings along one or more axes. There is a simple modification to one of the closest pair algorithms [14] that avoids these sorts and experimentally seems to process roughly $\Theta(n^{3/2})$ events — but we have not been able to prove this formally. For Delaunay, of course, there are no internal events, so the trivial KDS for Delaunay also processes $\Theta(n^{3/2})$ events in expectation.

The case of the minimum spanning tree seems to be especially difficult to analyze in this random setting, and the above methods do not apply. An upper bound of $O(n^{5/2})$ (to be contrasted with the best known worst-case upper bound of $O(n^3 2^{\alpha(n)})$ [34]) follows from the fact that the MST is a subgraph of the Delaunay triangulation and some standard batching argument. A major subgoal in solving this problem is to answer the following static question: given n points chosen independently at random according to a pre-

scribed distribution, what is the expected size of the (graph) diameter of their MST?

10 Kinetic Event Scheduling

In the efficient implementation of kinetic data structures, a key problem is the efficient scheduling and de-scheduling of events (in our implementations this accounts for 70% to 90% of the total cost). The twist that makes this different from a standard priority queue maintenance problem is that the calculation of the event times is part of our cost. Since our proofs evolve rapidly over time, the accurate calculation of an event time may be wasteful if the corresponding certificate gets removed from the proof before its failure time. We cannot, of course, know the future, but this situation suggests that we compute less accurately event times that are far into the future, and refine our estimates as they get closer and closer to the present time.

Motivated by these issues, we have been studying the following problem. Suppose we maintain a set $\mathcal{S}$ of low degree polynomials $\{f_1(t), f_2(t), \ldots, f_k(t)\}$ — these are associated with the certificates currently active in a KDS. For example, assuming we have points with polynomial motions, if the event we worry about is that 'points A, B, and C become collinear', or equivalently 'the triangle ABC changes sign', then the associated polynomial $f(t)$ will simply be the signed area of the triangle ABC at time t. There is the notion of the current time t_0, and we are interested in finding quickly time t_1, the smallest real root of any of the f_i which is larger than t_0. The time t_1 is the time of the next kinetic event. Then we advance time by setting $t_0 \leftarrow t_1$ and let our KDS update its certificate set and thus the set of active polynomials. If the KDS is responsive, the changes to $\mathcal{S}$ will be relatively small. We then repeat this process all over to continue the simulation.

In order to maintain the next largest real root of this slowly evolving set of polynomials we can of course calculate all the real roots of each polynomial to the required precision for the simulation and insert all these roots into a standard priority queue. But, as remarked above, this may spend root-finding cycles for events that will never happen. We have been investigating an approach based on Sturm sequences in which for each active polynomial $f(t)$ we isolate its roots into a set of intervals whose size increases exponentially as t goes to infinity. This exponentially increasing set of intervals contains all the real roots of f — of course we allow intervals that contain many roots. The leftmost interval of f represents f in a priority queue where the polynomials compete to determine the smallest real root. Both the process of resolving comparisons between such intervals, as well as certain large steps forward in time, can cause us to refine these interval sequences to obtain tighter bounds on the roots of f. We are still developing the theoretical basis of this approach and a preliminary implementation. Note that, assuming polynomial or rational motions, an interval-based approach may allow a simulation using only rational arithmetic.

11 Implementation Issues

In order to validate the use of kinetic data structures in practice, we have implemented a number of the previously discussed structures and compared them to alternative methods for maintaining the same configuration function. This implementation has raised a number of additional research issues that need to be addressed.

11.1 Kinetic convex hulls in practice

In addition to the kinetic method presented in Section 6, we implemented the straightforward Delaunay triangulation KDS of Section 7 and a 'brute-force' algorithm. We implemented the Delaunay KDS because it is used in several kinetic problems, but in particular it contains the convex hull as a substructure. For the brute-force structure, we simply calculate the time at which each point will hit (or leave) the convex hull, assuming its current motion remains unchanged. When a point on the hull has a flight plan update, we must recalculate the event times of all the other points[4]. Furthermore, whenever a point enters or leaves the convex hull, all events must be rescheduled.

[4]When a non-hull point updates its flight plan, we need only recalculate its own event time.

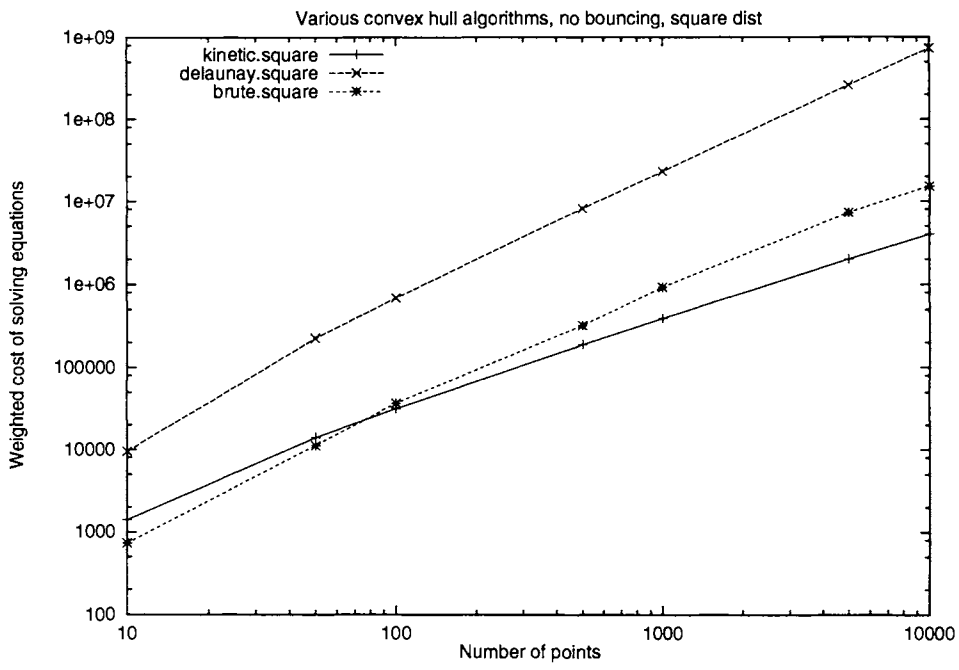


Figure 3: *Cost (weighted sum of the number of equations solved) of the three different methods that maintain the* CONVEX HULL, *when the points have linear motions.*

In order to remain independent of specific implementation details, the cost of a KDS was taken to be a sum over the set of polynomial equations solved, each weighted by an amount corresponding to the difficulty of solving equations of that degree. We ran these three methods with n ranging up to 10000 points on random motions like those in Section 9[5]. We let the simulation run until the convex hull stabilized and there was no more events in the event queue (Figure 3). As Figure 3 shows, the kinetic structure becomes superior to the brute force structure for n less than 100, and it is always much superior to the Delaunay KDS as the latter is hampered by having to solve fourth degree equations (as opposed to second degree for the convex hull KDS). Three snapshots from our implementation are shown in Figure 4. On the bottom is a time window showing event times. The long bars represent external events, while the short bars show internal events. A public demo of this software is available at `www-graphics.stanford.edu/~jbasch/demokin`.

Several additional experiments are described in [14] and in a forthcoming fuller version of that paper based on current work. Our primary finding is that, with a careful implementation, kinetic data structures do not pose major difficulties in practice and perform well on several natural motion distributions. For a large number of points, alternative data structures suffer from non-locality or expensive root-finding operations.

11.2 Open research issues

The implementation of kinetic structures raises numerous questions that we intend to address.

- We have already discussed the issue of avoiding the precise calculation of event times until they are needed. There is also concern about processing events out of of order, or multiple times, because of numerical errors. With some preliminary efforts

[5]Alternative distributions, such as the uniform unit disk and the Gaussian, gave qualitatively similar results.

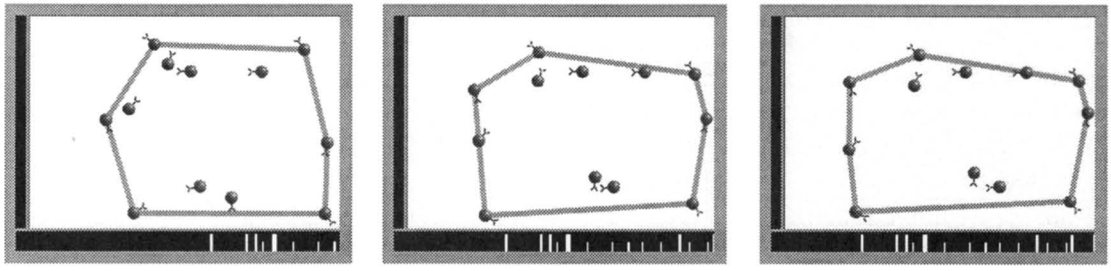


Figure 4: *Three frames from a kinetic convex hull animation.*

in this direction, we were able to run reliable simulations involving 50,000 points with about 400,000 events for the convex hull KDS.

- It is not clear that processing all events in the ideal time order is necessary for a correct simulation. If two events happen at almost the same time, but they involve sets of objects that are far apart from each other and do not otherwise interact according to the CF, it may be acceptable to process these events in either order. Of course we do want to guarantee that all events affecting any particular object happen in the correct sequence. This then would give us a 'local clock' for each object and we would not require global synchronization between all these clocks — an approach reminiscent of a topological sweep [26], but now in the time domain.

- What if too many events happen in a short time interval and we do not have the computational resources to process them all in that interval? Can we batch the processing of nearby events for efficiency? Given a description of a class of motions for the objects (e.g., bounds on velocities, etc.), can we prove that with a given amount of computational resources we will never get into this bottleneck?

12 Conclusions and Impact of the Work

We have presented the notion of kinetic data structures, based on the idea of animating proofs through time. Kinetic structures raise numerous new and interesting combinatorial and algorithmic questions, while at the same time offering the promise of significant impact in all areas of computer science dealing with simulating motion or continuous change in the physical world.

KDSs implement time sweeps, and as such have a lot in common with geometric space sweep methods (event queues, event scheduling, etc.). The novel aspect of kinetic structures is that in the time dimension the future is not known *a priori* — it is decided as we go along the time axis. Such structures are based on proofs or certificate sets that are maintained under continuous motion of the participating objects. They raise questions about what kinds of proofs are possible to certify the correctness of various geometric configuration functions — lower bounds on proof size and locality become interesting problems. At this point in the development of the area, we do not understand deeply which proofs animate well in the sense that they evolve gracefully across certificate failures. There is some evidence that proofs derived from *parallel* algorithms (especially of the kinds used in *parametric searching* [44]) animate better than those coming from sequential algorithms. The intuition is that proofs from parallel algorithms are 'shallower' (in the sense of smaller maximal deduction depth, not in the common mathematical sense) — their parts are more independent and they have relative short deduction chains. These attributes help make proof updates less costly and give good bounds on locality.

Kinetic data structures also motivate the development of better techniques for estimating event counts (CF changes) in both the worst and average cases. In addition, the current analyses of KDSs need to be re-

fined to include the number of flight plan updates in the complexity. The interaction between kinetic and dynamic structures needs to be better developed as well — we saw that many problems require both. Finally, more careful analyses are needed for the cases where only some objects are moving, or large groups of objects are moving rigidly together.

Acknowledgements

The author wishes to thank Pankaj Agarwal, Julien Basch, John Hershberger, Micha Sharir, and Li Zhang for many useful discussions. This work was funded by ARO–MURI grant DAAH04-96-1-007, and NSF grants CCR-9623851 and IRI-9619625.

References

[1] P. Agarwal and M. Sharir. The complexity of the union of unit cylinders in space. Manuscript. To appear (1997).

[2] P. K. Agarwal, H. Edelsbrunner, O. Schwarzkopf, and E. Welzl. Euclidean minimum spanning trees and bichromatic closest pairs. *Discrete Comput. Geom.*, 6(5):407–422, 1991.

[3] P. K. Agarwal, J. Erickson, and L. J. Guibas. Kinetic binary space partitions for intersecting segments and disjoint triangles. In *Proc. 9th ACM-SIAM Sympos. Discrete Algorithms*, page to appear, 1998.

[4] P. K. Agarwal and L. J. Guibas. Kinetic problems on arrangements. Manuscript. To appear (1998).

[5] P. K. Agarwal, L. J. Guibas, J. Hershberger, and E. Veach. Maintaining the extent of a moving set of points. In *5-th workshop on algorithms & data structures (WADS)*, 1997.

[6] P. K. Agarwal, L. J. Guibas, T. Murali, and J. Vitter. Cylindrical static and kinetic binary space partitions. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 39–48, 1997.

[7] P. K. Agarwal, L. J. Guibas, and M. Rauch. Kinetic minimum spanning trees. Manuscript. To appear (1998).

[8] P. K. Agarwal, O. Schwarzkopf, and M. Sharir. The overlay of lower envelopes and its applications. *Discrete Comput. Geom.*, 15:1–13, 1996.

[9] B. Aronov and M. Teichmann. Online maintenance of visibility and shortest-path information. Manuscript. To appear (1997).

[10] M. J. Atallah. Some dynamic computational geometry problems. *Comput. Math. Appl.*, 11:1171–1181, 1985.

[11] D. Baraff. Interactive simulation of solid rigid bodies. *IEEE Computer Graphics and Applications*, 15(3):63–75, May 1995.

[12] J. Basch, L. J. Guibas, and J. Hershberger. Data structure for mobile data. In *Proc. 8th ACM-SIAM Sympos. Discrete Algorithms*, pages 747–756, Jan. 1997.

[13] J. Basch, L. J. Guibas, and G. Ramkumar. Sweeping lines and line segments with a heap. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 469–471, 1997.

[14] J. Basch, L. J. Guibas, C. D. Silverstein, and L. Zhang. A practical evaluation of kinetic data structures. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 388–390, 1997.

[15] J. Basch, L. J. Guibas, and L. Zhang. Proximity problems on moving points. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 344–351, 1997.

[16] J. L. Bentley, B. W. Weide, and A. C. Yao. Optimal expected-time algorithms for closest-point problems. *ACM Trans. Math. Softw.*, 6:563–580, 1980.

[17] J.-D. Boissonnat and M. Yvinec. *Géométrie algorithmique.* Ediscience international, Paris, 1995.

[18] B. Chazelle, H. Edelsbrunner, L. Guibas, and M. Sharir. Diameter, width, closest line pair and parametric searching. *Discrete Comput. Geom.*, 10:183–196, 1993.

[19] D. Z. Chen and O. Daescu. Maintaining visibility of a polygon with a moving point of view. In *Proc. 8th Canad. Conf. Comput. Geom.*, pages 240–245, 1996.

[20] Y. Chrysanthou. *Shadow Computation for 3D Interaction and Animation.* Ph.D. thesis, Queen Mary and Westfield College, University of London, 1996.

[21] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, Cambridge, MA, 1990.

[22] B. Delaunay. Sur la sphère vide. A la memoire de Georges Voronoi. *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskih i Estestvennyh Nauk*, 7:793–800, 1934.

[23] T. Dey. Improved bounds for k-sets and k-th levels. Manuscript. To appear (1997).

[24] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*, volume 10 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Heidelberg, West Germany, 1987.

[25] H. Edelsbrunner, L. Guibas, and M. Sharir. The complexity of many cells in arrangements of planes and related problems. *Discrete Comput. Geom.*, 5:197–216, 1990.

[26] H. Edelsbrunner and L. J. Guibas. Topologically sweeping an arrangement. *J. Comput. Syst. Sci.*, 38:165–194, 1989. Corrigendum in 42 (1991), 249–251.

[27] J. Erickson. New lower bounds for Hopcroft's problem. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 127–137, 1995.

[28] S. Gottschalk, M. Lin, and D. Manocha. OBB-Tree: A hierarchical structure for rapid interference detection. In H. Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 171–180. ACM SIGGRAPH, Addison Wesley, Aug. 1996.

[29] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *Journal of Computational Physics*, 73:325–348, 1987.

[30] L. Guibas, J. S. B. Mitchell, and T. Roos. Voronoi diagrams of moving points in the plane. In *Proc. 17th Internat. Workshop Graph-Theoret. Concepts Comput. Sci.*, volume 570 of *Lecture Notes Comput. Sci.*, pages 113–125. Springer-Verlag, 1991.

[31] D. Halperin and M. Sharir. New bounds for lower envelopes in three dimensions, with applications to visibility in terrains. *Discrete Comput. Geom.*, 12:313–326, 1994.

[32] J. Hershberger. Finding the upper envelope of n line segments in $O(n \log n)$ time. *Inform. Process. Lett.*, 33:169–174, 1989.

[33] S. Kahan. A model for data in motion. In *Proc. 23th Annu. ACM Sympos. Theory Comput.*, pages 267–277, 1991.

[34] N. Katoh, T. Tokuyama, and K. Iwano. On minimum and maximum spanning trees of linearly moving points. *Discrete Comput. Geom.*, 13:161–176, 1995.

[35] D. J. Kim, L. J. Guibas, and S. Y. Shin. Fast collision detection among multiple moving spheres. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 373–375, 1997.

[36] K. Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms*. Prentice Hall, Englewood Cliffs, NJ, 1994.

[37] B. F. Naylor. Interactive solid geometry via partitioning trees. In *Proceedings of Graphics Interface '92*, pages 11–18, May 1992.

[38] M. H. Overmars and J. van Leeuwen. Maintenance of configurations in the plane. *J. Comput. Syst. Sci.*, 23:166–204, 1981.

[39] A. Rényi and R. Sulanke. Über die konvexe Hülle von n zufällig gerwähten Punkten I. *Z. Wahrsch. Verw. Gebiete*, 2:75–84, 1963.

[40] M. Sharir. Almost tight upper bounds for lower envelopes in higher dimensions. *Discrete Comput. Geom.*, 12:327–345, 1994.

[41] M. Sharir and P. K. Agarwal. *Davenport-Schinzel Sequences and Their Geometric Applications*. Cambridge University Press, New York, 1995.

[42] H. Tamaki and T. Tokuyama. How to cut pseudo-parabolas into segments. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 230–237, 1995.

[43] S. J. Teller and C. H. Séquin. Visibility preprocessing for interactive walkthroughs. In T. W. Sederberg, editor, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 61–69, July 1991.

[44] S. Toledo. Extremal polygon containment problems and other issues in parametric searching. M.s. thesis, Dept. Comput. Sci., Tel Aviv Univ., Tel Aviv, 1991.

[45] E. Torres. Optimization of the binary space partition algorithm (BSP) for the visualization of dynamic scenes. In C. E. Vandoni and D. A. Duce, editors, *Eurographics '90*, pages 507–518. North-Holland, Sept. 1990.

[46] L. Zhang, H. Devarajan, and J. B. P. Indyk. Probabilistic analysis for combinatorial functions of moving points. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 442–444, 1997.

Taylor & Francis
Taylor & Francis Group
http://taylorandfrancis.com

The Polygon Exploration Problem: A New Strategy and a New Analysis Technique

Frank Hoffmann, *Freie Universität Berlin, Berlin, Germany*
Christian Icking, *FernUniversität Hagen, Hagen, Germany*
Rolf Klein, *FernUniversität Hagen, Hagen, Germany*
Klaus Kriegel, *Freie Universität Berlin, Berlin, Germany*

We provide a new on-line strategy that enables a mobile robot with vision to explore an unknown polygon by a tour less than 26.5 times as long as the shortest watchman tour. This improves considerably on the best upper bound of 133 known so far. Our strategy uses a new way of dynamically decomposing the polygon. The analysis is based on a novel geometric structure called the angle hull.

1 Introduction

Suppose a mobile robot has to explore an unknown simple polygon. The robot starts walking at a given boundary point, s; it is equipped with a vision system that continuously provides the visibility polygon of the robot's current position. When each point of the polygon has been visible at least once, the robot returns to s.

The problem is in designing a *competitive* exploration strategy, i. e. one that guarantees the robot's path never to exceed in length a fixed constant times the length of the shortest watchman tour through s.

The *off-line* version of this problem, where the polygon is known, has been extensively studied. Chin and Ntafos [3] have shown that it is sufficient and necessary for the shortest watchman tour from s to visit the prolongations of some edges, the so-called essential cuts. Moreover, they have shown that the shortest watchman tour can be computed in time $O(n^4)$; their result has later been improved on by Tan and Hirata [9] to $O(n^2)$. Carlsson et al. [2] have proven that the shortest watchman tour without a specified point s can be computed in time $O(n^3)$. Furthermore, Carlsson and Jonsson [1] proposed an $O(n^6)$ algorithm for computing the shortest path inside a simple polygon from which each point of the boundary is visible, where start and end point are not specified.

For the *on-line* problem, Deng et al. [4] were the first to claim that such a competitive strategy does in fact exist. In [5] they gave a proof for the rectilinear case, where a simple greedy strategy can be applied that performs surprisingly well.

For the general case, we have presented a strategy with a proven factor of 133 in [6]. Although this value compares favorably against the bound discussed in [4], which was in the thousands, it is still too large to be attractive to practitioners. The difficulty is in providing tight length estimates for curves of complex structure.

In this paper we show how to explore an unknown polygon by a tour at most 26.5 times as long as the shortest watchman tour containing s. Even more interesting than the big improvement on the competitive factor are the methods by which it is achieved.

Former exploration strategies were using subdivisions of the polygon that were e. g. based on the shortest path tree of s. The robot had to explore the subpolygons one by one; however, in doing so it would often catch a glimpse of, or even enter, neighbouring subpolygons. This made for a complicated and lossful analysis.

The strategy presented in this paper avoids this difficulty. The subdivision we are using here leads to groups of vertices, rather than subpolygons. Moreover, it is dynamical, in the sense that suitable groups of vertices are defined by the robot's behavior, rather than by structural properties of the polygon. As a consequence, the local exploration paths can now more easily be charged to the elements of the subdivision.

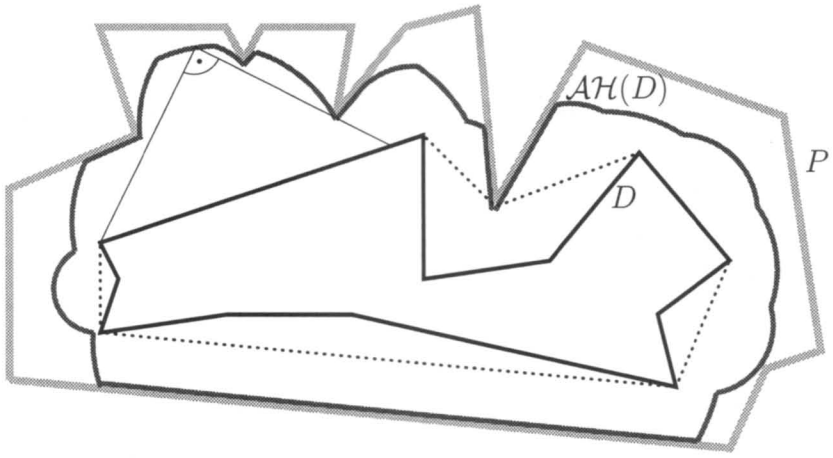


Figure 1: *The angle hull $\mathcal{AH}(D)$ with respect to a polygon P.*

In analyzing the performance of our strategy we are using a new geometric structure called the *angle hull.* Let D be a relatively convex[1] polygon inside a simple polygon, P. Then the *angle hull*, $\mathcal{AH}(D)$, of D consists of all points in P that can see two points of D at an angle of 90°, see Figure 1. In [7] we have shown the following.

Theorem 1 *Let P be a simple polygon containing a relative convex polygon D. The arc length of the boundary of the angle hull, $\mathcal{AH}(D)$, with respect to P is less than 2 times the length of D's boundary. This bound is tight.*

The outline of this paper is as follows. Section 2 contains both a hierarchical description of the strategy and its analysis. In Section 2.1 we first discuss how to explore a single corner of a polygon, namely the invisible edge behind the clockwise[2] first right reflex[3] vertex that is visible; note that this vertex may change while the robot is still moving. From Theorem 1 we obtain a bound to the length of the resulting path in terms of the length of the shortest path that leads to the final position.

Then, in Section 2.2, we use this technique for efficiently exploring groups of right reflex vertices in clockwise order. The length of the resulting local tour is shown to be bounded by the perimeter of the relative convex hull (= $\mathcal{RCH}$) of certain *base points*, times a constant.

In Section 2.3 we show how the robot recursively detects, and explores, an exhaustive system of groups of right and left reflex vertices. Groups of vertices that are on sufficiently different recursive levels give rise to base point sets whose $\mathcal{RCH}$s are mutually invisible. Therefore, the sum of their hull's perimeters is less than the perimeter of the $\mathcal{RCH}$ of their union. Since all base points are contained in the angle hull of the optimum watchman tour, W_{opt}, the perimeter of their $\mathcal{RCH}$ must be less than the perimeter of $\mathcal{AH}(W_{opt})$. Now we can apply Theorem 1 a second time and obtain an upper bound to the total length of all local exploration tours in terms of the length of W_{opt}.

Since we are using Theorem 1 twice, in analyzing different hierarchical levels of the strategy, it contributes a quadratic term to our competitive factor of 26.5.

[1]The *relative convex hull*, $\mathcal{RCH}(D)$, of a subset D of a polygon P is the smallest subset of P that contains D and, for any two points of D, the shortest path in P connecting them.

[2]With repect to the order of vertices on the polygon's boundary, starting from s.

[3]A *reflex vertex* is one whose internal angle exceeds 180°.

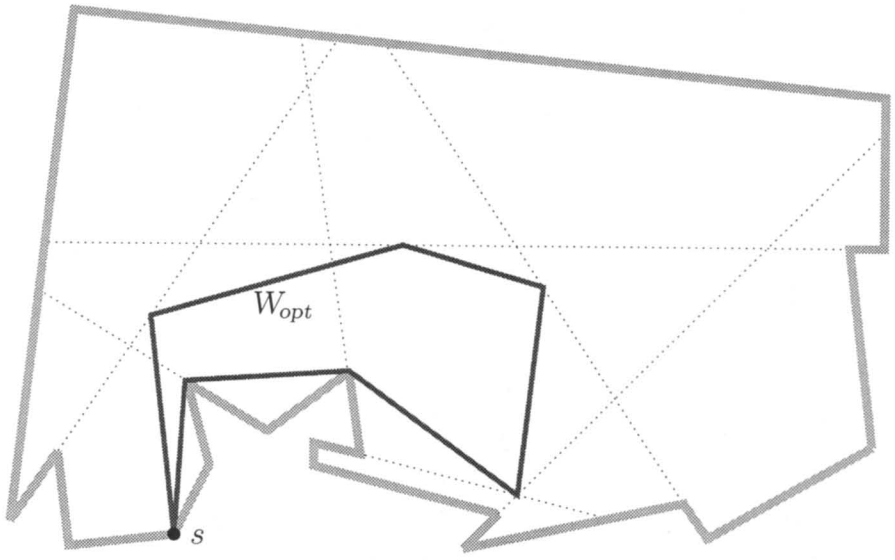


Figure 2: *The optimum watchman tour visits all cuts of the polygon.*

2 The Strategy and its Analysis

Let P be a simple polygon and let s be a point on its boundary. The *shortest path tree* of s consists of all shortest paths from s to the vertices of P. Its nodes are reflex vertices of P. Those vertices touching their shortest path from the right are called *right reflex vertices*, left reflex vertices are defined accordingly. If one follows the shortest path from s to reflex vertex v, one of its adjacent polygon edges remains invisible until v is actually reached. The extension into the polygon of this invisible edge is called a *cut*.

Exploring a polygon is equivalent to visiting all of its cuts. Figure 2 shows an example of the *optimum watchman tour*, W_{opt}, containing a boundary point, s. Tan and Hirata [9] have provided an off-line algorithm for computing W_{opt} within time $O(n^2)$, for a polygon of n edges.

We say a vertex has been *discovered* after it has been visible at least once from the robot's current position. A vertex is *unexplored* as long as its cut has not been reached, and *fully explored* thereafter.

In an unknown polygon, even exploring a single reflex vertex requires a little care. For example, one cannot afford to go straight to the vertex in order to get to its cut: The cut could be passing by the start point very closely, so that a much shorter path would be optimal.

We avoid this difficulty as follows. Whenever the robot wants to explore a right reflex vertex, r, visible from some local start point, p, it approaches r along the clockwise oriented circle spanned[4] by p and by r, denoted with $circ\,(p, r)$. Consequently, when the robot reaches the cut of r at some point c, the ratio of the length of the circular arc from p to c over their euclidean distance is bounded by $\frac{\pi}{2} \approx 1.57$.

One might wonder if the subproblem of exploring a single vertex can be solved more efficiently by using curves other than circular arcs. This is, in fact, the case; Icking et al. [8] have shown that an optimum ratio of ≈ 1.212 is achieved by curves that result from solving certain differential equations. However, these curves are lacking a useful property possessed by circular arcs: The intersection point, c, of the circular arc with the cut is just the point on the cut closest to p, due to Thales' theorem.

[4]By the circle *spanned* by two points, a and b, we mean the smallest circle that contains these points.

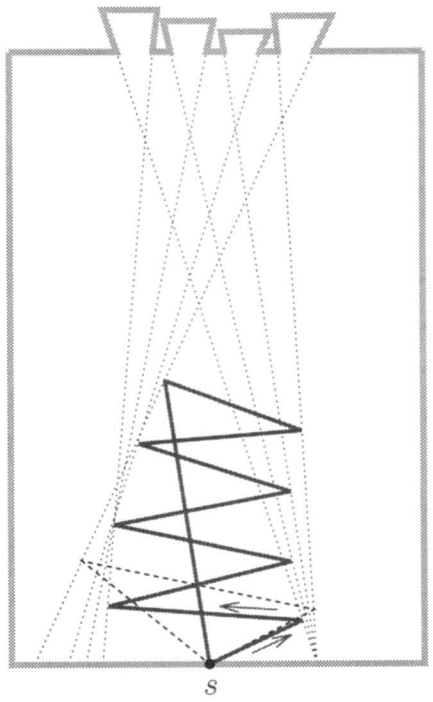


Figure 3: *Visiting cuts in the order in which their vertices appear on the boundary does not lead to a competitive strategy.*

In rectilinear polygons, the cut of each visible reflex vertex is known, and two cuts can only cross perpendicularly. This makes it possible to apply a simple greedy exploration strategy, where the robot always walks to the cut of the clockwise next reflex vertex one of whose edges is invisible; see [5].

For general polygons, this greedy approach is bound to fail, as Figure 3 illustrates. The example polygon shown there suggests exploring left and right reflex vertices separately. However, it is not really obvious how to do this in general, since e. g. the existence of a left reflex vertex at the end of a long chain of right vertices is initially not known to the robot. Therefore, it seems necessary to partition left and right reflex vertices into compact groups that can be explored one by one.

2.1 Exploring a single vertex

Each exploration of a group of right vertices[5] starts from a vertex of P called a *stage point*; these points are situated on the shortest path tree rooted at s and must also be visited by the optimum watchman tour. The first stage point encountered is the start point, s.

We would like to explore the vertices of one group in clockwise order, as they appear on the boundary of the polygon after the stage point.

But initially, the extent of a group is not fully known to the robot. When it sets out to explore the group the robot is equipped with a sorted list, denoted by *TargetList* in the procedure *ExploreRight* given below, that contains *some* unexplored right vertices of the group, among them those visible from the current position.

In this section we show how the robot explores the first element of *TargetList*. Exploring a group essentially involves iterating this procedure until the list is empty; this will be discussed in Section 2.2.

The robot starts approaching the first vertex, r, of *TargetList* on a circular arc spanned by the stage point and by r. As it proceeds, the list is maintained in the following way. All newly detected right vertices are

[5]In what follows, only right vertices, and groups of right vertices, are being addressed. Left vertices are dealt with analogously.

inserted. If one of them comes before r in clockwise order, the robot ceases approaching its old target, r, and starts exploring the vertex that is now the first of the list.

If the robot crosses the cut of a right vertex different from the vertex r currently being explored, the former vertex is removed from *TargetList* because it has been explored by the way. Eventually the target is deleted from the list, when its cut has been reached.

Procedure *ExploreRight* contains a complete description of this basic exploration strategy.

We are using the abbreviation *CP* to denote the robot's current position. In our code, *CP* is a global variable whose value can be changed only by **walk** statements of the program.

```
procedure ExploreRight ( inout TargetList,
      inout ToDoList );
  BasePoint := CP;
  Target := First (TargetList);
  if Target not visible then
    walk on shortest path from BasePoint to Target
    until Target becomes visible;
  Back := last vertex before CP of shortest path
        from BasePoint to CP;
  walk clockwise along circ (Back, Target)
    while maintaining TargetList and ToDoList
    whenever First (TargetList) changes
          let Target := First (TargetList);
    whenever Back becomes invisible update Back;
    exceptions for walking along the circle:
      if the boundary of P blocks
            the walk on the current circle then
        walk clockwise along the boundary
        until the circular walk is again possible;
      if Target is becoming invisible then
        walk towards Target
        until the blocking vertex is reached;
  until Target is fully explored;
end ExploreRight;
```

Besides exploring right vertices, *ExploreRight* serves a second purpose. When a right reflex vertex is explored, all of its children in the shortest path tree have already been discovered. Those vertices having a left child are inserted into a list, *ToDoList*, as candidates for future stage points, together with references to their left children.

The point from which the robot starts executing procedure *ExploreRight* is called a *base point*. The first base point of each group exploration is always the group's stage point itself. However, exploring one group can generate many base points, among them such that do not lie on the boundary of P.

When *ExploreRight* is invoked, the first vertex of *TargetList* need not be visible from the base point; the robot may have detected it at an earlier stage. In this case the robot walks along the shortest path towards this vertex until it becomes visible, and continues from there.

We summarize the rules by which *ExploreRight* proceeds.

1. The current *target* vertex, i. e. the vertex whose cut we are intending to reach at the moment, is always the clockwise first among those right reflex vertices that have been discovered but not yet fully explored, i. e. the first element of *TargetList*.
2. To explore a right reflex vertex, r, we follow the clockwise oriented circle spanned by r and by the last vertex before *CP* on the shortest path from s to *CP*.
3. When the view to the current target vertex gets blocked (or when the boundary is hit) we walk straight towards the target (or follow the boundary) until motion according to (2) becomes possible again.

Figure 4 demonstrates how this strategy works. Initially, r_3 is the only right vertex visible, so that the robot's path begins with a circular arc spanned by s and by r_3. At point a, right vertex r_2 becomes visible. It is situated before r_3 on the boundary; therefore, the robot switches to exploring r_2, according to rule 1.

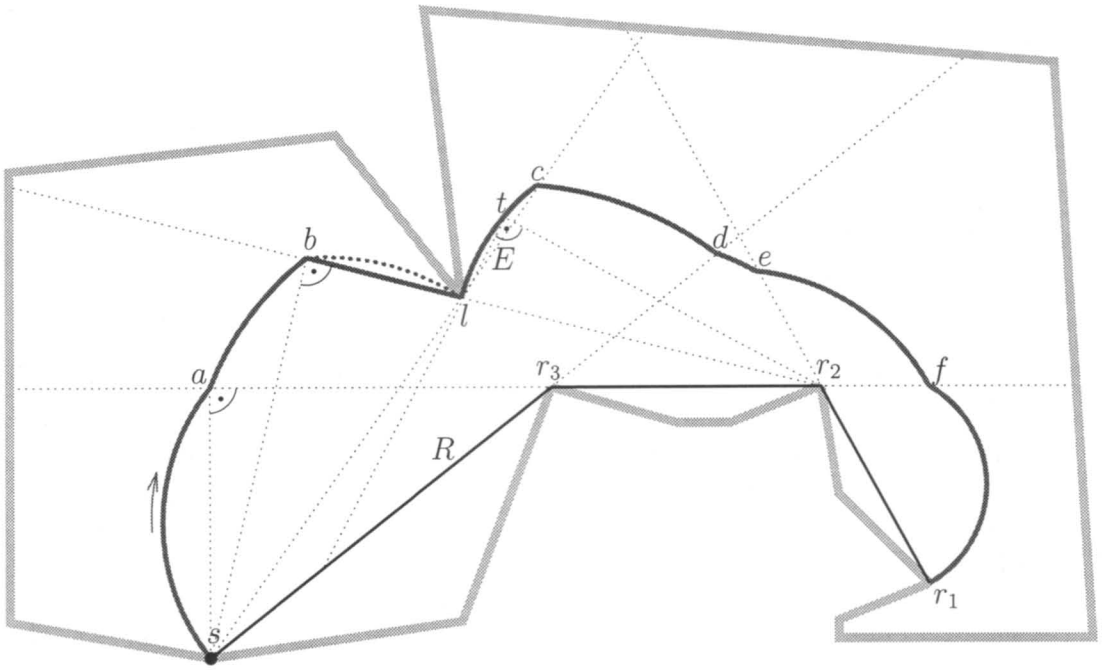


Figure 4: *While executing ExploreRight, the target vertex is initially r_3, then changes to r_2 and finally to r_1.*

Note that the circle spanned by s and by r_2 is passing through a, too, so that it is in fact possible to apply rule 2 at this point.

At point b, vertex r_2 would become invisible if the robot were to follow the circular arc. Now rule 3 applies, causing the robot to walk straight to the left reflex vertex l. From there, a circular motion is again possible; but the shortest path from s to CP now contains vertex l. By rule 2, the robot continues its approach to r_2 along the arc spanned by l and by r_2.

From c on, the shortest path to s is the line segment. Since the circle spanned by s and by r_2 is passing through c the robot has no difficulties in applying rule 2. At d, the shortest path to s changes again; now it contains vertex r_3. The robot changes its path accordingly and gets to point e from which vertex r_1 becomes visible. From here, the robot explores r_1. Its path follows the circle spanned by r_3 and by r_1, the former changing to r_2 at f. Eventually, the robot arrives at r_1, thereby fully exploring r_1. Here procedure *ExploreRight* terminates.

Now we provide an upper bound to the length of the resulting path. The crucial observation is the following.

Lemma 2 *Suppose procedure ExploreRight terminates with the robot reaching the cut of target vertex r_1. Then the robot's path is part of the boundary of the angle hull $\mathcal{AH}(R)$ of the shortest path, R, from the base point to r_1, except for straight line segments leading to blocking vertices.*

Proof: Let t be a point on the robot's path that is not contained on a straight line segment. Assume that, at t, the robot is exploring right reflex vertex r_2, as in the example shown in Figure 4. Since r_2 and r_1 are in convex position, vertex r_2 lies on path R.

Now consider the shortest path, T, from the stage point to t. As a consequence of rule 2 and by Thales' theorem, the last line segment, E, of T is perpendicular to the line through t and r_2. Since the backward prolongation of E is bound to hit R, we know that point t can see two points of R at a right angle. Thus, t be-

longs to the angle hull $\mathcal{AH}(R)$; it lies on the boundary because the angle's sides are both touching path R. □

If the robot's path contains straight line segments leading to blocking vertices, like the segment from b to l in Figure 4, these segments are replaced with circular arcs in the angle hull $\mathcal{AH}(R)$. Thus, the robot's path to the cut of r cannot be longer than the angle hull's perimeter. If it ends at the point r itself, as in Figure 4, we can directly apply Theorem 1 to Lemma 2 and obtain the desired upper bound. If the robot reaches the cut of r at some point different from r, one can apply a suitable modification of Theorem 1.

We have obtained the following result.

Lemma 3 *The robot's path from the base point to the cut of the target vertex explored by procedure ExploreRight is not longer than twice the length of the shortest path.*

There is a symmetric procedure *ExploreLeft* which is identical to *ExploreRight*, except that left/right, and clockwise/counterclockwise are exchanged.

2.2 Exploring a group of vertices

In Figure 4, the vertices r_2, r_3 have already been explored when *ExploreRight* terminates at r_1, and *TargetList* has become empty. This need not be the case in general. We now discuss how to deal with the rest of this list.

As mentionend before, the current position where *ScoutRight* starts is called a *stage point*. When *ScoutRight* is called, *TargetList* contains at least all unexplored right vertices visible from *StagePoint*. The shortest path from s to each of these vertices passes through *StagePoint*.

The task of *ScoutRight* is, by using *ExploreRight*, to explore the vertices in *TargetList* and all those right vertices which become visible during the exploration. This set of vertices constitutes a *group*. It will turn out that the vertices explored by one call of *ScoutRight* are the stage point's *purely right descendants* in the shortest path tree from s, i.e. those whose shortest path from the stage point has only right turns.

```
procedure ScoutRight (in TargetList, out ToDoList);
  StagePoint := CP;
  ToDoList := empty list;
  while TargetList is not empty do
    ExploreRight (TargetList, ToDoList);
    (* CP is now on the cut, C, of the last target. *)
    walk to closest point to s on C
      while maintaining TargetList and ToDoList;
  walk on the shortest path back to StagePoint;
end ScoutRight;
```

TargetList is an input parameter which is used and changed as appropriate. *ToDoList* is an output parameter which is initialized to be empty. It will be incrementally added to by *ExploreRight*, as explained in Section 2.1, and also in *ScoutRight* itself during the **walk** step. Finally, *ToDoList* contains, of all vertices explored, those who have left children, together with references to the latter.

Each call of *ExploreRight* explores the first vertex of *TargetList*; however *First* (*TargetList*) can change during the exploration, as we have seen in Section 2.1. After returning from *ExploreRight* the *CP* is on the cut just reached. Now the robot walks along the cut to the point closest to the stage point. This point will be the next base point. The reason for this step will become clear in the proof of Lemma 5.

For an example, see Figure 5. Point s is the stage point and also the first base point, and *ExploreRight* is called with *First* (*TargetList*) $= r_6$. While exploring r_6, point r_1 is discovered at point a and becomes *First* (*TargetList*). At $CP = b$ procedure *ExploreRight* returns. Meanwhile, r_2 and r_5 have been added to *TargetList* while r_1 and r_6 have been removed. Point b is also the closest point to s on the current cut.

As we continue with exploring *First* (*TargetList*) $= r_2$, point r_5 gets explored by the way. Once the cut of r_2 is reached, we walk to c, the closest point to s on the cut. Similar for r_3; while walking along the cut to the point closest to s, which is r_3 itself, r_4 gets explored and no unexplored right vertices remain.

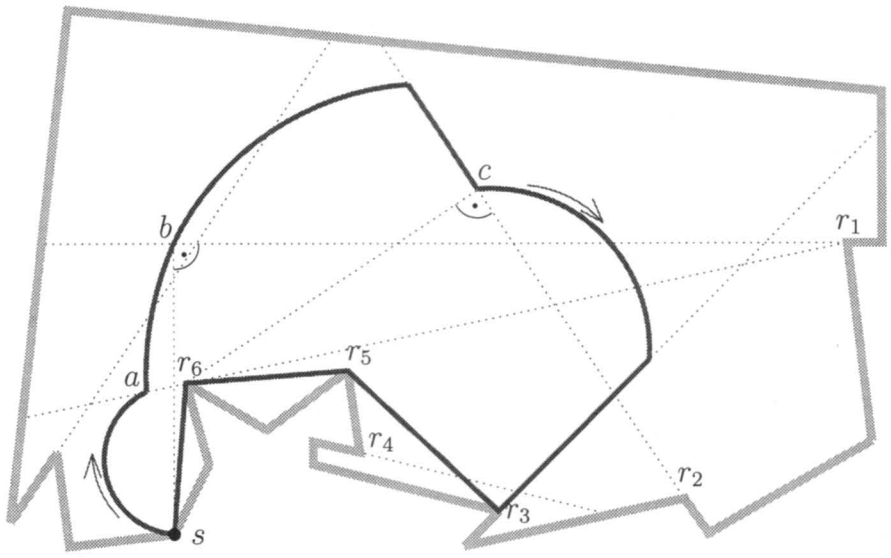


Figure 5: *Exploring a group of right vertices.*

As before with *ExploreRight*, for *ScoutRight* we also have a symmetric counterpart, *ScoutLeft*.

Now let us estimate the length of the path travelled.

Lemma 4 *The robot's path between two consecutive base points is at most* 3 *times as long as the shortest path.*

Proof: Let us call the base points b_1 and b_2, and let c be the point where the cut is reached. Then c is also the cut's closest point to b_1, by property 2 of *ExploreRight*. By Lemma 3, the robot's path to c is not longer than twice the length of the shortest path from b_1 to c and therefore is also not longer than twice the length of the shortest path from b_1 to b_2, see Figure 6.

It remains to account for the walk along the cut from c to b_2. This line segment can be orthogonally projected onto the shortest path from b_1 to b_2 and therefore must be shorter. □

The next steps consist in comparing the length of a *ScoutRight* tour with the relative convex hull of the base points visited.

Lemma 5 *The length of a path caused by a call to ScoutRight does not exceed* $3\sqrt{2}$ *times the perimeter of the relative convex hull of the base points visited.*

Proof: Let sp be the stage point and $sp = b_0, \ldots, b_{m-1}, b_m = sp$ the sequence of base points visited by *ScoutRight*. Note that $b_1, \ldots, b_{m-1}$ appear in this clockwise order as leaves of the shortest path tree from s to $b_1, \ldots, b_{m-1}$. So even factor 3 of Lemma 4 would apply if all base points b_i were vertices of their relative convex hull, $\mathcal{RCH}$, in P. Suppose that b_{i-1} and b_{i+1} are situated on $\mathcal{RCH}$ but b_i is not; see Figure 7. If b_i were too close to s then its cut would intersect every possible path from s to b_{i-1}, in particular the robot's path, contradicting the fact that vertex r_i is explored *after* r_{i-1}. Therefore, the cut of b_i must pass above b_{i-1}, and it follows that

$$|b_{i-1}b_i| + |b_ib_{i+1}| \leq \sqrt{2}\,|b_{i-1}b_{i+1}|.$$

This argument generalizes to more than three points. □

In relating the robot's path to the optimum watchman tour, the following lemma is crucial.

Lemma 6 *All base points visited are contained in the angle hull* $\mathcal{AH}(W_{opt})$.

Proof: A base point b is, by definition, the closest point to s of a cut. The optimum watchman tour W_{opt}

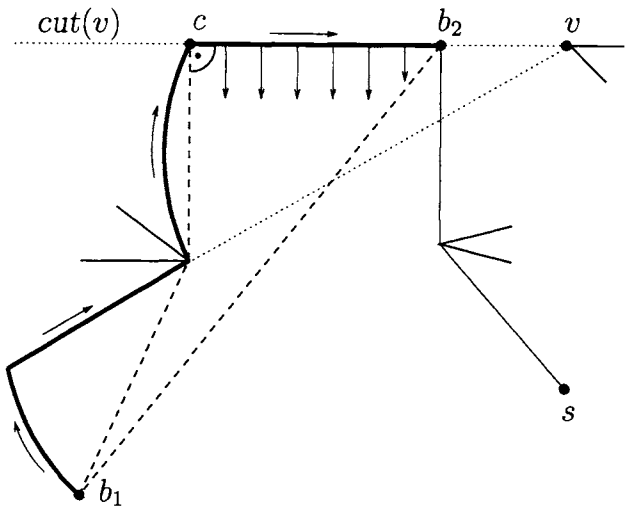


Figure 6: *Line segment $c\,b_2$ must be shorter than the shortest path from b_1 to b_2.*

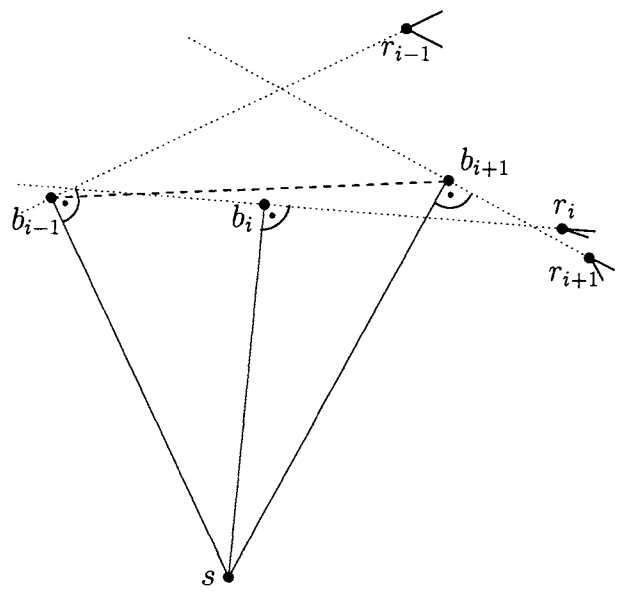


Figure 7: *The cut of vertex r_i containing the base point b_i must not intersect the shortest path from s to b_{i-1}.*

connects the start point s to the cut. Let E be the last edge of the shortest path from s to the cut, i. e. to b, as in procedure *ExploreRight*.

In most cases, edge E is orthogonal to the cut. Then we have a right angle at b whose one side goes along the cut and touches W_{opt}, while the other side K extends edge E. Either the other endpoint of E equals s, so that K touches W_{opt} in s, or K separates s and the cut, and W_{opt} must also be touched by K.

In the remaining case, when there is no right angle between edge E and the cut, point b must be one endpoint of the cut, i. e. it is the target vertex itself or the other endpoint. By similar arguments there is an angle of $> 90°$ both of whose sides touch W_{opt}. □

2.3 Subdividing the polygon

Now we want to combine the exploration of several groups of vertices to finally explore the whole polygon P. This is done by making the *Scout*-procedures recursive.

procedure *ScoutRightRec* (**in** *TargetList*);

ScoutRight (*TargetList*, *ToDoList*);
(* *ToDoList* gets filled in. *)

Clean up *ToDoList*:
retain only those right vertices in *ToDoList*,
which are highest up in the shortest path tree;

for all vertices v of *ToDoList* in clockwise order **do**
walk on the shortest path to v;
(* connect stage points *)

ScoutLeftRec({all known left descendants of v in counterclockwise order});

end *ScoutRightRec*;

The task of *ScoutRightRec* is to explore, from the current position *CP*, all vertices in the input parameter *TargetList* and everything behind.

ScoutRightRec performs in three steps. The *TargetList* is handed over to *ScoutRight*, so *CP* is the new stage point, and after the exploration we are back at this point. We are given a *ToDoList* of candidates for stage points in recursive explorations.

The next step is a necessary cleanup for the *ToDoList*, which contains all purely right descendants of the current stage point which have left children. Some of these right vertices are descendants of others in this list, they must be removed from the list. Only maximal (highest up) right vertices are retained, these will become stage points in further steps. To each of these future stage points we associate a list of all known left descendants that were referenced in *ExploreRight* and *ScoutRight*.

Finally, the remaining vertices in *ToDoList* are visited in clockwise order, at each vertex procedure *ScoutLeftRec* is called to explore the list of all known left descendants (as *TargetList*) from there. *ScoutLeftRec* is the symmetric counterpart of *ScoutRightRec* with one particularity. In the **for** loop, the vertices in *ToDoList* *are also visited in clockwise order*. The reason for this will become clear in the proof of Theorem 10.

To conclude the bottom-up presentation of our strategy, we show the main program. Its task is, of course, to explore a polygon, P, starting at a boundary point s. First, in a call to the non-recursive *ScoutRight*, the right vertices visible from s are explored. The next target list contains all left children of the right vertices just explored and the left vertices visible from s. All these, and everyting behind, gets explored by a call to the recursive *ScoutLeftRec* with this target list.

procedure *Main* (**in** P, **in** s);

ScoutRight ({clockwise list of all right vertices visible from s}, *ToDoList*);

walk on the shortest path back to s;

TargetList:=
{all left children of the vertices of *ToDoList*};

Add all left vertices visible from s into *TargetList* and sort counterclockwise;

ScoutLeftRec (*TargetList*);

walk on the shortest path back to s;

end *Main*;

Each call of *ScoutRight* or *ScoutLeft* generates a set of base points. For estimating the length of the complete tour, we distribute all these sets into three categories.

The set of base points generated by the call of *ScoutRight* in *Main* belongs to category 0. For the remaining sets, we use their level of recursion to determine their category: the set of base points generated by a call of *ScoutRight* or *ScoutLeft* at total recursion depth i belongs to category (i mod 3).

For example, the very first call of *ScoutLeft* belongs to category 1, and the calls of *ScoutRight* one level deeper belong to category 2. All calls of *ScoutLeft* have an odd level, and all calls of *ScoutRight* an even level.

A key observation is the following.

Lemma 7 *The relative convex hulls of two sets of base points of the same category are mutually invisible, with a possible exception for their stage points.*

Proof: There are two main cases.

If two sets of base points have the same recursion depths then they have stage points of the same type, left or right. One stage point may be able to see some points of the other hull, but it hides all of its own hull from each point of the other set.

Otherwise, their recursion depths differ by at least three and a similar argument holds. □

As a consequence, we conclude that the union of all base points of one category has no shorter perimeter than the perimeters of all of its sets of base points together. Let $\mathrm{per}(\mathcal{RCH}(A))$ denote the perimeter of the relative convex hull of set A.

Lemma 8 *Let B_1 and B_2 be two sets of base points of the same category. Then we have per$(\mathcal{RCH}(B_1))$ + per$(\mathcal{RCH}(B_2)) \leq$ per$(\mathcal{RCH}(B_1 \cup B_2))$.*

As a consequence, we can estimate the path length caused by all calls of *ScoutRight* or *ScoutLeft* in the same category.

Lemma 9 *The path length caused by all calls of ScoutRight and ScoutLeft in one category is less than $6\sqrt{2} \leq 8.5$ times the length of W_{opt}.*

Proof: Let the category consist of sets B_i, $i = 1, \ldots$ of base points. By Lemma 5, the length of the path created by one call of *ScoutRight* or *ScoutLeft* with set B_i is not greater than $3\sqrt{2}\,\mathrm{per}(\mathcal{RCH}(B_i))$.

The relative convex hulls $\mathcal{RCH}(B_i)$ are mutually invisible (Lemma 7), hence we conclude from Lemma 8 for the path length, L, caused by all calls of *ScoutRight* or *ScoutLeft* of this category

$$L \leq 3\sqrt{2} \sum_i \mathrm{per}(\mathcal{RCH}(B_i)) \leq 3\sqrt{2}\,\mathrm{per}(\mathcal{RCH}(\bigcup_i B_i)).$$

All base points considered are contained in the angle hull of W_{opt}, as Lemma 6 has shown, hence the perimeter of their relative convex hull is shorter than the perimeter of $\mathcal{RCH}(\mathcal{AH}(W_{opt}))$.

The perimeter of $\mathcal{RCH}(\mathcal{AH}(W_{opt}))$ is not longer than the perimeter of the angle hull of W_{opt} itself. By Theorem 1 this is not greater than twice the length of W_{opt} and the claim follows. □

As the main result for our complete strategy, we obtain a factor of 26.5.

Theorem 10 *For a polygon, P, and a start point s on the boundary of P, a procedure call Main (P, s) explores the polygon and returns to s. The total path length used is less than $(18\sqrt{2} + 1) \leq 26.5$ times the length of the optimum watchman tour from s.*

Proof: Since we have three categories of base point sets, all *ScoutRight* and *ScoutLeft* calls together cause a path length of less than $3 \cdot 3\sqrt{2}|W_{opt}|$.

It remains to bound the path length caused by the walks during the **for** loops of *ScoutRightRec* and *ScoutLeftRec*. They only connect stage points by shortest paths, and all those stage points are visited in clockwise order along the boundary of P, independently of whether this is done in *ScoutRightRec* or *ScoutLeftRec*.

But all stage points are also visited by W_{opt} in the same order. Therefore, we can be sure that all those

walks together make up for an additional path length of at most $|W_{opt}|$. □

3 Conclusions

We have seen that a combination of suitable analysis techniques is necessary for proving an upper bound for the competitive factor of a rather simple strategy. We believe that the actual performance is considerably lower still.

Surely, there are many interesting variations and generalizations of the polygon exploration problem. For example, one could study different cost models for the robot's motion. Also the case of polygons with holes deserves investigation.

Acknowledgement

This work was partially supported by the Deutsche Forschungsgemeinschaft, grant Kl 655/8-2.

References

[1] S. Carlsson and H. Jonsson. Computing a shortest watchman path in a simple polygon in polynomial-time. In *Proc. 4th Workshop Algorithms Data Struct.*, volume 955 of *Lecture Notes in Computer Science*, pages 122–134. Springer-Verlag, 1995.

[2] S. Carlsson, H. Jonsson, and B. J. Nilsson. Finding the shortest watchman route in a simple polygon. In *Proc. 4th Annu. Internat. Sympos. Algorithms Comput. (ISAAC 93)*, volume 762 of *Lecture Notes in Computer Science*, pages 58–67. Springer-Verlag, 1993.

[3] W.-P. Chin and S. Ntafos. Watchman routes in simple polygons. *Discrete Comput. Geom.*, 6(1):9–31, 1991.

[4] X. Deng, T. Kameda, and C. Papadimitriou. How to learn an unknown environment. In *Proc. 32nd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 298–303, 1991.

[5] X. Deng, T. Kameda, and C. H. Papadimitriou. How to learn an unknown environment I: the rectilinear case. Technical Report CS-93-04, Department of Computer Science, York University, Canada, 1993.

[6] F. Hoffmann, C. Icking, R. Klein, and K. Kriegel. A competitive strategy for learning a polygon. In *Proc. 8th ACM-SIAM Sympos. Discrete Algorithms*, pages 166–174, 1997.

[7] F. Hoffmann, C. Icking, R. Klein, and K. Kriegel. Moving an angle around a region. Technical report, Department of Computer Science, FernUniversität Hagen, Germany, 1998.

[8] C. Icking, R. Klein, and L. Ma. How to look around a corner. In *Proc. 5th Canad. Conf. Comput. Geom.*, pages 443–448, 1993.

[9] X. Tan and T. Hirata. Constructing shortest watchman routes by divide-and-conquer. In *Proc. 4th Annu. Internat. Sympos. Algorithms Comput.*, volume 762 of *Lecture Notes Comput. Sci.*, pages 68–77. Springer-Verlag, 1993.

The Computational Power of Frictional Mechanical Systems

John Reif, *Duke University, Durham, NC, USA*
Zheng Sun, *Duke University, Durham, NC, USA*

In this paper we define a class of mechanical systems consisting of rigid objects connected by frictional contact linkages between surfaces. We prove that a universal Turing Machine (TM) can be simulated by a (universal) frictional mechanical system in this class. Our universal frictional mechanical system has the property that it can reach a distinguished final configuration through a sequence of legal movements if and only if the universal TM accepts the input string encoded by its initial configuration. There are two implications from this result. First, the mover's problem is undecidable when there are frictional linkages. Second, a mechanical computer can be constructed that has the computational power of any conventional electronic computer and yet has only a constant number of mechanical parts.

In addition, we discuss the universal frictional mechanical system in the context of an error model that allows error up to ϵ in each mechanical operation. We show that for a universal TM M, a frictional mechanical system of this ϵ-error model can be constructed such that, given any space bound S, the system can simulate the computation of M on any input string ω if M decides ω in space bound S, provided that $\epsilon < 2^{-cS}$ for some constant c.

1 Introduction

There are two contexts where our result may be of interest: the robotics mover's problem and mechanical computing machine.

1.1 Robotics mover's problems

The *robotics mover's problem* (or the *motion planning problem*) is to plan the motion of a robot between distinguished configurations under specifie physical constraints (e.g., avoiding obstacles) and possibly also kinodynamic constraints. There are two categories of problems studied in this area: one is the decision problem and the other is the optimization problem. The decision problem is to decide, given the physical and kinodynamic constrains, whether there is a sequence of legal movements for the robot to reach the target configuration, while the optimization problem is to find the optimal path (trajectory) that leads to the target configuration, according to a predefined cost function on the path (trajectory).

The first hardness result for a mover's problem was given by Reif [6]. He showed that the generalized mover's problem of moving n linked polyhedra through a set of 3-D obstacles is PSPACE-hard. He proved this by reducing the computation of any reversible Turing Machine on an input string to an instance of the generalized mover's problem. Hopcroft, Joseph and Whitesides [4] improved this result by proving that the mover's problem for 2-D linkages is PSPACE-hard. Later the generalized mover's problem was proved to be in PSPACE by Canny [1]. In [5] Hopcroft, Schwartz and Sharir proved that motion planning for multiple independent rectangular boxes sliding inside a rectangular box is PSPACE-hard. Reif and Sharir [7] introduced the 3-D mover's problem in the presence of moving obstacles and showed that this problem is PSPACE-hard even in the case where the object to be moved is a disc with bounded velocity.

1.2 Frictional mechanical systems and the frictional mover's problem

All the models in the works in robotics mover's problems mentioned above assume that there is no friction

between objects, and most of the models only allow collision-free movements so that different objects cannot even contact with each other. The only work that addressed on motion planning in the presence of friction is [9] by Sellen. He proved that dynamic motion planning problem with forbidden movements (in particular, sliding) is undecidable by showing that the actions of a TM can be realized by logical and arithmetic operations which can be implemented by mechanical means. However, he did not provide a construction of mechanical system that simulates a TM.

We define a *frictional mechanical system* to be a collection of rigid objects in 3-D space whose surfaces are composed of linear or quadratic surface patches specified by rational coefficients. All objects are non-penetrable, i.e., the only allowed intersection is via surface contact. Each surface patch of each object is also specified as either *frictional* or *sliding* (non-frictional). If two objects with frictional surfaces contact with each other, it is assumed that there is no sliding[1] between them. If at least one of the two contacting surfaces is designated to be a sliding surface, there will be no friction between them so that they can slide freely. Furthermore, some objects in the space are specified to move monotonically. In particular, there may be some discs in the system that can only rotate in one direction. We say that the objects can be moved *legally* if all the above constraints are satisfied.

We define the "resource bound", denoted by R, to be the number of distinct objects in the frictional mechanical system. As each object can be specified by a constant number of surface patches, each of which can be specified by a constant number of rational coefficients, the total number of binary bits used to encode an object is bounded by a constant. Hence, the total number of bits in the binary representation of the frictional mechanical system is $O(R)$.

The *frictional mover's problem* is to determine whether these objects in the frictional mechanical system can be moved legally from a specified initial configuration to a specified final configuration. We prove that the frictional mover's problem is undecidable by reducing the acceptance problem for Turing Machine[2] (TM), A_{TM}[3], to the frictional mover's problem. Given a universal TM [4] M, we construct a frictional mechanical system to simulate this machine. This frictional mechanical system will have the property that the objects in this system can be moved from an initial configuration, which encodes an input string ω of M, to a configuration corresponding to the accepting state of M if and only if M accepts ω. Therefore, as the acceptance problem for Turing Machine is undecidable, so is the frictional mover's problem. This implies that there is no realistic machine that can solve this problem.

An interesting property of this frictional mechanical system is that, if M accepts ω, there will be a unique simple path (i.e., one that does not repeat the same configuration) from the initial configuration to the final configuration.

The proof will actually construct, for any given TM M, a frictional mechanical system that simulates M. Every movable object in the system is engaged or linked directly or indirectly with a "power disc" so that, when the power disc rotates, it will make those objects move accordingly. For any input string ω of M, this frictional mechanical system can be set to an initial configuration encoding ω so that, after the "power disc" has rotated a sufficient number of cycles, this system will result in a configuration encoding the accepting state (rejecting state) of M, if and only if M accepts (rejects) ω.

This problem can be regarded as a generalization of the *movable object problem*[10] (see also [3], [2]). The movable object problem is to decide whether a robot

[1]Sliding is a move in a direction tangent to the surfaces at the contact point.

[2]A Turing Machine is an abstract machine with a finite state control and a tape that can store an infinite string of symbols. There is a read-write head on the tape which allows the machine to read or write the symbol at the current position of the head. The machine can write to the current position and move the head left or right according to the current state and the current symbol in a specified way.

[3]The acceptance problem A_{TM} for Turing Machine is to determine, given the description of a TM M and its input ω, whether M accepts ω.

[4]A universal TM M will take the description of any TM M' and any input string ω of M' as an input and simulate the behavior of M' presented with input ω.

can move the objects amidst obstacles in the space to reach a target configuration. That is, there are two types of objects in the space: obstacles that can not be moved (or even touched) by the robot; and objects that are movable by the robot. And the goal of the robot is to either rearrange the movable objects in the space to a desired configuration or reach a target configuration of itself (or both).

Compared to the previous works on the movable object problem, our model is in 3-D and the surfaces of the objects in the space can be non-linear. Further, in addition to moving an object by "grasping" or "pushing" (directly on indirectly), a robot in our model can move objects by using the friction between it and surrounding objects. More specifically, the power disc in the frictional mechanical system can be deemed as a rather "dumb" robot, who is restricted to rotate in a specified direction without translation. And the problem is to ask whether this robot can rearrange the objects in the system to a target configuration by its rotation. In our following discussion, we will refer our problem as the "frictional mover's problem" although all the discussion applies to this generalization of the movable object problem as well.

1.3 Frictional mechanical system with error

We prove that a frictional mechanical system can be constructed to simulate a universal TM. Therefore, this system can be used for arbitrary finite computation, just like any conventional computer. However, the underlying assumption is that this frictional mechanical system can be constructed exactly as it is specified. We are also interested in the computational power of such a frictional mechanical system in the case inaccuracy is allowed in the construction of the mechanical devices in the system.

There are many factors which might induce error in the computations by a frictional mechanical system including the precision of manufacture of the parts. For example, the circumference of a disc may not exactly be manufactured to be a circle. The radius of a disc may not be manufactured to be exactly as it is specified. When two discs are very close but still not in contact with each other theoretically, they may already have surface contact so that the rotation of one disc will move the other one, even though they are not supposed to do so.

Since there are constant number of mechanical devices in our mechanical system, we can let ϵ to be the upper bound for the errors which occur in a single operation. This is our ϵ-error model. We prove that, given a space bound S, our frictional mechanic system in this ϵ-model can simulate the universal TM M on any input string ω which can be decided by M in space bound S, if $\epsilon = O(2^{-cS})$ for some constant c.

We also prove that, given a universal TM M with space bound S, there exists an $\epsilon = O(1)$ such that a frictional mechanical system in ϵ-error model can simulate the computation of M presented with any input ω if M decides ω in space bound S. This result provides decreased required precision of parts at the expense of increased number of parts, which increases with S.

1.4 Mechanical computing machines

In 1822, Charles Babbage designed and constructed the Difference Engine. This machine was specially designed for the evaluation of polynomials. Later in 1833, Babbage proposed (but did not fully construct) a new device, Analytical Engine, which was conceived to solve general arithmetic problems.

About half a century after Babbage's death, Dr. Vannevar Bush resumed the work of building a mechanical computing device. In 1925 he, along with some associates, made a mechanical calculator powered by electric motor. His machine was an analog one, in the sense that arithmetic operations were carried out by mechanical means and in terms of physical measurements.

In 1939, Howard Aiken, in collaboration with four I.B.M. engineers, built a general purpose computing machine, the so called Automatic Sequence Controlled Calculator, Mark I. Just like Babbage's Analytical Engine, Mark I performed computation by manipulating mechanical devices.

Even though Babbage claimed that the Analytical Machine could be used to solve any arithmetic problem, it is doubtful whether this machine is as powerful as a

universal computing machine. The difficulty is that Babbage did not have a general concept of an abstract computing machine, such as a Turing Machine (TM), which was introduced by Alan Turing in 1936. To show that a mechanical system has the computational power of an electronic computer, it is sufficient and necessary to show that this mechanical system can simulate a universal TM, as proposed by Turing. Although Babbage designed his Analytical Machine to have a control unit used to guide the arithmetic operations, it was not explicitly shown by Babbage how it could simulate a general finite state control, which is the core of a universal TM. The electro-mechanical computer, Mark I, was designed to be capable of fulfilling any computing tasks and thus should be as powerful as the modern computers. However, it used an electronic device as the central controller. To our knowledge, our mechanical system, as it will be described in later sections, is the first mechanical system that can perform general-purpose computation without using any electronic devices.

Another limitation of Babbage's Analytical Engine is its representation of numbers. The Analytical Engine was a digital computer. Each digit of a number was represented by a mechanical device, such as a dial. The accuracy of Babbage's machine, in terms of number of decimal places, was determined by the number of dials used to represent a number. Therefore, no matter how precisely the machine was built, it would not improve its accuracy. The only way to improve the accuracy of the machine's computation (or increase the number of data bits) is to increase the number of dials used to represent a number. This will change the structure of the Analytical Engine and make it bigger and more complicated. Our frictional mechanical system, however, is an analog computer, in the sense that a number (in fact, the entire data of the computation) is represented by the rotational position of a single disc. The accuracy of our system depends on the accuracy of the measurement and the precision of the mechanical devices. Therefore, it can be arbitrarily accurate as we reduce the error in transitional (or rotational) measurement as well as the error in building mechanical devices.

1.5 Notations

In the following sections, the universal TM with endmarks is denoted by quintuple $M = (Q, \Sigma, \delta, q_0, \{q_{\sigma-2}, q_{\sigma-1}\})$ as follows:

1 $Q = \{q_0, q_1, \cdots, q_{\sigma-1}\}$ is the set of states.
2 $\Sigma = \{0, 1, \cdots, m-1\}$ is the tape alphabet. Here 0 denotes the *blank* symbol. 1 and 2 are the left and right endmarks respectively.
3 $\delta : Q \times \Sigma \rightarrow Q \times \Sigma \times \{L, R\}$ is the transition function.
4 q_0 is the start state.
5 $\{q_{\sigma-2}, q_{\sigma-1}\}$ is the set of halting states. Particularly, $q_{\sigma-2}$ is the rejecting state and $q_{\sigma-1}$ is the accepting state.

At anytime during the computation, we use the "current working space" of the TM to denote the portion of the tape where the read-write head has visited so far. The current working space is always bounded by the a left endmark and a right endmark. We denote the tape status by a string $\omega_1\omega_2\cdots\omega_{k_1-1}\check{\omega}_{k_1}\omega_{k_1+1}\cdots\omega_{k_2}$. Here ω_1 and ω_{k_2} are the left and right endmark respectively, and $\check{\omega}_{k_1}$ denotes that read-write head is at the k_1-th cell of the tape. The read-write head will never replaces the left endmark by another symbol nor will it go beyond the left endmark. Whenever the read-write head replace the right endmark by another symbol, it will pad a right endmark to the right of the symbol.

It is easy to see that, for any TM (without endmarks) M' , there is a equivalent TM with endmarks as described above. Therefore, all universal TMs mentioned in the following discussion are assumed to have endmarks.

Since discs, in particular, the rotational positions of discs, play a very important role in our frictional mechanical system, we want to specify several terminologies which we will be using frequently in the following discussion.

Each disc used in our system has a specified orientation called the "initial orientation". Therefore, the rotational position of a disc D can be specified by the angle the disc has rotated from its initial orientation. For our convenience, we will use "the angle of D" to denote this angle. Also, we will say that the angle

recorded (or represented) by the rotational position of disc D is θ. Further, if we say "to increase (decrease) the angle or rotational position of a disc D by an angle of θ", we mean to rotate D counterclockwise (clockwise, respectively) by an angle of θ.

A *partial disc* is a portion of a disc bounded by two radii and the remaining portion of the circumference. As we will show later, partial discs are very useful in our system too.

1.6 Organization of this paper

Section 2 presents several basic mechanical devices widely used in our frictional mechanical system. After that, we provide the full description of the frictional mechanical system that simulates a universal Turing Machine. In the last section, the model of the frictional mechanical system with errors is discussed.

2 Basic Gadgets

We prove the reduction from A_{TM} to the frictional mover's problem by constructing a frictional mechanical system for a universal TM M. This system is composed of discs, partial discs, cylinders, and other geometric objects. There is a special disc (the "power disc") which is specified to only rotate clockwise. The rotation of this disc will force other objects in the system to move, due to frictional linkages between them. This frictional mechanical system can "simulate" M in the sense that, for any input string ω of M, this system can be set to a corresponding initial configuration so that a distinguished final configuration can be reached if and only if M accepts ω. Therefore, if one can decide this instance of frictional mover's problem, he can also decide the corresponding instance of A_{TM}. As it is well known that A_{TM} is undecidable, so must be the frictional mover's problem.

Friction is very important in this system as it not only provides method for moving other objects in the system but also preserves the state of the system and thus the state of the Turing Machine the system simulates. It also guarantees the system properly transfers from one state to another.

Before giving the construction of the frictional mechanical system, we first introduce in this section several "gadgets" which perform the basic functionalities and thus are used widely in our system. For details, please see the full version of this paper[8].

There are seven kinds of basic gadgets:

Converting device. A converting device is used to convert between rotational displacement and transitional displacement.
Sequencer. The sequencer is a key component in the frictional mechanical system as it provides mechanism for moving all the objects in the system in a specified way.
Transitional Movement Sequence Controller. A transitional event sequencer controller allows the sequencer to periodically move an object in the system by a constant distance and then move it back.
Rotational Movement Sequencer Controller. A rotational event sequencer controller allows the sequencer to periodically rotate a disc by a constant angle.
Resettable Rotational Disc. A resettable rotational disc has two states: the free rotation state when it can be rotated by engaging with another disc; and the reset state when it will return to a specified initial orientation no matter how much it is rotated in the free rotation period. Virtually all devices need to use resettable rotational discs.
Non-linear Mapping Controller. A non-linear mapping device is used to implement a non-linear mapping between rotations. This is a very crucial component in the construction of the simulation of a finite state control.
Selection Controller. A selection controller helps the sequencer to initiate one among several event sequences depending on the rotational (or transitional) position of an object in the system. This is yet another key component in the construction of the simulation of a finite state control.

In the following subsections, we will give detailed description for each of the basic gadgets.

2.1 Converting device

The most basic device is one that can convert a rotational displacement to a transitional displacement, or vice versa. This can be done by attaching one end A of an arm to a fixed position on the rim of a disc. The other end B of the arm is restricted to move along a line l passing the center of the disc. When the disc rotates, end B of the arm will move accordingly on l. Similarly, if end B of the arm is forced to move on l, it will cause the disc to rotate accordingly.

2.2 Sequencer

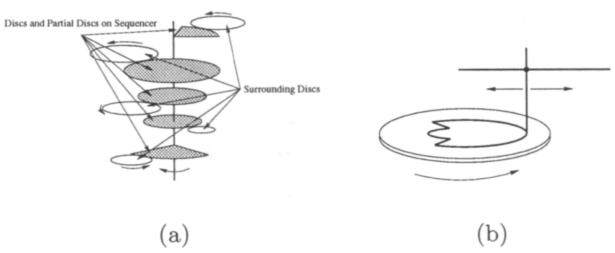


Figure 1: *Sequencer*(a) *and Transitional Movement Sequence Controller*(b).

All the objects in the system are moved, directly or indirectly, by a mechanism called sequencer which is composed of group of discs and partial discs. These discs and partial discs share the same axis and rotate with same angular speed at anytime. Each disc or partial disc is engaged with one or more surrounding discs. Therefore, when the sequencer is rotating, it will make these surrounding discs rotate. Also, the rotation of these surrounding discs will then move other objects in the system, as we know how to convert a rotational displacement to a transitional displacement.

Each cycle of the sequencer finishes one step of computation of the universal TM it simulates. As the rotation of the sequencer determines the starting and finishing of any movement sequence of any other object linked with it, we may use an interval $[\theta_1, \theta_2]$ to denote the time period during which the sequencer rotates from angle θ_1 to angle θ_2 in each cycle rather than using the actual time.

2.3 Transitional movement sequence controller

It will be useful if the sequencer can move an object to a specified position during time period $[\theta_1, \theta_2]$ of each rotational cycle of the sequencer, and move it away from that position otherwise. This can be done by the controller shown in figure 1(b).

There is a circular gap on the surface of the disc which is on the sequencer. The curve of this gap is defined by $r = r(\theta)$ in a polar coordinate with origin at the center of the disc. A vertical bar is restricted in the circular gap. Also, it is restricted to move on a line passing the center of the disc. Radius $r(\theta)$ has the property that $r(\theta) = r_0$ for $\theta \in [\theta_1, \theta_2]$ and $r(\theta) > r_0$ for $\theta \notin [\theta_1, \theta_2]$. Here r_0 is a constant. Therefore, when the controller is rotating in one direction, the bar will move back and forth along the line. More specifically, in time interval $[\theta_1, \theta_2]$, the vertical bar is at the position of $(r_0, 0)$ (in polar coordinate). In other time, it is at some position $(r', 0)$ where $r' > r_0$.

For our convenience, in the following discussion, we may say "the sequencer moves an object to a certain position during a certain time period in each rotational cycle" implicitly assuming that we have constructed a transitional movement sequence controller, as described here, that fulfills this task.

Transitional movement sequence controllers will be used frequently in our frictional system. All the rotational discs, except those on the sequencer, need to be clamped down whenever they are not rotated, directly or indirectly, by the sequencer so that their orientations will be maintained. This can be done by moving a *blocking bar* (a bar with a frictional surface) to a position where it has surface contact with the disc in the clamping period of each cycle.

2.4 Rotational movement sequence controller

A rotational movement sequence controller is a partial disc in the sequencer used to rotate another disc (target disc) in a specified time interval during each cycle. With a transitional movement sequence controller, a blocking bar can be moved to clamp the target disc when the partial disc lose contact with the target disc.

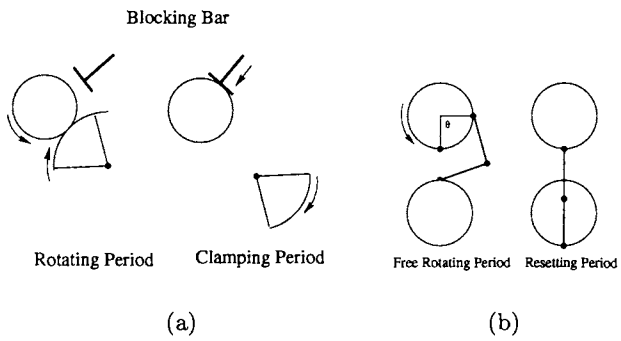


Figure 2: *Rotational Movement Sequence Controller* (a) *and Resettable Rotational Disc* (b).

When the partial disc rotates to such an orientation that its rim contacts the rim of the target disc, this bar is moved away from the target disc so that this disc can be rotated by the partial disc.

Therefore, the disc will stay stationary during the clamping period, and rotate with the partial disc otherwise. The length of the rotating period is determined by the angle of the partial disc. And the total angle the target disc rotates during the rotating period is determined by the length of the arc of the partial disc as well as the radius of the target disc. Therefore, by setting these parameters appropriately, we can let the target disc to rotate a certain angle, say, π, in each cycle.

Again, for our convenience, in the following discussion, we will say "the sequencer rotates a disc by a specified angle during a certain time period" implicitly assuming that we have constructed a rotational movement sequence controller, as described in this subsection, to do it.

2.5 Resettable rotational disc

With a rotational movement sequence controller, the sequencer can rotate a disc by a constant angle during each cycle. Frequently it is necessary to rotate a disc ("target disc") by a certain angle θ (which is recorded by another disc "source disc") and then reset it by rotating it by θ in the other direction, no matter what θ is. This kind of disc is called "resettable rotational disc". As shown in figure 2(b), two discs are linked by a two-segment arm. Each of these two discs has a radius of 1. The distance between the centers of the two discs is d, and the total length of the two-segment arm is d too.

The upper disc is the target disc. The lower disc connects to a rotational movement sequence controller, so that at the time when we want to reset the target disc, the lower disc will rotate by an angle of π. Because the length of the two-segment is exactly the same as the distance between the centers of two discs, the upper disc will be forced to rotate to such an orientation that the point where the arm is attached to is at the lowerest position. When it is the time to release the target disc and allow it to be rotated freely by another disc it engages with, the lower disc rotates by another π so that it restores to its initial orientation.

Using resettable rotational discs, we can construct a mechanical device that will increase the angle of a disc by the angle recorded by another disc. Suppose we have two resettable rotational discs. Disc D_1 is set at θ_1 and D_2 is set at θ_2. We want to increase the angle of D_2 by θ_1 while having the angle of D_1 remain unchanged. A third disc D_3 is needed here and it is set at its initial orientation. The radii of D_1, D_2 and D_3 are the same.

First both D_2 and D_3 are engaged with D_1 and then D_1 is reset to its initial orientation. D_2 (and D_3) will be rotated counterclockwise by an angle of θ_1. This will set D_2 to angle $\theta_1 + \theta_2$ and D_3 to angle θ_1. Then, D_2 is dis-engaged from D_1. The next step is to reset D_3 to its initial orientation. As D_3 is still engaged with D_1, D_1 will be rotated counterclockwise by an angle of θ_1. Therefore, the orientations of D_1 and D_3 will be set just the same as when they were before, while the angle of D_2 is increased by θ_1, which is the angle recorded by D_1.

Observe that we can amplify or reduce θ_1 by a constant factor C before adding it to D_2. This can be done by setting $r_1 = r_3 = Cr_2$, where r_1, r_2 and r_3 are the radii of the three discs respectively.

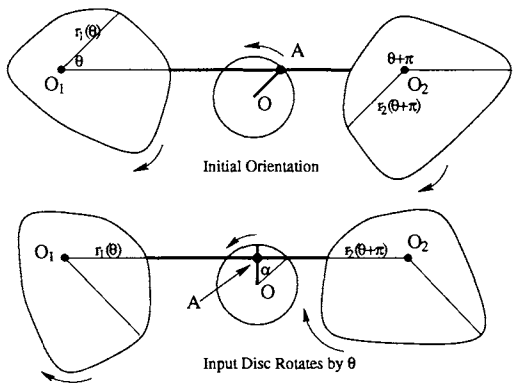


Figure 3: *Non-linear Mapping Device.*

2.6 A non-linear mapping device

Engaging two rotational discs D_1 and D_2 will allow us to perform linear mapping from the angle of rotation of D_1 to the angle of rotation of D_2. By "linear mapping", we mean that if D_1 rotates by an angle of θ, D_2 will rotate by an angle of $C\theta$, here C is the ratio of radius between D_1 and D_2. However, sometime it is necessary to perform non-linear (or even non-monotone) mapping on the rotations of discs. In particular, a device with the following property is needed:

> A non-linear mapping device has two discs, an input disc and an output disc, each of which can rotate any $\theta \in [0, \frac{\pi}{2}]$ from its initial orientation. Given $\theta_1, \theta_2, \cdots, \theta_n$, $\alpha_1, \alpha_2, \cdots, \alpha_n$, $0 \le \theta_1 < \theta_2 < \cdots < \theta_n \le \frac{\pi}{2}$, $0 \le \alpha_i \le \frac{\pi}{2}$, if the rotation of the input disc is θ_i, the rotation of the output disc should be α_i. Further, if $\alpha_1 < \alpha_2 < \cdots < \alpha_n$, the mapping should be monotone. I.e., for any $\theta, \theta_i < \theta < \theta_{i+1}$, if the rotation of the input disc is θ, then the rotation of the output disc, α, should be in the range of (α_i, α_{i+1}).

Non-linear mapping can be implemented by the device described in figure 3. The horizontal bar shown in the figure is restricted to move horizontally only. The joint between the bar and the spoke (a radiating bar from the center of a disc to its circumference) of the central disc D has such a property that it can move along the spoke but will remain stationary with respect to the bar. Therefore, when the bar is sliding back and forth horizontally, it will move the spoke which will in turn make D rotate accordingly.

Initially, D is set at the orientation where the angle between the spoke of D and the x-axis is $\pi/4$. And each end of the horizontal bar touches, but is not attached to, the rim (circumference) of one of the two irregular discs, D_1 and D_2. We want to design the shapes of D_1 and D_2 in such a way that, when D_1 and D_2 rotate clockwise by an angle of θ_i from this initial orientation, the central disc will rotate counterclockwise by an angle of α_i from the initial orientation.

Let $r_1(\theta)$ ($r_2(\theta)$) be the distance between O_1 (O_2, respectively) and the intersection point of the rim of D_1 (D_2, respectively) and the ray with angle θ that starts from O_1 (O_2, respectively). If we let $r_1(\theta) + r_2(\theta + \pi)$ be a constant for any $0 < \theta < \pi/2$, the two ends of the horizontal bar will remain contact with the two irregular discs when these two discs rotate by the same angle from their initial orientation.

Let the radius of D be R, and let the distance between the center O of D and the horizontal bar be $\frac{1}{\sqrt{2}}R$. Therefore, the range of the angle between the spoke and the x-axis is $[\pi/4, 3\pi/4]$. Let h_1 be the horizontal distance between O and the center O_1 of the irregular disc D_1, and let d_1 be the length of the part of the horizontal bar between the left end and the joint point A with the spoke. To achieve the desired property, it is sufficient that the following equality holds:

$$\frac{r_1(\theta_i) + d_1 - h_1}{\frac{1}{\sqrt{2}}R} = \tan(\frac{\pi}{4} - \alpha_i), \text{ for } i = 1, 2, \cdots, n$$

This can be done by choosing $r_1(\theta_i)$ to be $(h_1 - d_1 + \frac{1}{\sqrt{2}}R\tan(\frac{\pi}{4} - \alpha_i))$, for all $i, 1 \le i \le n$.

Also, the initial orientation implies $(r_1(0) + d_1 - h_1)/(1/\sqrt{2}R) = 1$.

For any $\theta \in (\theta_i, \theta_{i+1})$, to ensure that the mapping is monotone in interval $[\theta_i, \theta_{i+1}]$, we specify

$$r(\theta) = \frac{(\theta - \theta_i) * r(\theta_{i+1}) + (\theta_{i+1} - \theta) * r(\theta_i)}{\theta_{i+1} - \theta_i}$$

Therefore, if $\alpha_1 < \alpha_2 < \cdots \alpha_n$, the non-linear mapping will be monotone in interval $[0, \pi/2]$.

The device discussed above is used to simulate a mapping between rotational displacements. However, since the converting device is available to convert transitional displacement to rotational displacement or vice versa, a mapping device can be constructed to simulate non-linear mapping between transitional displacements.

2.7 Selection controller

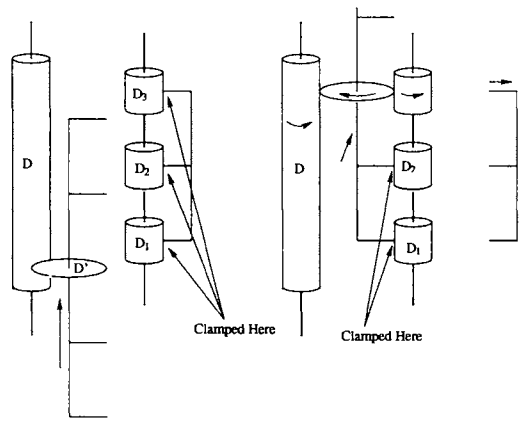


Figure 4: *Selection Controller.*

Another useful device is the so-called "selection controller", which will rotate one of several cylinders depending on the transitional displacement of a bar from its initial position or, equivalently, the rotational displacement of a disc from its initial orientation. Therefore, different position of the bar will incur different movement of the objects in the space.

Suppose there are n cylinders ("choice cylinders") from which one is chosen to be rotated depending on the transitional displacement of a sliding bar. Let l denote the transitional displacement of the sliding bar. We assume that $0 < l < 1$. Further, if $\frac{2i}{2n-1} < l < \frac{2i+1}{2n-1}$ for some i, $0 \leq i \leq n-1$, the selection controller will select the i-th choice cylinder to rotate. This can be implemented by the device in the figure 4.

To the left there is a long cylinder S engaged with a rotational movement sequence controller so that S is rotated by an angle of 2π during a certain period of each cycle and remains static in other time. To the right there is a pile of n identical cylinders, $D_0, D_1, \cdots, D_{n-1}$. These cylinders share the same axis but each can rotate independently. The height of each cylinder is $\frac{1}{2n-1}$ and the distance between two contiguous cylinders is also $\frac{1}{2n-1}$. The radii of $D_0, D_1, \cdots, D_{n-1}$ and D are all r. The distance between D's axis and D_is' axis is $2r + 2R$.

The sliding bar is attached to a so-called "selective engager" shown in the middle of the figure. The selective engager consists of a disc D' with radius R and $2n-2$ blocking bars, each of which has length R. $n-1$ of these blocking bars are above D' and the others are below D'. The vertical distance between any two contiguous bars is $\frac{2}{2n-1}$, and so are the distance between D' and the bar immediately above D' as well as the distance between D' and the bar immediately below D'.

In each cycle, initially the selective engager is placed in such a position that D' is at the same height as the lower face of D_1. Also, the selective engager does not contact either D or any of D_is. After the sliding bar moves vertically by a distance of l, $\frac{2i}{2n-1} < l < \frac{2i+1}{2n-1}$ for some i, the selective engager will move upwards by a distance of l accordingly so that it will be at the same height as D_i.

Then, the selective engager is moved horizontally to the position right between D and D_is. Since the distance between D's axis and D_is' axis is $2r+2R$, D' will contact both D and D_i. Also, any of $D_0, D_1, \cdots, D_{n-1}$ other than D_i will contact one of those blocking bars so that they are forbidden from rotating.

After removing the blocking bars originally clamping the n cylinders, only n_i can rotate freely. Therefore, when D is rotated by an angle of 2π, it will rotate D' and in turn rotates D_i. Since D and D_i have the same radius, D_i will be rotated by exactly 2π.

The functionality of this device is just like the **switch—case** structure in a C program. In our remaining section, we will say "If the transitional (rotational) displacement of a bar (disc, respectively) is in the range of (a, b), the sequencer will do ..." implicitly

assuming that we have constructed a particular selection controller as described here to do this task.

3 Simulation of a Universal TM

Given the basic gadgets, we are ready to construct a frictional mechanical system that simulates a universal TM. Our frictional mechanical system consists of two components, a "control component" simulating the finite state control and a "tape component" simulating the read-write tape. All the objects in the system are linked or engaged with a sequencer, which controls the behavior of those objects. The sequencer is engaged with the power disc which is constantly rotating clockwise. Therefore, the sequencer will be rotating counterclockwise constantly and it will drive all the objects in the system in a specified way as it rotates.

In the control component there is a disc D_{state} whose rotational position represents the current state of the universal TM. Similarly, in the tape component, there are three discs $D_{current}$, D_{left} and D_{right} whose rotational positions together represent the status of the read-write tape. (The status of the tape includes the contents of the tape as well as the current position of the read-write head.)

The control component will take the current symbol and the current state of the universal TM, both of which are represented by rotational positions of certain discs, as input and compute the next state, the next move of the head, and the symbol replacing the current symbol, all of which are also represented by rotational positions. The tape component will then modify the rotational positions of $D_{current}$, D_{left} and D_{right} depending on the next move and the replacing symbol provided by the control component.

The first subsection presents the methods used to encode the current state of the universal TM and the current status of the read-write tape by rotational positions. The next two subsections describe the construction of the control component and the tape component respectively. The last subsection shows how these two components work together to simulate a universal TM and subsequently how the two main theorems of this paper are proved.

3.1 Encoding the configuration

The configuration of a TM includes the current state, the current tape contents and the current head location. To be able to simulate a TM, our frictional mechanical system should be able to record the configuration of the TM by rotational positions of discs in the system.

The status of the read-write tape of the universal TM can be encoded by the rotational positions of three discs $D_{current}$, D_{left} and D_{right}. If $\omega_1\omega_2\cdots\omega_{k_1-1}\check{\omega}_{k_1}\omega_{k_1+1}\cdots\omega_{k_2}$ is the current tape status, the rotation of $D_{current}$, $\beta_{current}$, the rotation of D_{left}, β_{left}, and the rotation of D_{right}, β_{right}, are set as follows:

$$\beta_{current}(\omega_1\omega_2\cdots\check{\omega}_{k_1}\cdots\omega_{k_2}) = \frac{2\omega_{k_1}\pi}{2m-1}$$

$$\beta_{left}(\omega_1\omega_2\cdots\check{\omega}_{k_1}\cdots\omega_{k_2}) = 2\pi\left(\sum_{i=1}^{k_1-1}\frac{\omega_{k_1-i}}{(2m-1)^i}\right)$$

$$\beta_{right}(\omega_1\omega_2\cdots\check{\omega}_{k_1}\cdots\omega_{k_2}) = 2\pi\left(\sum_{i=1}^{k_2-k_1}\frac{\omega_{i+k_1}}{(2m-1)^i}\right)$$

In particular, $\beta_{left} = 0$ if $k = 1$, and $\beta_{right} = 0$ if $k = t$.

Therefore, $\beta_{current}$ encodes the current symbol, and D_{left} (D_{right}) encodes the sub-string to the left (right, respectively) of the current symbol.

It is easy to see that this encoding function has the following properties: (i) $\frac{2i\pi}{2m-1} \le \beta_{current} \le \frac{(2i+1)\pi}{2m-1}$, if $\omega_{k_1} = i$; (ii) $\frac{2i\pi}{2m-1} \le \beta_{left} \le \frac{(2i+1)\pi}{2m-1}$, if $\omega_{k_1-1} = i$; and (iii) $\frac{2i\pi}{2m-1} \le \beta_{right} \le \frac{(2i+1)\pi}{2m-1}$, if $\omega_{k_1+1} = i$.

Therefore, linking a selection controller with each of $D_{current}$, D_{left} and D_{right} will allow the frictional mechanical system to decide the current symbol of the tape and the two symbols next to it which will be used by the control component and the tape component.

During the simulation of the universal TM, if the current state of the universal TM is q_i, D_{state} will be set at a rotational position of $\lambda_i = \frac{i\cdot\pi}{2\sigma}$. Observe that every rotational position of D_{state} that represents a state

is between 0 and $\pi/2$. Again, linking a selection controller with D_{state} will allow the frictional mechanical system to decide the current state of the tape.

3.2 Finite state control component

We let $\delta_1 : Q\times\Sigma \rightarrow Q$, $\delta_2 : Q\times\Sigma \rightarrow \Sigma$ and $\delta_3 : Q\times\Sigma \rightarrow \{L, R\}$ denote the three components of the transition function δ respectively. More specifically, if the current state is q_j and the current symbol is i, the next state is $\delta_1(q_j, i)$, the symbol to replace the current symbol is $\delta_2(q_j, i)$ and the next move of the head is $\delta_3(q_j, i)$.

The control component consists of $3m$ non-linear mapping devices. The first m of these non-linear mapping devices, $M_{1,0}$,$M_{1,1}$, $\cdots$, $M_{1,m-1}$, are used to implement δ_1, the next state function. For each i, $0 \leq i \leq m-1$, $M_{1,i}$ is designed in such a way that, if the input disc of $M_{1,i}$ is rotated by an angle of $\lambda_j = \frac{j\pi}{2\sigma}$ from its initial orientation, the output disc will be rotated by an angle of $\lambda_{j'} = \frac{j'\pi}{2\sigma}$. Here $\delta_1(q_j, i) = q_{j'}$. In other words, if the input disc is rotated by an angle corresponding to the current state of the universal TM, the output disc will be rotated by an angle corresponding to the next state of the TM.

Suppose the current state is q_j and the current symbol is i. As the current symbol can be determined by applying a selection controller to $D_{current}$, the sequencer will be able to choose $M_{1,i}$ from $M_{1,0}, M_{1,1}, \cdots, M_{1,m-1}$ and rotate the input disc of $M_{1,i}$ by an angle of λ_j, which is recorded by D_{state}. Then, D_{state} will be set at the angle recorded by the output disc of $M_{1,i}$ and the non-linear mapping device $M_{1,i}$ will be reset to its initial position.

In the remaining $2m$ non-linear mapping devices, $M_{2,0}$, $M_{2,1}$, $\cdots$, $M_{2,m-1}$ are used to implement δ_2; and $M_{3,0}$, $M_{3,1}$, $\cdots$, $M_{3,m-1}$ are used to implement δ_3. For each $i, 0 \leq i \leq m-1$, $M_{2,i}$ has the property that, if the input disc of $M_{2,i}$ is rotated by an angle of $\lambda_j = \frac{j\pi}{2\sigma}$ from its initial orientation, the output disc will be rotated by an angle of $\frac{2\delta_2(q_j, i)\pi}{2m-1}$ from its initial orientation. Similarly, for each i, $0 \leq i \leq m-1$, $M_{3,i}$ has the property that, if the input disc of $M_{3,i}$ is rotated by an angle of λ_j, the output disc will be rotated by an angle of $\pi/4$ if $\delta_3(q_j, i) = L$ or $\pi/2$ if $\delta_3(q_j, i) = R$.

Therefore, the next move and the replacing symbol will be represented by rotational positions of certain discs. These rotational positions will be used by the tape component to update the rotational positions of $D_{current}$, D_{left} and D_{right} according to δ.

3.3 Read-write tape component

As mentioned above, the control component will generate two rotational positions, one representing the next move and the other representing the replacing symbol. Let $\omega_1\omega_2\cdots\check{\omega_{k_1}}\cdots\omega_{k_2}$ be the current tape status and ω'_{k_1} be the replacing symbol.

For our convenience, we use $\beta_{current,t}$ to denote the angle of $D_{current}$ at the beginning of the t-th rotational cycle (of the sequencer). $\beta_{left,t}$, $\beta_{right,t}$ and $\beta_{state,t}$ are defined likewise. First suppose the next move of the head is to go rightward. There are three cases:

1 $k_1 = k_2$, i.e., the head is at the right endmark:
The status of the tape will be changed to $\omega_1\omega_2\cdots\omega'_{k_1}\omega_{\check{k_2}+1}$. Here $\omega_{k_2+1} = 2$ (indicating the right endmark) as the head will pad a right endmark to the right. Therefore, we have the following recursive equations:

$$\beta_{current,t+1} = 2\frac{2\pi}{2m-1} \tag{1}$$

$$\beta_{right,t+1} = 0 \tag{2}$$

$$\beta_{left,t+1} = \frac{\beta_{left,t}}{2m-1} + \omega'_{k_1}\frac{2\pi}{2m-1} \tag{3}$$

2 $k_1 = 1$, i.e., the head is at the left endmark:
As the universal TM will never overwrite the left endmark, ω_1 will still be 1 (indicating the left endmark) and the status of the tape will be changed to $\omega_1\check{\omega_2}\omega_3\cdots\omega_t$. Therefore, we have

$$\beta_{current,t+1} = \frac{2\pi\omega_2}{2m-1} \tag{4}$$

$$\beta_{right,t+1} = (\beta_{right,t} - \omega_2\frac{2\pi}{2m-1}) * (2m-1) \tag{5}$$

$$\beta_{left,t+1} = \frac{2\pi}{2m-1} \tag{6}$$

3 $1 < k_1 < k_2$:
The status of the tape will be changed to $\omega_1\omega_2\cdots\omega'_{k_1}\omega_{k_1+1}\omega_{k_1+2}\cdots\omega_{k_2}$. Thus, correspondingly, the recursive equations are

$$\beta_{current,t+1} = \frac{2\pi\omega_{k_1+1}}{2m-1} \quad (7)$$

$$\beta_{right,t+1} = (\beta_{right,t} - \omega_{k_1+1}\frac{2\pi}{2m-1}) * (2m-1) \quad (8)$$

$$\beta_{left,t+1} = \beta_{left,t}/(2m-1) + \omega'_{k_1}\frac{2\pi}{2m-1} \quad (9)$$

Similarly, if the next move is to go leftward, there are two cases (the read-right head can not go leftward if it is at the left endmark):

1 If $k_1 = k_2$, i.e., the head is at the right endmark:

$$\beta_{current,t+1} = \frac{2\pi\omega_{k_1-1}}{2m-1} \quad (10)$$

$$\beta_{right,t+1} = \begin{cases} \frac{2\cdot 2\pi}{2m-1} & \text{if } \omega'_{k_1} = 2 \\ \frac{\omega'_{k_1}\cdot 2\pi}{2m-1} + \frac{2\cdot 2\pi}{(2m-1)^2} & \text{if } \omega'_{k_1} \neq 2 \end{cases} \quad (11)$$

$$\beta_{left,t+1} = (\beta_{left,t} - \omega_{k_1-1}\frac{2\pi}{2m-1})(2m-1) \quad (12)$$

2 If $1 < k_1 < k_2$:

$$\beta_{current,t+1} = \frac{2\pi\omega_{k_1-1}}{2m-1} \quad (13)$$

$$\beta_{right,t+1} = \beta_{right,t}/(2m-1) + \omega'_{k_1}\frac{2\pi}{2m-1} \quad (14)$$

$$\beta_{left,t+1} = (\beta_{left,t} - \omega_{k_1-1}\frac{2\pi}{2m-1})(2m-1) \quad (15)$$

With selection controllers, the system can change the rotational positions of $D_{current}$, D_{left} and D_{right} from $\beta_{current,t}$, $\beta_{left,t}$ and $\beta_{right,t}$ to $\beta_{current,t+1}$, $\beta_{left,t+1}$ and $\beta_{right,t+1}$ respectively according to the recursive equations described above.

3.4 Reduction from A_{TM} to the frictional mover's problem

With mechanical components that can simulate the transition function and the tape of a universal TM, the frictional mechanical system that simulates the universal TM is almost immediate. The movements of the objects in this system are controlled by the rotations of the sequencer. In each cycle of rotation of the sequencer, the frictional mechanical system will finish the simulation of one step of the universal TM.

At the beginning of each cycle, the symbol at the current head location is decided depending on the rotational position of $D_{current}$. According to the current symbol as well as the current state recorded by D_{state}, the control component will make a sequence of moves and then decide (i) the next state, (ii) the symbol replacing the current symbol and (iii) the next move of the tape head. The tape component will then make a sequence of moves to change the rotational positions of $D_{current}$, D_{left} and D_{right} according to the symbol replacing the current symbol and the next move. This finishes the simulation of one step of the computation of the universal TM.

Initially, D_{state} is set at the orientation encoding the start state of the TM. The orientations of $D_{current}$, D_{left} and D_{right} also correspond to the initial status of the read-write tape. The simulation for the computation of the universal TM terminates when the rotational position of D_{state} is found to be corresponding to the accepting or rejecting state of the universal TM (we call these rotational positions "terminating orientations") at the beginning of a rotational cycle of the sequencer. [5]

Herewith we have proved the following theorem:

Theorem 1 *For any universal TM M, a frictional mechanical system can be constructed which has the property that, for any input string ω of M, the objects to the system can be set in a corresponding initial configuration so that a specified final configuration can be reached if and only if M accepts ω.*

As M is a universal TM, our frictional mechanical system can be used for general-purpose computing. Thus, it has the computational power of any conventional electronic computer.

To prove that the frictional mover's problem is undecidable, we will show that A_{TM} can be reduced to the

[5] Observe that D_{state} might be at a terminating orientation momentarily as it rotates around. However, if at the beginning of a rotational cycle D_{state} stays at a terminating orientation, this means that the universal TM has reached an accepting or rejecting state.

frictional mover's problem in log-space. Therefore, it is necessary that the frictional mechanical system, in particular, all the objects in the system and their positions, can be described by rational coefficients. We can easily specify all the discs (including partial discs), bars, and cylinders in the system by rational coefficients. The irregular discs used in the non-linear mapping devices, however, have to be specified by some irrational coefficients as indicated in Section 2.6. But we will prove in the next section the following lemma:

Lemma 2 *The irrational coefficients of irregular discs can be replaced by rational numbers without inducing any error in simulating the universal TM.*

With this lemma, we have:

Theorem 3 *The frictional mover's problem is undecidable.*

4 A Frictional Mechanical System with Error

As mentioned in the introduction of this paper, our mechanical frictional system can simulate a universal TM without any error only if the system can be constructed and work exactly as it is specified. In the presence of errors, it is not possible to record the current configuration of the universal TM by the rotational positions of D_{state}, $D_{current}$, D_{left} and D_{right} exactly as it is. Therefore, the computational power of the frictional mechanical system is restricted as the errors in the rotational positions of these discs may be accumulated significant enough to induce incorrect result in the simulation of the universal TM.

We use ϵ to denote the upper bound for the error in an angle which occurs in a single mechanical operation. ϵ is determined by the degree of accuracy in constructing mechanical devices as well as measuring rotational (and transitional) displacements.

Let $\beta'_{state,t}$ denote the correct angle of D_{state} at the beginning of the t-th rotational cycle if the frictional mechanical system has no error. Further, let $\Delta_{state,t} = \mid \beta'_{state,t} - \beta_{state,t} \mid$, i.e., $\Delta_{state,t}$ is the error in $\beta_{state,t}$. $\beta'_{current,t}$, $\Delta_{current,t}$, $\beta'_{left,t}$, $\Delta_{left,t}$, $\beta'_{right,t}$ and $\Delta_{right,t}$ are defined accordingly.

There are only m valid values for $\beta'_{current,t+1}$, (i.e., $0, \frac{2\pi}{2m-1}, \cdots, \frac{2(m-1)\pi}{2m-1}$) and the difference between any two valid values is at least $\frac{2\pi}{2m-1}$. Therefore, as long as $\Delta_{current,t+1} < \frac{\pi}{2(2m-1)}$, a selection controller can be used to determine $\beta'_{current,t+1}$ from $\beta_{current,t+1}$. This means that $\beta_{current,t+1}$ can be corrected at the beginning of each cycle.

The same analysis applies to $\beta_{state,t+1}$. Therefore, only the errors in $\beta_{left,t}$ and $\beta_{right,t}$ may be accumulated to the next cycle. As at each step, $\beta_{left,t}$ and $\beta_{right,t}$ are only used to distinguish ω_{k_1-1} and ω_{k_1+1}, errors in $\beta_{left,t}$ and $\beta_{right,t}$ will not cause any incorrectness if they are less than $\frac{\pi}{2(2m-1)}$.

We will have a digression here to prove Lemma 1 before we further discuss the errors in β_{left} and β_{right}. In the previous section, we claimed (without proving) that we can change the irrational coefficients of the irregular discs (on the non-linear mapping devices) to rational numbers without affecting the result of the simulation of the universal TM. This is because, if we construct the irregular discs using rational coefficients very close to the original irrational coefficients, each non-linear mapping device might not be 100% accurate in the sense that, if the input angle is corresponding to the current state, the output angle will not be same as but very close to the angle corresponding to the next state. If the difference is small enough, a selection controller can be used to determine the next state depending on β_{state}. This proves Lemma 1.

Returning to the error model, if the errors in β_{left} and β_{right} can be controlled in a certain range, the simulation of the universal TM will not produce any incorrect result. One way of avoiding the accumulation of the errors in β_{left} and β_{right} is to construct a frictional mechanical system that simulates a TM M' equivalent to M but whose head will sweep the tape periodically. Due to the length of the paper, we only give the theorem here. Interested readers please refer to the full version of this paper[8].

Theorem 4 *For any universal TM M, a frictional mechanical system with error can be constructed to*

simulate M. It has the property that, for any space bound S, if the single-operation error of the system, ϵ, is bounded by $O(2^{-cS})$ for some constant c, then given any input string ω which M decides in space bound S, the frictional mechanical system will reach a final configuration from an initial configuration encoding ω if and only if M accepts ω. Further, the frictional mechanical system will take at most $(2T+1)T$ cycles to finish the computation if M decides ω in T steps.

With the following assumption, we improve the above theorem in the sense that periodic sweeps of the tape can be avoided so that the frictional mechanical system will take T cycles to finish the computation if M decides ω in T steps. Again, we leave the proof in the full paper[8].

Assumption 5 *For any operation that will multiply an angle by a constant factor C, if before the operation the error in the angle is Δ, after the operation, the error will become $C\Delta + k\epsilon$ for some constant k.*

This implies that $\frac{1}{\epsilon}$ will have to increase exponentially as S increases to maintain the correctness of the frictional mechanical system. If in the frictional mechanical system S discs instead of three discs are used to encode the contents of the tape, each representing one cell, a frictional mechanical system in ϵ-error model can simulate the universal TM M with space bound S where $\epsilon = O(1)$. This is because now each disc has only constant number of valid rotational positions and selection controllers can be used to distinguish the correct value represented by the disc. This is better than the previous result in the sense that $\frac{1}{\epsilon}$ does not increase as S increases. However, the frictional mechanical system is now dependent on S, as it needs S discs to encode the tape. In other words, this model of frictional mechanical systems is a digital computer, just like the Analytical Engine. If we want to increase the computational power of the frictional mechanical system by using larger space bound S, we will have to add more discs used to represent the tape. For detailed description of this model please see the full paper[8].

Acknowledgment

This research was supported by grants NSF CCR-9725021 and CCR-96-33567 and Army Research Office contract DAAH-04-96-1-0448.

References

[1] J. Canny. Some algebraic and geometric computations in PSPACE. In Richard Cole, editor, *Proceedings of the 20th Annual ACM Symposium on the Theory of Computing*, pages 460–467, May 1988.

[2] P.C. Chen and Y.K. Hwang. Motion planning for a robot and a movable object amidst polygonal obstacles. In *Proceedings: 1991 IEEE International Conference on Robotics And Automation*, volume 1, pages 444–449, April 1991.

[3] B. Dacre-Wright, J.-P. Laumond, and R. Alami. Motion planning for a robot and a movable object amidst polygonal obstacles. In *Proceedings: 1992 IEEE International Conference on Robotics And Automation*, volume 3, pages 2474–2480, May 1992.

[4] J. E. Hopcroft, D. A. Joseph, and S. H. Whitesides. Movement problems for 2-dimensional linkages. *SIAM Journal on Computing*, 13(3):610–629, 1984.

[5] J. E. Hopcroft, J. T. Schwartz, and M. Sharir. On the complexity of motion planning for multiple independent objects; PSPACE-hardness of the "warehouseman's problem". *International Journal of Robotics Research*, 3(4):76–88, 1984.

[6] J. Reif. Complexity of the mover's problem and generalizations. In *Proceedings of the 20th IEEE Symposium on Foundations of Computer Science*, pages 421–427, October 1979. Also appeared in Complexity of the General Mover's Problem, Chapter 11 in *Planning, Geometry and Complexity of Robot Motion*, pages 267–281, J. Hopcroft, J. Schwartz, and M. Sharir, editors, Ablex Pub., Norwood, NJ, 1987.

[7] J. Reif and M. Sharir. Motion planning in the presence of moving obstacles. In *26th Annual Symposium on Foundations of Computer Science*, pages 144–154, October 1985.

[8] J. Reif and Z. Sun. The computational power of frictional mechanical systems. Downloadable from "http://www.cs.duke.edu/ ~sunz/frictional.ps".

[9] J. Sellen. Lower bounds for geometrical and physical problems. *SIAM Journal on Computing*, 25(6):1231–1253, December 1996.

[10] G. Wilfong. Motion planning in the presence of movable obstacles. In *Proceedings of the Fourth Annual Symposium on Computational Geometry*, pages 279–288, 1988.

Issues in Nonprehensile Manipulation

Kevin M. Lynch, *Northwestern University, Evanston, IL, USA*

This paper outlines geometric and algorithmic issues common to various types of nonprehensile manipulation and gives some results for planar dynamic manipulation.

1 Overview

Nonprehensile manipulation is manipulation without a form- or force-closure grasp. Examples include pushing, throwing, juggling, tapping, batting, and rolling (Mason [24]; Higuchi [11]; Bühler and Koditschek [6]; Erdmann [8]; Huang *et al.* [12]; Zumel and Erdmann [40]; Aiyama *et al.* [1]; Trinkle and Zeng [37]). In each of these examples, the robot takes advantage of the natural task dynamics to help control the motion of the part. Nonprehensile manipulation occupies the majority of the manipulation spectrum, comprising everything between situations where the robot exerts complete control to situations where the natural dynamics exert complete control. During a baseball throw, the ball is at first held firmly in the hand, then is allowed to roll off the fingers, and finally follows a free-flight trajectory determined by gravity and air resistance. The nonprehensile manipulation problem is to arrange the rolling motion on the fingers such that the release state will allow the ball to reach the goal state.

Nonprehensile manipulation allows a simple, low degree-of-freedom robot to control more part freedoms through relative motion (slipping, rolling, and free flight) and to cause motion outside of its kinematic workspace. One cost of using nonprehensile manipulation, relative to pick-and-place manipulation, is that motion planning and control are harder. A nonprehensile manipulation planner must reason about dynamics. Pick-and-place is usually concerned only with kinematics, and perhaps static friction for grasping (Lozano-Pérez *et al.* [16]). Also, it is not obvious what tasks are achievable by nonprehensile manipulation. With pick-and-place, if the initial and goal configurations of the part are within the robot's kinematic workspace and there is a free path connecting these configurations, then the goal configuration is achievable.

Interesting geometric and algorithmic problems in nonprehensile manipulation include

- generalizing the notion of a robot's kinematic workspace to consider the dynamically accessible workspace of the manipulated object;
- developing controllability tests based on the geometry, friction coefficient, and mass distribution of the object (and kinematic and dynamic descriptions of the manipulator when appropriate);
- characterizing classes of manipulator-object systems that are controllable;
- using properties of the system dynamics to derive efficient and complete motion planners; and
- determining feedback laws that stabilize the motion plans.

We have studied many of these issues in quasistatic pushing in (Lynch and Mason [20]; Lynch [17]). This paper summarizes our recent work on controllability and motion planning for planar dynamic nonprehensile manipulation, connects this work to control of an underactuated manipulator, and introduces the notion of dynamic workspaces in nonprehensile manipulation.

2 Example Systems

To motivate our discussion of controllability and motion planning, we will use the example of dynamic nonprehensile manipulation of a planar rigid body $\mathcal{B}$ in zero gravity. The body $\mathcal{B}$ moves like a puck on an air table. Depending on the control inputs used, $\mathcal{B}$ represents

1. a hovercraft or planar spacecraft with unilateral thrusters;
2. an object being pushed with point contact along its boundary;
3. the third link of a 3R planar underactuated robot with a passive third joint; or
4. an object being manipulated by a one joint robot.

The only constraints on the hovercraft thrusters are that they are fixed in the body frame and provide unilateral thrust. In the case of dynamic pushing, we assume a two-degree-of-freedom point robot P which can push $\mathcal{B}$ at any point on its closed, piecewise smooth perimeter Γ. If we think of Γ as a pin-hole with P captured inside of it, we get the underactuated 3R robot, where the pin-hole is the third joint. In the last case, a one-degree-of-freedom revolute robot pushes $\mathcal{B}$ with point contact. See Figure 1.

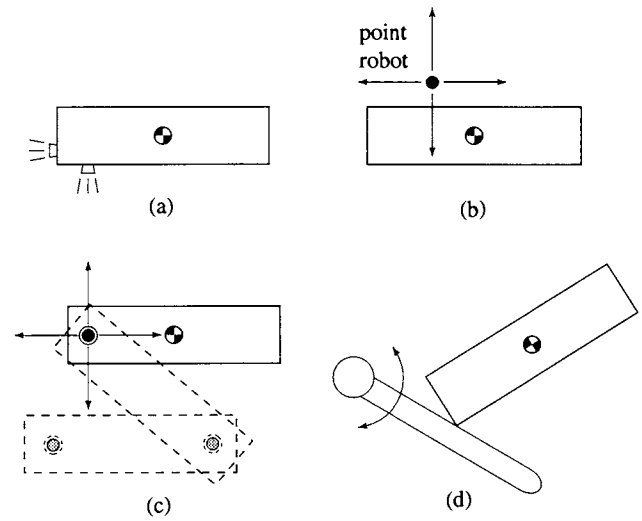


Figure 1: *Control systems with the same dynamics but different control inputs. (a) A planar hovercraft with unilateral thrusters. (b) Dynamic pushing with a two-degree-of-freedom point robot. (c) The third link of a 3R planar robot with a passive third joint. (d) An object being manipulated by a single joint robot.*

3 Controllability

The configuration space of $\mathcal{B}$ is $\mathcal{C} = SE(2)$, the set of planar positions and orientations, and its state space is the tangent bundle $T\mathcal{C}$. A coordinate frame Σ_b is attached to the center of mass of $\mathcal{B}$, and its configuration in an inertial frame Σ_w is given by $\mathbf{q} = (x_w, y_w, \phi_w)^T$. The state of $\mathcal{B}$ is written $(\mathbf{q}, \dot{\mathbf{q}}) \in T\mathcal{C}$. We define the zero velocity section Z as the three-dimensional space of zero velocity states $(\mathbf{q}, \mathbf{0})$.

The manipulation system can be written as the nonlinear control system

$$(\dot{\mathbf{q}}, \ddot{\mathbf{q}}) = X_0(\mathbf{q}, \dot{\mathbf{q}}) + \sum_{i=1}^{n} u_i X_i(\mathbf{q}, \dot{\mathbf{q}}), \qquad (1)$$

where $X_0(\mathbf{q}, \dot{\mathbf{q}}) = (\dot{x}_w, \dot{y}_w, \dot{\phi}_w, 0, 0, 0)^T$ is the drift vector field, $\mathbf{u} = (u_1, \ldots, u_n)^T$ is the control input, and $X_i(\mathbf{q}, \dot{\mathbf{q}})$ is a control vector field. For systems 1–3, we choose the unit mass to be the mass of $\mathcal{B}$ and the unit distance to be the radius of gyration of inertia of $\mathcal{B}$, and control forces are fixed in the body frame. For these systems, $X_i(\mathbf{q}, \dot{\mathbf{q}}) = (0, 0, 0, f_{xi} \cos\phi_w - f_{yi} \sin\phi_w, f_{xi} \sin\phi_w + f_{yi} \cos\phi_w, \tau_i)^T$, where (f_{xi}, f_{yi}) is a unit force expressed in the body frame Σ_b ($f_{xi}^2 + f_{yi}^2 = 1$) and τ_i is the torque about the center of mass. We write $\mathbf{f}_i = (f_{xi}, f_{yi}, \tau_i)^T$ and $\mathcal{F} = \bigcup_{i=1\ldots n} \mathbf{f}_i$.

We obtain each of the four systems (hovercraft, dynamic pushing, underactuated robot, and one joint robot) by placing different constraints on the control vector $\mathbf{u}$ and the forces $\mathcal{F}$.

Hovercraft. The thruster forces must be unilateral. We will also place the constraints that only one thruster can be active at a time with unit thrust magnitude. The control set is $U_1 = \{\mathbf{0}, (1, 0, \ldots, 0)^T, (0, 1, \ldots, 0)^T, \ldots, (0, 0, \ldots, 1)^T\}$. The hovercraft has n thrusters providing forces $\mathcal{F}$ fixed in the body frame Σ_b.

Dynamic pushing. This case is the same as the hovercraft case, with the added restriction that each contact force $\mathbf{f}_i \in \mathcal{F}$ must be realizable from point contact somewhere on the closed, piecewise smooth perimeter Γ of $\mathcal{B}$.

Underactuated robot. We choose $n = 2$, with $f_1 = (1, 0, 0)^T$ and $f_2 = (0, 1, \tau)^T$—the forces act along the x-axis of Σ_b through the center of mass of $\mathcal{B}$, and along the y-axis of Σ_b with nonzero torque, respectively. The control set is $U_3 = [-1, 1] \times [-1, 1]$. These controls are realizable at all joint angles away from the singularity $\sin\theta_2 = 0$, where θ_2 is the angle of the second link with respect to the first link.

One joint robot. We set $n = 1$ corresponding to the single input, the angular acceleration $\ddot{\theta}$ of the one joint robot. The force $\mathbf{f}_1$ is normal to the robot's surface at the point contact with the object, and therefore is not fixed in the body frame Σ_b as in the previous three systems. By an input transformation we can treat the control input as $\mathbf{u} = u \in U_4 = \{0, 1\}$.

A *feasible trajectory* for the body $\mathcal{B}$ is a solution of (1) for a control function $\mathbf{u} : [0, T] \to U_i$, $i = 1, \ldots, 4$ for the systems 1–4 above.

Modifying notation from Nijmeijer and van der Schaft [28], we define $R^V(\mathbf{q}_0, \dot{\mathbf{q}}_0, T)$ to be the reachable set from $(\mathbf{q}_0, \dot{\mathbf{q}}_0)$ at time $T > 0$ by feasible trajectories remaining in the neighborhood V of $(\mathbf{q}_0, \dot{\mathbf{q}}_0)$ at times $t \in [0, T]$. Define $R^V(\mathbf{q}_0, \dot{\mathbf{q}}_0, \leq T) = \bigcup_{0 \leq t \leq T} R^V(\mathbf{q}_0, \dot{\mathbf{q}}_0, t)$. Then (1) (or simply $\mathcal{B}$) is *small-time accessible from* $(\mathbf{q}_0, \dot{\mathbf{q}}_0)$ if $R^V(\mathbf{q}_0, \dot{\mathbf{q}}_0, \leq T)$ has nonempty interior on $T\mathcal{C}$ for any neighborhood V of $(\mathbf{q}_0, \dot{\mathbf{q}}_0)$ and all $T > 0$. $\mathcal{B}$ is *accessible from* $(\mathbf{q}_0, \dot{\mathbf{q}}_0)$ if there exists a finite time T such that $R^{T\mathcal{C}}(\mathbf{q}_0, \dot{\mathbf{q}}_0, \leq T)$ has nonempty interior. $\mathcal{B}$ is *small-time locally controllable from* $(\mathbf{q}_0, \dot{\mathbf{q}}_0)$ if $R^V(\mathbf{q}_0, \dot{\mathbf{q}}_0, \leq T)$ contains a neighborhood of $(\mathbf{q}_0, \dot{\mathbf{q}}_0)$ for any neighborhood V and all $T > 0$. We define $\mathcal{B}$ to be simply *locally controllable from* $(\mathbf{q}_0, \dot{\mathbf{q}}_0)$ if there exists a $T > 0$ such that $R^V(\mathbf{q}_0, \dot{\mathbf{q}}_0, \leq T)$ contains a neighborhood of $(\mathbf{q}_0, \dot{\mathbf{q}}_0)$ for any neighborhood V. $\mathcal{B}$ is *controllable from* $(\mathbf{q}_0, \dot{\mathbf{q}}_0)$ if, for any $(\mathbf{q}_1, \dot{\mathbf{q}}_1) \in T\mathcal{C}$, there exists a finite time T such that $(\mathbf{q}_1, \dot{\mathbf{q}}_1) \in R^{T\mathcal{C}}(\mathbf{q}_0, \dot{\mathbf{q}}_0, T)$. The phrase "from $(\mathbf{q}_0, \dot{\mathbf{q}}_0)$" can be eliminated from each of these definitions if the condition applies at all $(\mathbf{q}_0, \dot{\mathbf{q}}_0)$.

A second-order system can be locally controllable only on the zero velocity section Z. (For any initial nonzero velocity state $(\mathbf{q}, \dot{\mathbf{q}})$ and any $\epsilon > 0$, there is an open subset V containing $(\mathbf{q}, \dot{\mathbf{q}})$ and $(\mathbf{q} - \epsilon\dot{\mathbf{q}}, \dot{\mathbf{q}})$ such that the system cannot attain $(\mathbf{q} - \epsilon\dot{\mathbf{q}}, \dot{\mathbf{q}})$ without leaving V.)

3.1 Hovercraft

Partial results for the hovercraft case have been reported previously by Manikonda and Krishnaprasad [22] and Lewis and Murray [15]. Manikonda and Krishnaprasad [22] observed that the Hamiltonian dynamics of the hovercraft on the cotangent bundle $T^*\mathcal{C}$ are invariant to the Lie group $\mathcal{C}$, suggesting the study of the reduced dynamics on the three-dimensional quotient manifold $T^*\mathcal{C}/\mathcal{C}$. They showed that the reduced dynamics of a hovercraft with a single bilateral thruster (or two opposing unilateral thrusters) are controllable on $T^*\mathcal{C}/\mathcal{C}$—the hovercraft is controllable on its velocity space $T\mathcal{C}/\mathcal{C}$.

Lewis and Murray [15] studied the set of reachable configurations for mechanical control systems starting at rest. For an initial configuration $\mathbf{q}$ at zero velocity and neighborhood $V_\mathcal{C}$ of $\mathbf{q}$ on $\mathcal{C}$, they define $R_\mathcal{C}^{V_\mathcal{C}}(\mathbf{q}, T)$ to be the set of reachable configurations (with any velocity) at time T by trajectories remaining in the configuration neighborhood $V_\mathcal{C}$. They call a system *small-time locally configuration controllable* if $R_\mathcal{C}^{V_\mathcal{C}}(\mathbf{q}, \leq T)$ contains a neighborhood of $\mathbf{q}$ on $\mathcal{C}$ for any neighborhood $V_\mathcal{C}$ and all $T > 0$. (This is a weaker condition than small-time local controllability, which requires the locally reachable set to be a neighborhood of $(\mathbf{q}, \mathbf{0})$ on the full state space $T\mathcal{C}$, not just the configuration space.) They showed that a hovercraft with two bilateral thrusters (or four unilateral thrusters) is small-time locally configuration controllable.

Here we give stronger results for the unilateral thruster case (Lynch [21]). For a hovercraft with one to three unilateral thrusters, we get a new property with each additional thruster: one thruster yields small-time accessibility on the hovercraft's state space $T\mathcal{C}$; two thrusters yield global controllability on $T\mathcal{C}$; and three

thrusters yield small-time local controllability on the zero velocity section Z.

Proposition 1 *The hovercraft is small-time accessible with a single thruster ($n = 1$) if and only if the line of action of the thruster does not pass through the center of mass.*

Proof: To test for small-time accessibility we examine the Lie algebra of the system vector fields X_0 and X_1. Without loss of generality, assume the thruster is aligned with the y-axis of Σ_b ($\mathbf{f}_1 = (0, 1, \tau)^T$), yielding $X_1 = (0, 0, 0, -\sin\phi_w, \cos\phi_w, \tau)^T$. We define the Lie bracket vector fields $X_2 = [X_0, X_1]$, $X_3 = [X_1, [X_0, X_1]]$, $X_4 = [X_1, [X_0, [X_0, X_1]]]$, $X_5 = [X_1, [X_1, [X_0, [X_0, X_1]]]]$, $X_6 = [X_0, [X_1, [X_1, [X_0, [X_0, X_1]]]]]$. We find that

$$\det(X_1\ X_2\ X_3\ X_4\ X_5\ X_6) = -16\tau^8,$$

indicating that these six vector fields span the tangent space $T_{(\mathbf{q},\dot{\mathbf{q}})}\mathcal{TC}$ at any state $(\mathbf{q}, \dot{\mathbf{q}})$, provided $\tau \neq 0$ (the line of action of the thruster must not pass through the center of mass). Therefore the system satisfies the Lie Algebra Rank Condition (LARC) and the system is small-time accessible (Hermann and Krener [10]). □

A single thruster is never sufficient for controllability, as the hovercraft's angular velocity is not controllable. It turns out that two thrusters are sufficient.

Proposition 2 *The hovercraft is controllable with two thrusters ($n = 2$) if and only if the two thrusters provide torque of opposite signs ($\tau_1 > 0, \tau_2 < 0$).*

The proof of this is given in (Lynch [21]). The approach is to find maneuvers that render the hovercraft controllable on its velocity space (the quotient space $\mathcal{TC}/\mathcal{C}$) and controllable on the zero velocity section Z. These properties are sufficient to show global controllability.

Proposition 3 *The hovercraft is small-time locally controllable on the zero velocity section Z with three thrusters ($n = 3$) if their lines of action intersect at a single point (which is not the center of mass) and the linear components of $\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3$ positively span the plane.*

Proof: Consider the system (Lewis and Murray [15])

$$(\dot{\mathbf{q}}, \ddot{\mathbf{q}}) = X_0(\mathbf{q}, \dot{\mathbf{q}}) + u_1 X_1(\mathbf{q}, \dot{\mathbf{q}}) + u_2 X_2(\mathbf{q}, \dot{\mathbf{q}}), \quad |u_i| \leq 1. \tag{2}$$

We will also need the bracket terms $X_3 = [X_0, X_1]$, $X_4 = [X_0, X_2]$, $X_5 = [X_1, [X_0, X_2]]$, $X_6 = [X_0, [X_1, [X_0, X_2]]]$.

Now we give some definitions necessary to apply a version of Sussmann's [36] sufficient condition for small-time local controllability to a system such as (2). For a bracket term B, we define $\delta_i(B)$ as the number of times X_i appears in B, and the degree of B is $\sum_{i=0}^{n} \delta_i(B)$. B is called a "bad" bracket if $\delta_0(B)$ is odd and $\delta_i(B)$ is even for all $i \in \{1, \ldots, n\}$, and B is a "good" bracket otherwise. A "bad" bracket B is "neutralized" at a state $\mathbf{p}$ if B, evaluated at $\mathbf{p}$, is the linear combination of "good" brackets of lower degree evaluated at $\mathbf{p}$. Sussmann proved that if the system satisfies the LARC at $\mathbf{p}$ and all "bad" brackets evaluated at $\mathbf{p}$ are neutralized, then the system is small-time locally controllable at $\mathbf{p}$.

Assume $X_1 = (0, 0, 0, \cos\phi_w, \sin\phi_w, 0)^T$ and $X_2 = (0, 0, 0, -\sin\phi_w, \cos\phi_w, \tau)^T$. The force $\mathbf{f}_1$ acts through the center of mass along the x-axis of Σ_b and $\mathbf{f}_2$ acts along the y-axis with torque τ about the center of mass. The force lines intersect at a point C not at the center of mass. Calculating the brackets above, we find that $\det(X_1\ X_2\ X_3\ X_4\ X_5\ X_6) = \tau^4$; the LARC is satisfied provided $\tau \neq 0$. Because we only use brackets up to degree four, the only "bad" brackets to be neutralized are the drift field (which vanishes at $\dot{\mathbf{q}} = \mathbf{0}$) and the "bad" brackets of degree three $[X_1, [X_0, X_1]]$ and $[X_2, [X_0, X_2]]$. We have

$$[X_1, [X_0, X_1]] = (0, 0, 0, 0, 0, 0)^T$$

$$[X_2, [X_0, X_2]] = (0, 0, 0, -2\tau\cos\phi_w, -2\tau\sin\phi_w, 0)^T$$

which are clearly neutralized (the latter being a multiple of X_1), and the system is small-time locally controllable. The two forces $\mathbf{f}_1$ and $\mathbf{f}_2$ span a body-fixed force/torque plane $\mathcal{P}$ which is neither the $\tau = 0$ plane nor orthogonal to the $\tau = 0$ plane, and the controls $|u_i| \leq 1$ define a compact, convex subset of $\mathcal{P}$ containing the origin in the interior (relative to $\mathcal{P}$). By

Sussmann's [36] Proposition 2.3, small-time local controllability for this system implies small-time local controllability for the bang-bang system with the extremal controls $u_1, u_2 \in \{-1, 1\}$. Scaling, small-time local controllability holds for any compact, convex set of control forces that contains a neighborhood of the origin in $\mathcal{P}$, and Sussmann's proposition indicates that the extremal forces alone are sufficient. Therefore, any set of control forces that *positively* spans the plane $\mathcal{P}$ also yields small-time local controllability. Any three unilateral forces which intersect at C and positively span the (x, y) plane also positively span the force plane $\mathcal{P}$. □

3.1.1 Shape derivatives in grasping and motion derivatives in local controllability

We can draw a parallel between form-closure grasping of a planar object and small-time local controllability of a planar body with unilateral thrusters. In particular, a first-order analysis of form-closure grasping, considering only the contact normals at the finger locations, indicates that four fingers are necessary to constrain a planar object (Reuleaux [30]). Similarly, a first-order analysis can be used to show that four unilateral thrusters are sufficient for small-time local controllability at zero velocity states. In both cases, the forces (from the frictionless fingers or the thrusters) must positively span the body-fixed three-dimensional force-torque space (f_x, f_y, τ). Four is the minimal number of vectors that can positively span a three-dimensional space.

Higher-order analyses can be used to reduce the number of fingers and thrusters needed. In Proposition 3 we proved that by using Lie brackets, *i.e.* motion derivatives, the number of thrusters required for small-time local controllability is reduced to three. Rimon and Burdick [31] have recently shown that contact curvature, *i.e.* shape derivative, may be used to decrease the number of fingers required for complete immobilization. In particular, they showed that three point fingers are sufficient for form-closure provided their contact normals positively span the plane and intersect at a single point C, and the centers of curvature of the perimeter of the body at the contacts do not all lie at C ([31], Proposition 4.1).

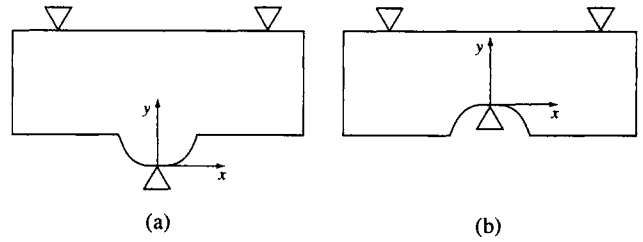


Figure 2: *Three finger grasps of two different objects. By a first-order mobility analysis, the only free motion for both objects is translation along the x-axis. (a) The bottom finger contacts the curve $y = x^4$ at $x = 0$. We have $y'(0) = y''(0) = y^{(3)}(0) = 0$ and $y^{(4)}(0) = 24$. A fourth-order test is required to determine that the object locally curves in the $+y$ direction, away from the finger, and motion along the x-axis is possible. (b) The bottom finger contacts the curve $y = -x^4$ at $x = 0$. A fourth-order test shows that motion along the x-axis is impossible, and the object is completely immobilized.*

Higher-order shape derivatives (not just curvature) can be used in determining form-closure (Figure 2). Bracket terms of higher-order than those used in the proof of Proposition 3 do not create linearly independent vector fields.

3.2 Dynamic pushing

Dynamic pushing is similar to the hovercraft, except the control forces must be realizable by point contact with the perimeter Γ of $\mathcal{B}$. We assume a two-degree-of-freedom point pushing robot, and we place no restrictions on the motion of the pusher.

We can use Proposition 2 to show that any object is controllable by a point robot pushing at a single point of contact on the object, provided friction is nonzero. See Figure 3.

Proposition 4 *For any planar body $\mathcal{B}$ with a closed, piecewise smooth curve Γ of available contact points, there exists a pushing contact point on Γ such that the object is controllable provided the friction coefficient at the contact is nonzero.*

Proof: The radius function $r : \Gamma \to \mathbf{R}$ measures the distance from the center of mass to points on the ob-

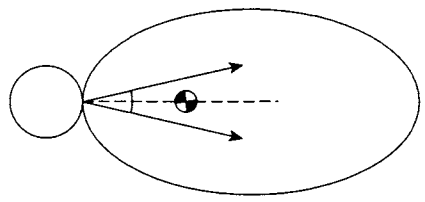


Figure 3: *The object is controllable by pushing at the contact point shown for any nonzero friction coefficient. An example friction cone is illustrated. Both positive and negative torques can be applied through the contact, making the object controllable by Proposition 2.*

ject's perimeter Γ. Assume the curve Γ is parameterized by s. At each point $\Gamma(s)$ where $dr(\Gamma(s))/ds = 0$, the contact normal of Γ passes through the center of mass. (If $dr(\Gamma(s))/ds$ is discontinuous at s, such as at a vertex of a polygon, the contact normal can be chosen as any value in the range defined by the normals as we approach s from both directions.) There are at least two such points because Γ is closed and $r(\Gamma)$ attains at least one local maximum and one local minimum. If Γ is not a single point, there is at least one point $\Gamma(s)$ at which $dr(\Gamma(s))/ds = 0$ and $r(\Gamma(s)) \neq 0$ for any center of mass location. If $r(\Gamma(s)) \neq 0$, $dr(\Gamma(s))/ds = 0$, and the friction coefficient is nonzero, then the center of mass is in the interior of the friction cone and positive and negative torques can be applied from $\Gamma(s)$. Applying Proposition 2, the proof is complete. □

Proposition 5 *Any planar object $\mathcal{B}$ with a closed, piecewise smooth curve Γ of available contact points is locally controllable at all states $(\mathbf{q}, \mathbf{0})$, unless the contact is frictionless and Γ is a circle centered at the object's center of mass.*

Remark. This property ensures that the object can be dynamically pushed to follow any planar path arbitrarily closely (at sufficiently slow speeds). This requires the point robot to switch between distant contacts. Because switching contact points takes finite time, $\mathcal{B}$ is simply locally controllable, not small-time locally controllable. Note that the entire robot-object system does not have this property—the robot must make large motions.

Proof: If Γ is not a circle, the set of frictionless forces that can be applied through Γ (the normals to Γ) positively span the three-dimensional force-torque space (f_x, f_y, τ) (Mishra *et al.* [25]; Markenscoff *et al.* [23]), and the object is locally controllable by Proposition 3. If Γ is a circle, then the frictionless forces through Γ intersect at C, the center of the circle. If the center of mass of $\mathcal{B}$ is not located at C, then $\mathcal{B}$ is locally controllable by Proposition 3. □

Dynamic pushing is, in one sense, a more complete primitive for planar manipulation than pick-and-place using a form- or force-closure grasp. No form- or force-closure grasp exists for a frictionless disk—the disk cannot be rotated by grasping and turning. On the other hand, any frictionless disk not centered at its center of mass is locally controllable by dynamic pushing.

3.3 Underactuated robot

We define R_A to be a fully-actuated 3R robot operating in a horizontal plane, and R_U to be the same robot with a passive third joint. The configuration of the robot is $\Theta = (\theta_1, \theta_2, \theta_3)^T \in M = T^3$. The workspace of the robot is populated with obstacles which include their boundaries, yielding the closed configuration obstacle set $CO \subset M$. We define the collision- and singularity-free configurations of the robot $M_{free} = \{\Theta \in M | \sin\theta_2 \neq 0, \Theta \notin CO\}$. The robot is confined to one of the two open sets $\theta_2 \in (0, \pi)$ or $\theta_2 \in (-\pi, 0)$, the RIGHTY and LEFTY configuration spaces.

Proposition 6 *If there exists a trajectory in M_{free} from $(\Theta_0, \mathbf{0})$ to $(\Theta_1, \mathbf{0})$ for the fully-actuated robot R_A, then there exists a trajectory in M_{free} from $(\Theta_0, \mathbf{0})$ to $(\Theta_1, \mathbf{0})$ for the robot R_U with the third joint passive.*

Proof: This result follows directly from Proposition 3. At any point of M_{free}, the robot can apply forces in any direction through the joint of the third link, the conditions of Proposition 3 are satisfied, and the third link is small-time locally controllable at zero velocity. Any neighborhood of a configuration $\mathbf{q} \in \mathcal{C}$ of the third link maps to a neighborhood of $\Theta = K^{-1}(\mathbf{q}) \in M$, where K^{-1} is the inverse kinematic mapping. This mapping is smooth and one-to-one on the LEFTY and RIGHTY

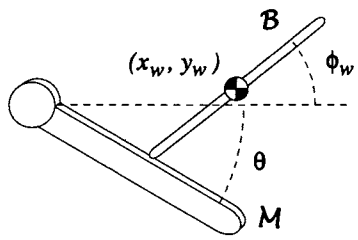


Figure 4: *A one-degree-of-freedom revolute robot manipulating a rod.*

configuration spaces, and $K^{-1}(\mathbf{q}) \in \text{int}(K^{-1}(V(\mathbf{q})))$, where $V(\mathbf{q})$ is any neighborhood of $\mathbf{q}$, so small-time local controllability of the third link on Z implies small-time local controllability of the 3R robot on its zero velocity state space $\{(\Theta, \mathbf{0}) | \Theta \in M_{free}\}$. Therefore, if there is a trajectory for R_A from $(\Theta_0, \mathbf{0})$ to $(\Theta_1, \mathbf{0})$ which remains in M_{free}, then R_U can follow the associated path arbitrarily closely. The motion of R_U may be much slower, since in general it must remain near zero velocity to ensure that the path can be followed closely. □

3.4 One joint robot

The results on dynamic pushing address the theoretical capabilities of dynamic nonprehensile manipulation from the viewpoint of the object alone. Just as important, however, are properties of the manipulator which is controlling the object. While an object may be controllable by point contact, the manipulator may not be able to achieve the contacts and motions necessary to bring the object to the desired state.

Consider the system of Figure 4. The manipulator $\mathcal{M}$ is a single revolute joint, and the object $\mathcal{B}$ is a unit mass rod in point contact with $\mathcal{M}$. The distance from the contact to the rod's center of mass is r and the rod's radius of gyration of inertia is $\rho > 0$. The rod represents an arbitrary polygon in vertex contact with the manipulator. Because the contact is nonprehensile, in any neighborhood of a manipulator-object state, the best we can hope for is small-time accessibility.

The configuration of the manipulator-object system is $\mathbf{q}' = (\mathbf{q}, \theta) \in \mathcal{C}'$, where θ is the angle of the revolute joint. We constrain the manipulator to stay in contact with the rod endpoint at all times, applying zero force (simply "following" the rod) or applying a unit force. Assuming the single link is thin, the three-dimensional submanifold of contact configurations is given by $\{\mathbf{q}' \in \mathcal{C}' \mid F(\mathbf{q}') = \cos\theta(y_w - r\sin\phi_w) + \sin\theta(r\cos\phi_w - x_w) = 0\}$. The equations of motion are

$$\ddot{x}_w = -u\sin\theta \tag{3}$$

$$\ddot{y}_w = u\cos\theta \tag{4}$$

$$\ddot{\phi}_w = -\frac{r}{\rho^2}u\cos(\phi_w - \theta), \tag{5}$$

where the contact force u is

$$\begin{aligned} u = {} & \rho^2(\ddot{\theta}(-r\cos(\phi_w - \theta) + x_w\cos\theta + y_w\sin\theta) + \\ & \dot{\phi}_w^2(-r\sin(\phi_w - \theta)) + \\ & \dot{\theta}^2(-r\sin(\phi_w - \theta) - x_w\sin\theta + y_w\cos\theta) + \\ & \dot{\phi}_w\dot{\theta}(2r\sin(\phi_w - \theta)) + \\ & \dot{\theta}\dot{x}_w(2\cos\theta) + \\ & \dot{\theta}\dot{y}_w(2\sin\theta))/(r^2\cos^2(\phi_w - \theta) + \rho^2). \end{aligned} \tag{6}$$

Rearranging Equation 6, we get the form $\ddot{\theta} = \ddot{\theta}_{drift} + u\ddot{\theta}_{control}$, where $\ddot{\theta}_{drift}$ is the acceleration of the robot needed to stay in contact with the rod while applying zero force, and $\ddot{\theta}_{control}$ is the additional acceleration of the robot required to apply a unit force to the rod. Treating the force u as the control input, the control system on the state space $T\mathcal{C}'$ is $(\dot{\mathbf{q}}', \ddot{\mathbf{q}}') = X_0(\mathbf{q}', \dot{\mathbf{q}}') + uX_1(\mathbf{q}', \dot{\mathbf{q}}')$, where the drift and control vector fields are written

$$\begin{aligned} X_0(\mathbf{q}', \dot{\mathbf{q}}') &= (\dot{x}_w, \dot{y}_w, \dot{\phi}_w, \dot{\theta}, 0, 0, 0, \ddot{\theta}_{drift})^T \\ X_1(\mathbf{q}', \dot{\mathbf{q}}') &= (0, 0, 0, 0, \\ &\quad -\sin\theta, \cos\theta, -\frac{r}{\rho^2}\cos(\phi_w - \theta), \ddot{\theta}_{control})^T. \end{aligned}$$

By examining the Lie algebra of these vector fields, we conclude that the rod is small-time accessible from generic states provided $r \neq 0$. If the robot can also release the part, the entire robot-object system is small-time accessible. The rod is not small-time accessible if the contact point coincides with the pivot of the manipulator—the robot can neither follow the motion of the drifting rod nor apply a force.

3.5 Dynamic workspaces

We would like to characterize the set of robot-object states reachable from another robot-object state. With pick-and-place manipulation, the reachable object states are limited by the reachable configurations of the end-effector, the "kinematic workspace." When we include dynamics, however, the robot is capable of much more. Consider an object thrown to the robot from outside its kinematic workspace. Although the robot cannot initially affect the motion of the object, we know that at some point the robot will be able to contact it and, perhaps, throw it back outside of its kinematic workspace. Given an initial state of the robot-object system, we might define its "dynamic workspace" to be the set of reachable states. This dynamic workspace is a function of the robot, the object, and the initial state.

We illustrate the idea with an example. The two-degree-of-freedom robot is a point which can move freely inside a circle of radius R centered at $(0,0)$. The object being manipulated is a frictionless disk of radius r which is not centered at its center of mass. For simplicity, the configuration of the disk $\mathbf{q} = (x_w, y_w, \phi_w)$ is measured at its center. By Proposition 1, we have small-time accessibility provided the robot can contact the disk and act as a single thruster. Therefore the disk's small-time accessible workspace is defined by $(x_w^2 + y_w^2)^{1/2} < R + r$. For local controllability at zero velocity states, Proposition 3 implies that the robot must be able to contact the object at three points on the disk which are not confined to any closed half-circle. (Note that the robot does not have to be able to contact the disk at all points on its perimeter.) For $R > r$, this condition is satisfied by $(x_w^2 + y_w^2)^{1/2} < (R^2 - r^2)^{1/2}$, which defines the disk's locally controllable workspace. The point robot can maneuver the disk to follow any path inside this workspace arbitrarily closely. If $r \geq R$, this workspace vanishes. See Figure 5.

Now assume a nonzero friction coefficient μ and friction angle α ($\mu = \tan\alpha$) between the robot and the disk. (The center of mass can now lie at the center of the disk.) With a nonzero friction coefficient, local controllability can be obtained from contacts con-

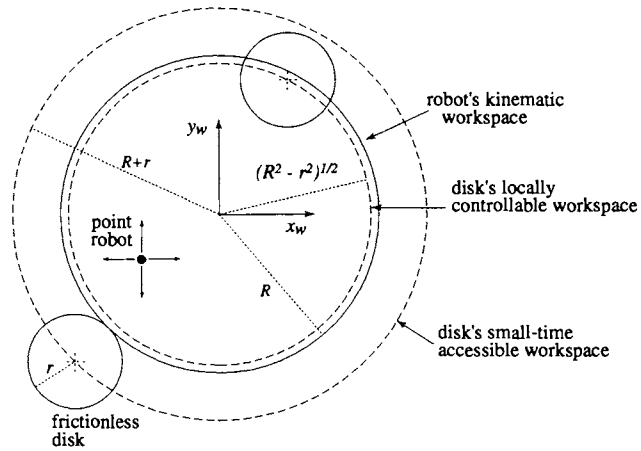


Figure 5: *The point robot, which can move freely in a circle of radius R, manipulates a frictionless disk (not centered at its center of mass) of radius r. The disk is small-time accessible provided its center stays in a circle of radius $R+r$, and it is locally controllable provided its center stays in a circle of radius $(R^2 - r^2)^{1/2}$. Disks are shown near the boundaries of the small-time accessible and locally controllable workspaces.*

fined to a closed half-circle (see Figure 6). As the friction angle α increases, the overlap between the robot's workspace and the disk required for local controllability decreases—the disk is locally controllable if $(x_w^2 + y_w^2)^{1/2} < (R^2 - r^2\cos\alpha)^{1/2} + r\sin\alpha$. As α approaches $\pi/2$ (infinite friction), the locally controllable workspace approaches the small-time accessible workspace, which is independent of the friction coefficient.[1]

Finally, we have accessibility if the disk can reach a full-dimensional subset of its state space, not necessarily in small-time. The workspace from which the disk is accessible is the set of states that will carry the disk into its small-time accessible workspace. An example is shown in Figure 7.

We call a system *dynamically singular* at any state outside the small-time accessible workspace. The mo-

[1] We assume that the robot can apply any force in the friction cone. In some cases this may result in "pulling"—the robot moves in a direction with a negative component in the contact normal (Lynch and Mason [18]).

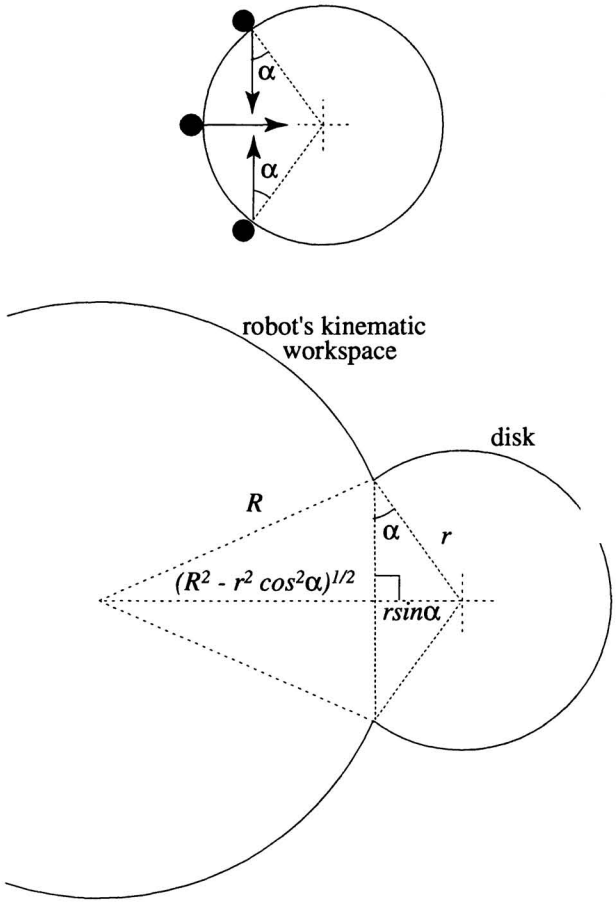


Figure 6: *Top: At the friction angle α, forces through the three robot contact points pass through a single point and positively span a half-plane. At any greater value of friction, the condition of Proposition 3 is satisfied, and the disk is locally controllable. Bottom: Determining the minimum overlap of the disk and the robot workspace that allows the disk to be locally controllable by robot pushing with a friction angle α.*

tion directions of the system are locally constrained to a lower-dimensional submanifold of the state space.

Obtaining a precise characterization of the accessible state space of an object during nonprehensile manipulation appears to be a very difficult problem in general. An approximate representation can be found by sampling the possible controls, simulating the system, and plotting the trajectories of the object.

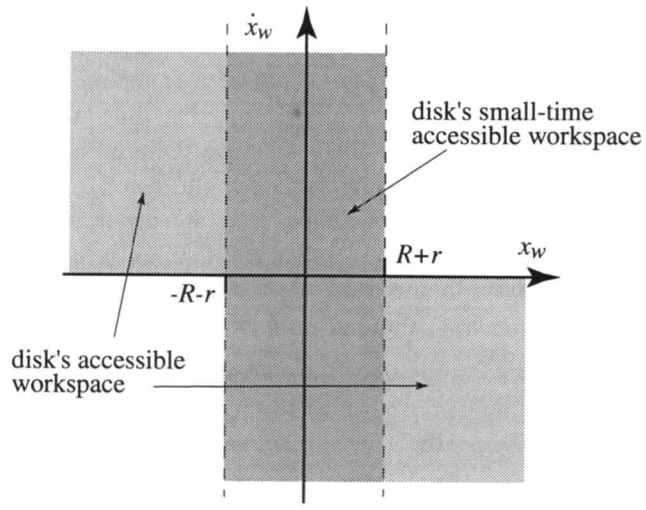


Figure 7: *The $(x_w, \dot{x}_w)$ state space for a disk centered at its center of mass, with $y_w = \dot{y}_w = 0$, ϕ_w and $\dot{\phi}_w$ arbitrary, and nonzero contact friction. The disk is small-time accessible in the band defined by $-R - r < x_w < R + r$, and the disk is accessible from the other shaded regions. The disk drifts into the small-time accessible band.*

4 Motion Planning

Nonprehensile manipulation is characterized by nonholonomic equality and inequality constraints. The former arise from using low degree-of-freedom robots to manipulate objects with more degrees-of-freedom; the latter arise from unilateral contact, friction constraints, and actuator constraints. We can distinguish between three basic approaches to the motion planning problem for nonholonomic systems (see also Popa and Wen [29]):

1) Control-theoretic algorithms make significant use of the structure of the system. Examples include systems without drift (Lafferiere and Sussmann [13]), systems which can be put into chained or differentially flat form (Murray *et al.* [26]; Murray *et al.* [27]), and Caplygin systems (Bloch *et al.* [4]). These approaches typically do not incorporate nonholonomic inequality constraints (as arise in nonprehensile contact) or configuration constraints such as obstacles and joint limits.

2) Search-based algorithms sample the control space and can easily incorporate nonholonomic equality, in-

equality, and configuration constraints. Straightforward search can be quite general, requiring little information about the system, but as a result may suffer from high computational complexity. Information about the structure of the system can be used to perform a principled discretization of the control space (based on a critical decomposition or extremal controls), resulting in more efficient planners (*e.g.*, the shortest path information used in the mobile robot path planner of (Laumond *et al.* [14])).

3) Gradient-descent algorithms attempt to use smoothness in the input-output map to speed convergence to a solution. Typically a finite-dimensional parameterization of the control is chosen (for instance using Fourier bases, polynomials, splines, or piecewise constant functions), and the problem is cast as a constrained nonlinear optimization problem. This approach suffers from the same problems common to all gradient-descent optimization: local optima, possibly slow convergence, and numerical instabilities (particularly when using finite-difference gradients). Gradient descent for the control of nonholonomic systems has been explored by many (see, for example, (Divelbiss and Wen [7]; Sussmann [35]; Popa and Wen [29]; Sontag [34]; Fernandes *et al.* [9]; Žefran *et al.* [39])).

These methods can be used in concert; in particular, gradient descent can be used to locally optimize a solution found by another method.

Below we briefly describe a search-based planner for the underactuated robot and a gradient-descent approach to motion planning for a one joint robot (Lynch and Mason [19]).

4.1 A search-based planner for the underactuated robot

We would like to find fast, collision-free trajectories from $(\Theta_{init}, \mathbf{0})$ to $(\Theta_{goal}, \mathbf{0})$ for the underactuated 3R robot. Because the robot is small-time locally controllable, it can follow any path closely. Arbitrary paths require the robot to stay near zero velocity, however. To find paths that can be executed quickly, we prefer to minimize the use of Lie bracket motions. We describe a planner that uses primitive motions corresponding to $\mathbf{f}_1 = (1,0,0)^T$ and $\mathbf{f}_2 = (0,1,\tau)^T$ ($X_1 = (0,0,0,\cos\phi_w,\sin\phi_w,0)^T$ and $X_2 = (0,0,0,-\sin\phi_w,\cos\phi_w,\tau)$) and minimizes the number of switches between them, implicitly minimizing the use of Lie bracket motions. By bringing the link velocity to zero at the switches, X_1 and X_2 yield velocity directions fixed in the frame of the passive link: pure translation along the link and pure rotation about the *center of percussion* of the link with respect to the joint. The vector fields X_1 and X_2 are unique in this respect, and for the purpose of path planning, this property allows us to treat the system as a first-order system with velocity vector fields $X_1' = (\cos\phi_w,\sin\phi_w,0)^T$ and $X_2' = (-\sin\phi_w,\cos\phi_w,\tau)^T$. This allows us to decouple the full trajectory planning problem into the computationally simpler problems of path planning and time-scaling the path according to the manipulator dynamics and actuator constraints.

The planner is a modification of a path planning algorithm for mobile robots (Barraquand and Latombe [3]) and robotic pushing (Lynch and Mason [20]). It is a simple best-first search which minimizes the number of switches between vector fields. Starting from the initial link configuration $\mathbf{q}_{init}$ (and corresponding initial joint configuration Θ_{init}), the planner integrates forward along each of $+X_1'$, $-X_1'$, $+X_2'$, and $-X_2'$ for a time δt, yielding four new link configurations. Each new collision-free configuration $\mathbf{q}_{new}$ (and corresponding joint angles Θ_{new}) is added to a search tree T and to a sorted list *OPEN* of configurations in T whose successors have not yet been generated. Configurations in *OPEN* are sorted by the number of switches in their paths. The first configuration in *OPEN* is then expanded. This process continues until a path is found to a user-specified goal neighborhood $\mathcal{G}(\mathbf{q}_{goal})$ or until *OPEN* is empty (failure). Note that the planner is not exact, as it only finds a path to a goal neighborhood.

For a small enough δt, this planner will find a solution when one exists. The planner is *resolution-complete*.

Once a path is found, we can apply the algorithms proposed by Shin and McKay [32] or Bobrow *et al.* [5] to find the time-optimal time-scaling of each motion

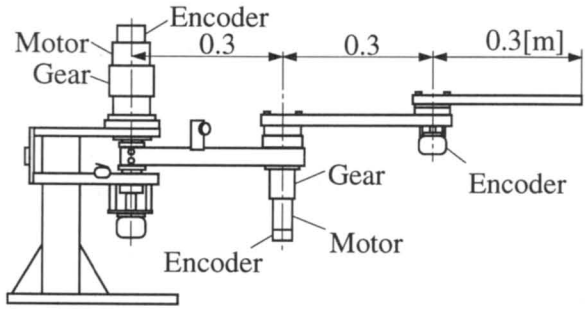


Figure 8: *The MEL underactuated manipulator.*

segment of the path. The concatenation of these trajectories yields the time-optimal trajectory following the path returned by the planner.

We implemented planner results on the Mechanical Engineering Laboratory underactuated manipulator shown in Figure 8. Nonlinear feedback control is used to stabilize the motion of the third link during the rotational and translational motion segments (Shiroma *et al.* [33]). Time-optimal trajectories cause actuators to saturate, allowing no margin for error-recovery in feedback control. In our experiments, times for each motion segment were chosen experimentally for fast but robust performance.

Figure 9 shows an 11 segment path successfully implemented on the robot. The obstacles shown in the figure have been grown by 2 cm in each direction over the real experimental setup, providing room for error in execution. Joint limits are $-1.17\,\text{rad} < \theta_1 < 1.17\,\text{rad}$ (where 0 is vertical on the page) and $0.5\,\text{rad} < \theta_2 < 1.95\,\text{rad}$.

4.2 Gradient-descent motion planning for the one joint robot

A one joint robot such as that of Figure 4 can control more degrees-of-freedom of an object by sequencing rolling, slipping, and free flight phases. We have previously described a motion planner for a 1 DOF robot performing rolling and throwing manipulation (Lynch and Mason [19]). This planner takes an initial guess for the trajectory of the robot and a sequence of manipulation phases (rolling, free flight, etc.) specified by the

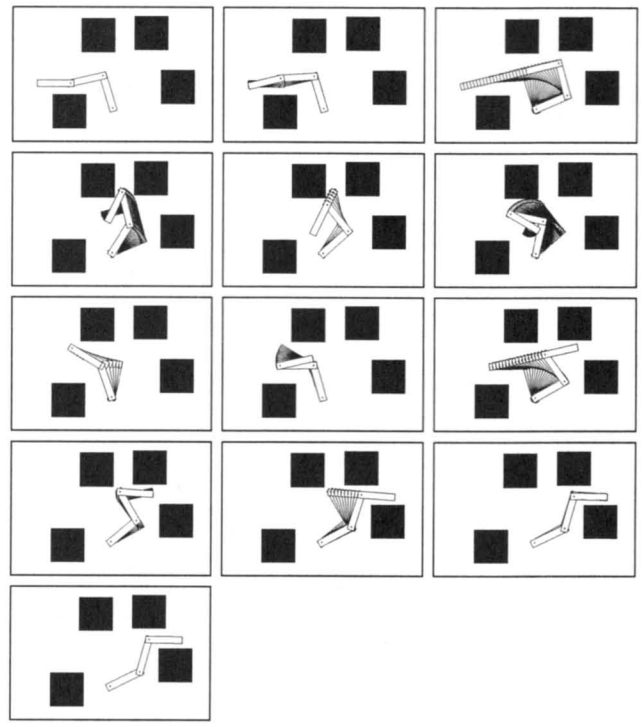


Figure 9: *A path found by the planner using grown obstacles.*

user, simulates the motion of the object, and using the results of the simulation, performs gradient-descent in the space of robot trajectories to find a motion which more nearly satisfies the goal state of the object and other constraints, such as friction inequalities and actuator limits. The manipulation problem is cast as a nonlinear optimization which is solved using sequential quadratic programming. The result is a robot motion, represented as a cubic spline, which takes the object to the goal state and locally optimizes a cost function. Several rolling and throwing tasks solved by the planner were successfully implemented on a real 1 DOF robot (see Figure 10). Žefran *et al.* [39] described a similar approach using nonlinear optimization for motion planning for systems with unilateral and changing dynamic constraints.

This system has much less apparent structure than the underactuated robot, so the gradient-descent planner assumes only smoothness in the input-output map. A major limitation of this planner relative to the

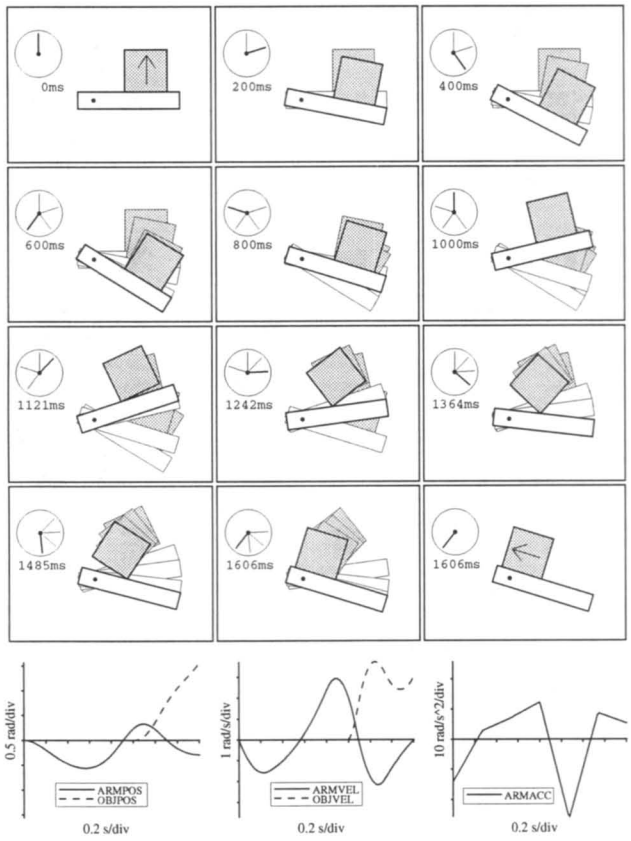


Figure 10: *A 1 DOF robot motion to roll the square from one edge to another, found by the nonlinear optimization.*

planner for the underactuated robot is that it is not complete—it may not find a solution when one exists. Also, because the set of achievable tasks is not obvious, it is not clear when the planner has failed and when the desired goal state is unreachable. This problem points out the need for a better understanding of the dynamic workspace of the robot-object system. Also, the presence of inequality constraints due to unilateral contact and bounded friction exacerbates convergence and local minima problems. It appears that this gradient-descent approach is better suited for local tuning of a rough solution than global motion planning.

5 Conclusion

Nonprehensile manipulation is a rich source of geometric problems. Examples include determining controllability (as discussed in this paper) and feedability (Akella *et al.* [2]; van der Stappen *et al.* [38]) based on the part geometry, contact friction, and mass distribution. These problems bear a resemblance to problems in robot grasping—characterizing graspable and ungraspable objects and determining the minimum number of fingers required for a grasp. The controllability and feedability properties are also closely bound up with characteristics of the manipulator, which may have fewer degrees-of-freedom than the object.

Nonprehensile manipulation also poses challenging problems for motion planning algorithms. We have described a gradient-descent motion planner which finds motion plans through a specified sequence of manipulation phases. Future motion planners should automatically determine a sequence of phases leading to the goal state. Also, the current motion planner makes very little use of the structure of the problem. Future work should be towards identifying and using this structure to derive complete motion planners. An example of using the structure of a nonprehensile manipulation problem is Erdmann's [8] planner for two palm manipulation based on a critical decomposition of the control space.

Given the ubiquity of nonprehensile manipulation in everyday tasks and industrial applications such as vibratory parts feeding, a formal attack on the issues outlined in this paper should go a long way toward building a science of manipulation.

Acknowledgments

This paper describes joint work with Matt Mason at Carnegie Mellon, and Naoji Shiroma, Hirohiko Arai, and Kazuo Tanie at the Biorobotics Division, Mechanical Engineering Laboratory, Tsukuba, Japan. This work was supported by a grant from NSF at Carnegie Mellon, a postdoctoral fellowship from the Science and Technology Agency of Japan, the Mechanical Engineering Laboratory, and Northwestern University.

References

[1] Y. Aiyama, M. Inaba, and H. Inoue. Pivoting: A new method of graspless manipulation of object by robot

fingers. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 136–143, Yokohama, Japan, 1993.

[2] S. Akella, W. Huang, K. M. Lynch, and M. T. Mason. Planar manipulation on a conveyor with a one joint robot. In *International Symposium on Robotics Research*, pages 265–276, 1995.

[3] J. Barraquand and J.-C. Latombe. Nonholonomic multibody mobile robots: Controllability and motion planning in the presence of obstacles. *Algorithmica*, 10:121–155, 1993.

[4] A. M. Bloch, M. Reyhanoglu, and N. H. McClamroch. Control and stabilization of nonholonomic dynamic systems. *IEEE Transactions on Automatic Control*, 37(11):1746–1757, 1992.

[5] J. E. Bobrow, S. Dubowsky, and J. S. Gibson. Time-optimal control of robotic manipulators along specified paths. *International Journal of Robotics Research*, 4(3):3–17, Fall 1985.

[6] M. Bühler and D. E. Koditschek. From stable to chaotic juggling: Theory, simulation, and experiments. In *IEEE International Conference on Robotics and Automation*, pages 1976–1981, Cincinnati, OH, 1990.

[7] A. W. Divelbiss and J. Wen. Nonholonomic path planning with inequality constraints. In *IEEE International Conference on Decision and Control*, pages 2712–2717, 1993.

[8] M. A. Erdmann. An exploration of nonprehensile two-palm manipulation using two zebras. In *Algorithms for Robotic Motion and Manipulation, WAFR 1996*, pages 239–254. A. K. Peters, Boston, MA, 1996.

[9] C. Fernandes, L. Gurvits, and Z. Li. Near-optimal nonholonomic motion planning for a system of coupled rigid bodies. *IEEE Transactions on Automatic Control*, 30(3):450–463, Mar. 1994.

[10] R. Hermann and A. J. Krener. Nonlinear controllability and observability. *IEEE Transactions on Automatic Control*, AC-22(5):728–740, Oct. 1977.

[11] T. Higuchi. Application of electromagnetic impulsive force to precise positioning tools in robot systems. In *International Symposium on Robotics Research*, pages 281–285. Cambridge, Mass: MIT Press, 1985.

[12] W. Huang, E. P. Krotkov, and M. T. Mason. Impulsive manipulation. In *IEEE International Conference on Robotics and Automation*, pages 120–125, 1995.

[13] G. Lafferiere and H. Sussmann. Motion planning for controllable systems without drift. In *IEEE International Conference on Robotics and Automation*, pages 1148–1153, Sacramento, CA, 1991.

[14] J.-P. Laumond, P. E. Jacobs, M. Taïx, and R. M. Murray. A motion planner for nonholonomic mobile robots. *IEEE Transactions on Robotics and Automation*, 10(5):577–593, Oct. 1994.

[15] A. D. Lewis and R. M. Murray. Configuration controllability of simple mechanical control systems. *SIAM Journal on Control and Optimization*, 35(3):766–790, May 1997.

[16] T. Lozano-Pérez, J. L. Jones, E. Mazer, and P. A. O'Donnell. *HANDEY: A Robot Task Planner*. The MIT Press, 1992.

[17] K. M. Lynch. Locally controllable polygons by stable pushing. In *IEEE International Conference on Robotics and Automation*, pages 1442–1447, 1997.

[18] K. M. Lynch and M. T. Mason. Pulling by pushing, slip with infinite friction, and perfectly rough surfaces. *International Journal of Robotics Research*, 14(2):174–183, Apr. 1995.

[19] K. M. Lynch and M. T. Mason. Dynamic underactuated nonprehensile manipulation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 889–896, 1996.

[20] K. M. Lynch and M. T. Mason. Stable pushing: Mechanics, controllability, and planning. *International Journal of Robotics Research*, 15(6):533–556, Dec. 1996.

[21] K. M. Lynch and M. T. Mason. Dynamic nonprehensile manipulation: Controllability, planning, and experiments, 1998. *International Journal of Robotics Research*, to appear.

[22] V. Manikonda and P. S. Krishnaprasad. Controllability of Lie-Poisson reduced dynamics. Institute for Systems Research 57-59, University of Maryland, May 1997.

[23] X. Markenscoff, L. Ni, and C. H. Papadimitriou. The geometry of grasping. *International Journal of Robotics Research*, 9(1):61–74, Feb. 1990.

[24] M. T. Mason. Mechanics and planning of manipulator pushing operations. *International Journal of Robotics Research*, 5(3):53–71, Fall 1986.

[25] B. Mishra, J. T. Schwartz, and M. Sharir. On the existence and synthesis of multifinger positive grips. *Algorithmica*, 2(4):541–558, 1987.

[26] R. M. Murray, Z. Li, and S. S. Sastry. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, 1994.

[27] R. M. Murray, M. Rathinam, and W. Sluis. Differential flatness of mechanical control systems: A catalog of prototype systems. In *ASME Int Mech Eng Congress and Expo*, 1995.

[28] H. Nijmeijer and A. J. van der Schaft. *Nonlinear Dynamical Control Systems*. Springer-Verlag, 1990.

[29] D. O. Popa and J. T. Wen. Nonholonomic path-planning with obstacle avoidance: A path-space approach. In *IEEE International Conference on Robotics and Automation*, pages 2662–2667, 1996.

[30] F. Reuleaux. *The Kinematics of Machinery*. MacMillan, 1876. Reprinted by Dover, 1963.

[31] E. Rimon and J. W. Burdick. New bounds on the number of frictionless fingers required to immobilize planar objects. *Journal of Robotic Systems*, 12(6):433–451, 1995.

[32] K. G. Shin and N. D. McKay. Minimum-time control of robotic manipulators with geometric path constraints. *IEEE Transactions on Automatic Control*, 30(6):531–541, June 1985.

[33] N. Shiroma, H. Arai, and K. Tanie. Nonlinear control of a planar free link under a nonholonomic constraint. In *International Conference on Advanced Robotics*, pages 103–109, 1997.

[34] E. Sontag. Gradient techniques for systems with no drift: A classical idea revisited. In *IEEE International Conference on Decision and Control*, pages 2706–2711, 1993.

[35] H. Sussmann. A continuation method for nonholonomic path-finding problems. In *IEEE International Conference on Decision and Control*, pages 2718–2723, 1993.

[36] H. J. Sussmann. A general theorem on local controllability. *SIAM Journal on Control and Optimization*, 25(1):158–194, Jan. 1987.

[37] J. C. Trinkle and D. C. Zeng. Prediction of the quasistatic planar motion of a contacted rigid body. *IEEE Transactions on Robotics and Automation*, 11(2):229–246, Apr. 1995.

[38] A. F. van der Stappen, K. Goldberg, and M. H. Overmars. Geometric eccentricity and the complexity of manipulation plans. Submitted to *Algorithmica*, November 1996.

[39] M. Žefran, J. Desai, and V. Kumar. Continuous motion plans for robotic systems with changing dynamic behavior. In J.-P. Laumond and M. Overmars, editors, *Algorithms for Robotic Motion and Manipulation*. A. K. Peters, Boston, MA, 1996.

[40] N. B. Zumel and M. A. Erdmann. Nonprehensile two palm manipulation with non-equilibrium transitions between stable states. In *IEEE International Conference on Robotics and Automation*, pages 3317–3323, 1996.

Local Observability of Rolling

Yan-Bin Jia, *Carnegie Mellon University, Pittsburgh, PA, USA*
Michael Erdmann, *Carnegie Mellon University, Pittsburgh, PA, USA*

This paper investigates local observability of the pose and motion of a curved three-dimensional object rolling on a rough horizontal plane. The plane models a controllable robotic palm imbued with tactile sensors. The palm can accelerate in arbitrary translational directions and the tactile sensors can determine the contact location between the palm and the rolling object at every instant in time. The object and contact motions are governed by a nonlinear system derived from the kinematics and dynamics of rolling. Through cotangent space decomposition, a sufficient condition on local observability of the system is obtained. This condition depends only on the differential geometry of contact and on the object's angular inertia matrix; it is satisfied by all but some degenerate shapes such as a sphere.

The above result demonstrates that the geometric and dynamic information of a manipulation task is often encoded in a small amount of tactile data.

1 Introduction

Geometry and mechanics are always closely tied to each other in a manipulation task. The states (or configurations) of an object and its manipulator evolve under the laws of mechanics and subject to the geometric constraints of contact. Such interaction often yields simple information into which the geometry and motions of the object and manipulator are encoded by the mechanics of the manipulation. For example, the contact between the object and the manipulator is a form of interaction through which the object's pose and motion may be conveyed to the manipulator in an implicit way. The reaction force is viewed as another form of interaction. From these kinds of encoded information we may be able to recover some essential unknowns about the task, especially the configuration (pose and motion) of the object.

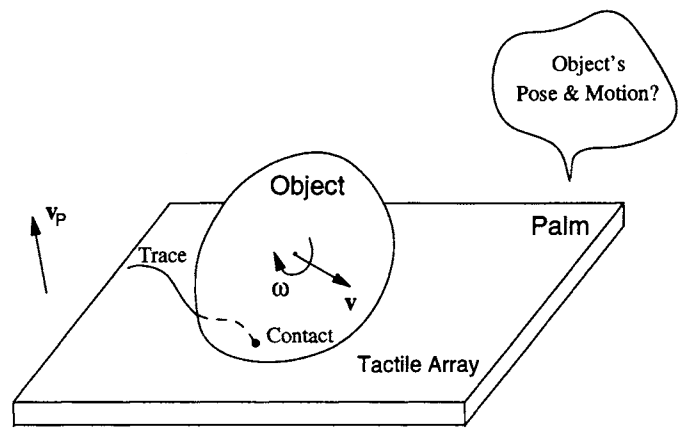


Figure 1: *Palm sensing.*

In this paper we will look into the contact information and see how much about a task can be reviewed from contact. More specifically, we would like to find out whether the contact between an object and a manipulator could be enough for the latter to "observe", at least locally, the pose and motion of the former.

The task chosen for our study involves a horizontal plane, or a palm, which can translate in arbitrary directions, and a smooth object which can only roll on the palm. As shown in Figure 1, the palm is trying to control the object's motion by executing certain motion of its own. The state of the object, comprising its pose and motion, however, is unknown at the moment. So the palm needs to determine the state before purposefully manipulating the object. We refer to this state estimation problem as *palm sensing.*

As the object rolls, its contact traces out a curve as a function of time in the plane (see Figure 1). Suppose

the palm is covered with an array of tactile cells which are able to detect the contact location at any time instant. In this paper we would like to know if this curve of contact contains enough information for the palm to know about the configuration of the object. Our approach is to study local observability of rolling from the contact curve. Through this investigation we hope to touch on the more general issue of mechanics-based sensing and information retrieval in robotic tasks.

Section 2 derives a nonlinear system from the kinematics and dynamics of rolling that describes the object's motion as well as the contact motions upon the surfaces of the object and palm. Section 3 establishes a sufficient condition on local observability for the configuration of the rolling object. This condition depends only on the object's angular inertia matrix and its local shape around the contact. Section 4 summarizes the result and addresses two research directions in the future.

1.1 Related work

We relate our work to previous work in rigid body dynamics, contact kinematics, sensing, and parts orienting. The work also draws upon the part of nonlinear control theory concerned with observability.

The first general discussion on the motion of a rigid body was due to Poisson in 1838, though the special case of a sphere was treated by Coriolis earlier in 1835. A few years later, friction was introduced into rigid body dynamics by Cournot in volumes 5 and 8 of *Crelle's Journal.* In 1861, Slesser gave the equations of a rigid body constrained to roll and pivot without sliding on a horizontal plane.

In his comprehensive introduction to rigid body dynamics [20], MacMillan coped with rolling in the case of a sphere on a smooth surface.

Observing the duality between a force and point, Brost and Mason [2] described a graphical method for the analysis of rigid body motion with multiple frictional contacts in the plane. Pang and Trinkle [23] introduced complementarity formulations to predict the accelerations and contact forces of multiple rigid bodies in contact with Coulomb friction.

Montana [21] derived a set of differential equations that govern the motion of a contact point in response to a relative motion of the objects in contact, and applied these equations to local curvature estimation and contour following. The kinematics of spatial motion with point contact was also studied by Cai and Roth [3] who assumed a tactile sensor able to measure the relative motion at the contact point. The special kinematics of two rigid bodies rolling on each other was treated by Li and Canny [17] in view of path planning. Assuming rolling contact only, Kerr and Roth [16] combined the forward kinematics of multifingered hands and the contact kinematics to solve for the finger joint velocities that are necessary for achieving a desired object motion.

Orienting mechanical parts was early studied by Grossman and Blasgen [10]. They used a vibrating box to constrain a part to a small finite number of possible stable poses and then determined the particular pose by a sequence of probes using a tactile sensor. Inspired by their result, Erdmann and Mason [6] constructed a planner that employs sensorless tilting operations to orient planar objects randomly dropped into a tray, based on a simple model of the quasi-static mechanics of sliding. Erdmann [5] also developed a working system for orienting parts using two palms, together with frictional contact analysis tools that can predict relative slip between parts.

Utilizing the theory of limit surfaces [8], Böhringer *et al.* [1] developed a geometric model for the mechanics of an array of microelectromechanical structures and showed how this structure can be used to uniquely and efficiently align a part up to symmetry. Approximating the limit surface of an object as an ellipsoid, Lynch *et al.* [19] designed a control system that can translate the object by pushing with tactile feedback.

Salisbury [24] first proposed the concept of fingertip force sensing with an approach for determining contact locations and orientations from force and moment measurements. Howe and Cutkosky [12] introduced dynamic tactile sensing in which sensors capture fine surface features during motion.

Grimson and Lozano-Pérez [9] used tactile measurements of positions and surface normals on a 3D object to identify and locate it from a set of known 3D objects, based on the geometric constraints imposed by these tactile data. Gaston and Lozano-Pérez [7] showed how to identify and locate a polyhedron on a known plane using local information from tactile sensors which includes the position of contact points and the ranges of surface normals at these points.

Hermann and Kerner [11] first studied observability using the observation space. A result due to Crouch [4] shows that an analytic system is observable if and only if the observation space distinguishes points in the state space. Luenberger-like asymptotic observers, first constructed by Luenberger [18] for linear systems, remain likely the most commonly used observer forms for nonlinear systems today.

Jia and Erdmann [14, 15] investigated how to "observe" a planar object being pushed by a finger. They established the local observability of the object's pose and motion from the contact motion on the finger; and introduced two nonlinear observers as sensing strategies.

2 The Rolling Motion

The motion of an object rolling without slipping on a plane is subject to two constraints: (1) The point in contact has zero velocity relative to the plane; and (2) the object has no rotation about the contact normal. In this section we shall study the rolling motion of an object and the resulting motions of the contact on the object and in the plane. We shall derive a system of nonlinear differential equations that describes the object's kinematics and dynamics.

Before we begin, the problem needs to be formally defined. As shown in Figure 1, it concerns an object $\mathcal{B}$ (with or without any initial velocity) moving on a horizontal plane $\mathcal{P}$, which is translating at velocity $\boldsymbol{v}_{\mathcal{P}}$. In order to maintain contact, the plane acceleration $\dot{\boldsymbol{v}}_{\mathcal{P}}$ must not exceed the gravitational acceleration g downward. The friction between $\mathcal{P}$ and $\mathcal{B}$ is assumed to be large enough to allow only the pure rolling motion of $\mathcal{B}$ (without slipping).

For the sake of simplicity and clarity, we make a few additional assumptions:

1. Object $\mathcal{B}$ has uniform mass distribution.
2. Object $\mathcal{B}$ is bounded by an orientable surface.
3. The contact stays in one proper patch $\boldsymbol{\beta}$ in the surface of $\mathcal{B}$ during the period of time when local observability is concerned.
4. The patch $\boldsymbol{\beta}$ is convex and has positive Gaussian curvature everywhere.
5. The patch $\boldsymbol{\beta}$ is principal.

The third assumption makes sense because local observability is concerned with an infinitesimal amount of time. The fourth assumption restricts the contact to a point. The fifth assumption is justified because every point in a surface has a neighborhood that can be reparametrized as a principal patch (see [25, pp. 320-323] for a proof).

2.1 Kinematics of rolling

To describe the motion of $\mathcal{B}$ and the motions of contact on both $\mathcal{P}$ and $\mathcal{B}$, we here set up several coordinate frames. Let $\boldsymbol{o}$ be the center of mass of $\mathcal{B}$. Let the coordinate frame $\Pi_{\mathcal{B}}$ as shown in Figure 2 be centered at $\boldsymbol{o}$ and defined by $\mathcal{B}$'s principal axes $\boldsymbol{x}_{\mathcal{B}}$, $\boldsymbol{y}_{\mathcal{B}}$, and $\boldsymbol{z}_{\mathcal{B}}$. The angular inertia matrix I with respect to $\Pi_{\mathcal{B}}$ is thus diagonal. Frame $\Pi_{\mathcal{B}}$ is moving constantly relative to the plane $\mathcal{P}$ due to the rolling of $\mathcal{B}$.

A frame $\Pi_{\mathcal{P}}$ with axes $\boldsymbol{x}_{\mathcal{P}}$, $\boldsymbol{y}_{\mathcal{P}}$, and $\boldsymbol{z}_{\mathcal{P}}$ is attached to the plane $\mathcal{P}$ such that axis $\boldsymbol{z}_{\mathcal{P}}$ is an upward normal to $\mathcal{P}$. The world coordinate frame, denoted by $\Pi_{\mathcal{W}}$, has the same orientation as frame $\Pi_{\mathcal{P}}$.

It is often convenient to describe the motion of $\mathcal{B}$ in terms of its body frame $\Pi_{\mathcal{B}}$ by velocity $\boldsymbol{v}$ and angular velocity $\boldsymbol{\omega}$. In the meantime, denote by $\boldsymbol{v}_{\mathcal{B}}$ and $\boldsymbol{\omega}_{\mathcal{B}}$ the velocities of $\mathcal{B}$ relative to the fixed frame that currently coincide with $\Pi_{\mathcal{B}}$. Hence $\boldsymbol{v}_{\mathcal{B}}$ and $\boldsymbol{v}$ are identical at the moment, so are $\omega_{\mathcal{B}}$ and $\boldsymbol{\omega}$. But their derivatives have quite different meanings, as will be discussed in Section 2.2.

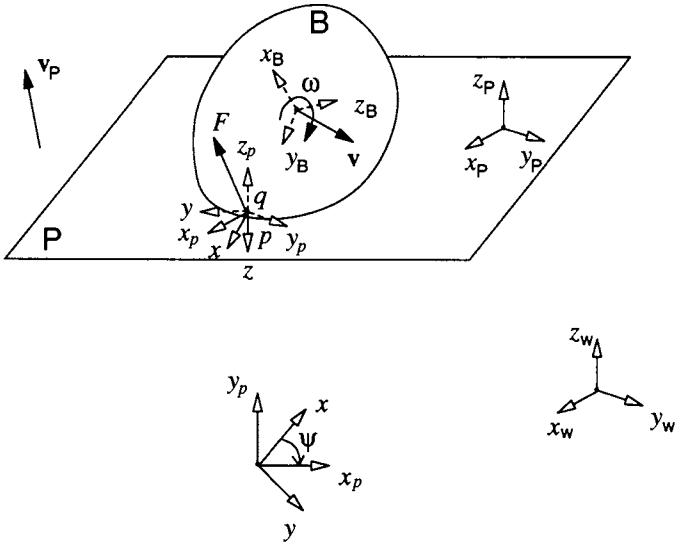


Figure 2: *The coordinate frames for rolling. The body frame $x_{\mathcal{B}}$-$y_{\mathcal{B}}$-$z_{\mathcal{B}}$ of object $\mathcal{B}$ defined by its principal axes; the frame $x_{\mathcal{P}}$-$y_{\mathcal{P}}$-$z_{\mathcal{P}}$ of plane $\mathcal{P}$; the frame x-y-z at the contact q on $\mathcal{B}$; the frame x_p-y_p-z_p at the contact p in $\mathcal{P}$; and the world frame $x_{\mathcal{W}}$-$y_{\mathcal{W}}$-$z_{\mathcal{W}}$. The frames $x_{\mathcal{P}}$-$y_{\mathcal{P}}$-$z_{\mathcal{P}}$, x_p-y_p-z_p, and $x_{\mathcal{W}}$-$y_{\mathcal{W}}$-$z_{\mathcal{W}}$ have the same orientation.*

The contact point on $\mathcal{B}$ is denoted by $\boldsymbol{q} = \boldsymbol{\beta}(\boldsymbol{s}) = \boldsymbol{\beta}(s_1, s_2)$ in $\Pi_{\mathcal{B}}$. Since $\boldsymbol{\beta}$ is a principal patch, the normalized Gauss frame Π at $\boldsymbol{q}$ is well-defined by axes

$$\boldsymbol{x} = \frac{\boldsymbol{\beta}_{s_1}}{\|\boldsymbol{\beta}_{s_1}\|}, \quad \boldsymbol{y} = \frac{\boldsymbol{\beta}_{s_2}}{\|\boldsymbol{\beta}_{s_2}\|}, \quad \boldsymbol{z} = \frac{\boldsymbol{\beta}_{s_1} \times \boldsymbol{\beta}_{s_2}}{\|\boldsymbol{\beta}_{s_1}\| \cdot \|\boldsymbol{\beta}_{s_2}\|},$$

all with respect to the body frame $\Pi_{\mathcal{B}}$. The parameters $\boldsymbol{s}$ of $\boldsymbol{\beta}$ are chosen such that $\boldsymbol{z}$ is the outward normal. The orientation of Π relative to the body frame $\Pi_{\mathcal{B}}$ is thus given by the 3×3 rotation matrix

$$R = (\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}). \tag{1}$$

Meanwhile, the contact point in the plane $\mathcal{P}$ is denoted by $\boldsymbol{p} = (\boldsymbol{u}, 0)^T = (u_1, u_2, 0)^T$ in frame $\Pi_{\mathcal{P}}$. We attach to $\boldsymbol{p}$ a frame Π_p (with axes $\boldsymbol{x}_p, \boldsymbol{y}_p, \boldsymbol{z}_p$) with the same orientation as frames $\Pi_{\mathcal{P}}$ and $\Pi_{\mathcal{W}}$. Let ψ be the angle of rotation needed to align axis $\boldsymbol{x}$ with axis $\boldsymbol{x}_p$ (see Figure 2). The matrix

$$R_\psi = \begin{pmatrix} \cos\psi & -\sin\psi \\ -\sin\psi & -\cos\psi \end{pmatrix}$$

therefore relates axes $\boldsymbol{x}, \boldsymbol{y}$ to axes $\boldsymbol{x}_p, \boldsymbol{y}_p$. Accordingly, Π is related to Π_p by the 3×3 rotation matrix

$$R_{pq} = \begin{pmatrix} R_\phi & 0 \\ 0 & -1 \end{pmatrix}.$$

Consequently, the orientation of the body frame $\Pi_{\mathcal{B}}$ relative to the contact frame Π_p is given by the matrix

$$\begin{aligned} R_{po} &= R_{pq} R^T \\ &= \begin{pmatrix} R_\phi & 0 \\ 0 & -1 \end{pmatrix} (\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})^T \\ &= \begin{pmatrix} R_\phi (\boldsymbol{x}, \boldsymbol{y})^T \\ -\boldsymbol{z}^T \end{pmatrix}. \end{aligned} \tag{2}$$

Now we see that the orientation of $\mathcal{B}$ relative to $\mathcal{P}$ (and thereby to the world frame $\Pi_{\mathcal{W}}$) is completely determined by $\boldsymbol{s}$ and ψ, which represent the three degrees of freedom of $\mathcal{B}$ in order to maintain contact with $\mathcal{P}$. The velocity and angular velocity of $\mathcal{B}$ in $\Pi_{\mathcal{W}}$ are given by $R_{po}\boldsymbol{v}$ and $R_{po}\boldsymbol{\omega}$, respectively.

The contact kinematics depends on the relative motion between the two contact frames Π and Π_p. Denote by $(\omega_x, \omega_y, \omega_z)^T$ the angular velocity of Π relative to Π_p and in terms of Π. Since Π is fixed relative to $\Pi_{\mathcal{B}}$ and the angular velocity of $\Pi_{\mathcal{B}}$ relative to Π_p is $\boldsymbol{\omega}$, we have

$$\begin{pmatrix} \omega_x \\ \omega_y \\ \omega_z \end{pmatrix} = R^T \boldsymbol{\omega} = (\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})^T \boldsymbol{\omega}. \tag{3}$$

Rolling without slipping imposes a constraint on the angular velocity $\boldsymbol{\omega}$:

$$\omega_z = \boldsymbol{z} \cdot \boldsymbol{\omega} = 0. \tag{4}$$

Also due to pure rolling the relative velocity of Π to Π_p is zero:

$$v_x = v_y = v_z = 0.$$

Therefore the absolute velocity of $\boldsymbol{q}$ equals $\boldsymbol{v}_{\mathcal{P}}$; in other words,

$$R_{po}(\boldsymbol{v}_{\mathcal{B}} + \boldsymbol{\omega}_{\mathcal{B}} \times \boldsymbol{\beta}) = \boldsymbol{v}_{\mathcal{P}}, \tag{5}$$

or equivalently,

$$R_{po}(\boldsymbol{v} + \boldsymbol{\omega} \times \boldsymbol{\beta}) \quad = \quad \boldsymbol{v}_{\mathcal{P}}. \tag{6}$$

The local geometry of $\mathcal{B}$ and $\mathcal{P}$ at the contact plays a major role in the contact kinematics. We will utilize Montana's kinematic equations of contact [21] which relates the contact motions to the relative motion of the two frames (in this case Π and Π_p) at the contact. First of all, we compute the shape operator, geodesic curvatures, and metric of $\mathcal{B}$ as

$$\begin{aligned} S &= (\boldsymbol{x}, \boldsymbol{y})^T(-\nabla_x \boldsymbol{z}, -\nabla_y \boldsymbol{z}) \\ &= \begin{pmatrix} \kappa_1 & 0 \\ 0 & \kappa_2 \end{pmatrix}, \\ K_g &= \boldsymbol{y}^T(\nabla_x \boldsymbol{x}, \nabla_y \boldsymbol{x}) \\ &= (\kappa_{g_1}, \kappa_{g_2}), \\ V &= \begin{pmatrix} \|\boldsymbol{\beta}_{s_1}\| & 0 \\ 0 & \|\boldsymbol{\beta}_{s_2}\| \end{pmatrix}, \end{aligned}$$

and those corresponding invariants of $\mathcal{P}$ as

$$\begin{aligned} S_{\mathcal{P}} &= \mathbf{0}, \\ K_{g_{\mathcal{P}}} &= \mathbf{0}, \\ V_{\mathcal{P}} &= I_2, \end{aligned}$$

where κ_1 and κ_2 are the principal curvatures of $\boldsymbol{\beta}$ at $\boldsymbol{q}$, κ_{g_1} and κ_{g_2} the geodesic curvatures of the principal curves at $\boldsymbol{q}$, and I_2 the 2×2 identity matrix. The kinematics of rolling follow from the above forms, (4), and Montana's equations:

$$\dot{\boldsymbol{u}} = R_\psi S^{-1}(\boldsymbol{y}, -\boldsymbol{x})^T \boldsymbol{\omega}; \tag{7}$$

$$\dot{\boldsymbol{s}} = V^{-1} S^{-1}(\boldsymbol{y}, -\boldsymbol{x})^T \boldsymbol{\omega}; \tag{8}$$

$$\dot{\psi} = K_g S^{-1}(\boldsymbol{y}, -\boldsymbol{x})^T \boldsymbol{\omega}. \tag{9}$$

Since the patch $\boldsymbol{\beta}$ has Gaussian curvature $K = \det S \neq 0$, the shape operator S is invertible.

Equations (7), (8), and (9) shall be better understood in the following geometric way. Let ρ_1 and ρ_2 be the radii of curvature of the normal sections in the principal directions $\boldsymbol{x}$ and $\boldsymbol{y}$ at the contact $\boldsymbol{q}$, respectively. From the convexity of $\boldsymbol{\beta}$ we have $\rho_1 = -\frac{1}{\kappa_1}$ and $\rho_2 = -\frac{1}{\kappa_2}$. For simplicity, let us assume the principal curves to be unit-speed; that is, $\|\boldsymbol{\beta}_{s_1}\| = \|\boldsymbol{\beta}_{s_2}\| = 1$. Hence equation (8) reduces to

$$\begin{aligned} \dot{\boldsymbol{s}} &= I_2 \begin{pmatrix} \rho_1 & 0 \\ 0 & \rho_2 \end{pmatrix} \begin{pmatrix} -\omega_y \\ \omega_x \end{pmatrix} \\ &= \begin{pmatrix} -\rho_1 \omega_y \\ \rho_2 \omega_x \end{pmatrix}. \end{aligned}$$

Figure 3 shows the normal sections of $\boldsymbol{\beta}$ in the $\boldsymbol{x}$ and $\boldsymbol{y}$ directions. Since $\boldsymbol{q}$ has zero relative velocity to $\boldsymbol{p}$, the angular velocity component ω_y generates a contact velocity component of $-\omega_y \rho_1$ along the s_1-parameter curve at $\boldsymbol{q}$, while the component ω_x generates a contact velocity component of $\omega_x \rho_2$ along the s_2-parameter curve at $\boldsymbol{q}$. Under rolling, the velocity $\dot{\boldsymbol{u}}$ is related to the velocity $\dot{\boldsymbol{s}}$ by matrix R_ψ. Equation (9), now written as,

$$\dot{\psi} = (\kappa_{gx}, \kappa_{gy})\dot{\boldsymbol{s}},$$

has a similar explanation from the definition of geodesic curvature.

2.2 Dynamics of Rolling

To apply Newton's second law, we need to use a fixed reference frame with respect to which the velocity and angular velocity can be properly differentiated. Let the reference frame be the fixed frame that instantaneously coincides with frame $\Pi_{\mathcal{B}}$. Therefore we shall use the absolute velocities $\dot{\boldsymbol{v}}_{\mathcal{B}}$ and $\dot{\boldsymbol{\omega}}_{\mathcal{B}}$ in this fixed frame.

Although $\boldsymbol{v}$ and $\boldsymbol{\omega}$ are relative velocities in terms of the moving frame $\Pi_{\mathcal{B}}$, it is not difficult to derive that [1]

$$\dot{\boldsymbol{v}}_{\mathcal{B}} = \boldsymbol{\omega} \times \boldsymbol{v} + \dot{\boldsymbol{v}}; \tag{10}$$

$$\dot{\boldsymbol{\omega}}_{\mathcal{B}} = \boldsymbol{\omega} \times \boldsymbol{\omega} + \dot{\boldsymbol{\omega}} = \dot{\boldsymbol{\omega}}. \tag{11}$$

Let F be the contact force on object $\mathcal{B}$. The dynamics of the object obey Newton's and Euler's equations:

$$\begin{aligned} F + mg\boldsymbol{z} &= m\dot{\boldsymbol{v}}_{\mathcal{B}} = m(\dot{\boldsymbol{v}} + \boldsymbol{\omega} \times \boldsymbol{v}); \\ \boldsymbol{\beta}(\boldsymbol{s}) \times F &= I\dot{\boldsymbol{\omega}}_{\mathcal{B}} + \boldsymbol{\omega}_{\mathcal{B}} \times I\boldsymbol{\omega}_{\mathcal{B}} = I\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times I\boldsymbol{\omega}. \end{aligned}$$

[1] The reader may either try a derivation himself or refer to MacMillan [20, pp. 175-176].

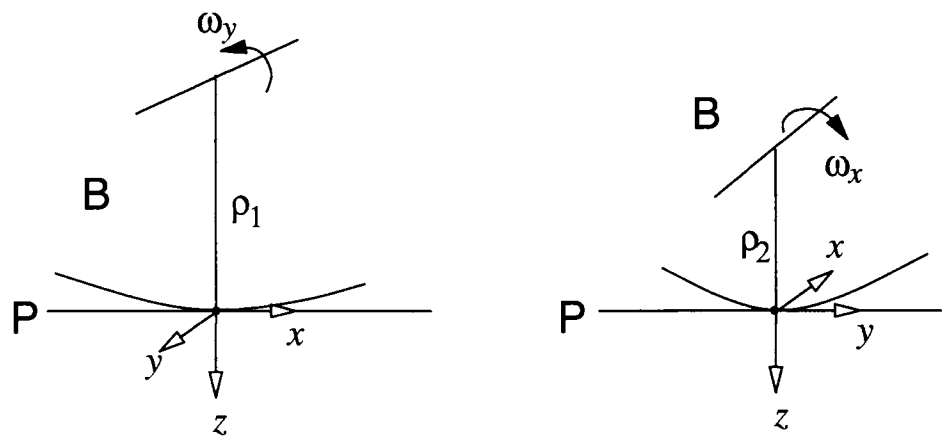


Figure 3: *The normal sections of a pure rolling object $\mathcal{B}$'s surface in the principal directions x and y at the contact. The radii of curvature are ρ_1 and ρ_2, respectively. The angular velocities at the centers of curvature about y and x are ω_y and ω_x, respectively. It is clear that the velocities of contact in x and y are $-\rho_1\omega_y$ and $\rho_2\omega_x$, respectively.*

Immediately from the above equations we can eliminate the contact force F which may be anywhere inside the contact friction cone:

$$\boldsymbol{\beta} \times m(\dot{\boldsymbol{v}} + \boldsymbol{\omega} \times \boldsymbol{v} - g\boldsymbol{z}) \quad = \quad I\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times I\boldsymbol{\omega}. \quad (12)$$

Next, differentiate the kinematic constraint (5) and plug (10) and (11) in:

$$\begin{aligned}(R_{po}\boldsymbol{\omega}) \times R_{po}(\boldsymbol{v} + \boldsymbol{\omega} \times \boldsymbol{\beta}) + R_{po}\Big(\dot{\boldsymbol{v}} + \boldsymbol{\omega} \times \boldsymbol{v} \\ + \dot{\boldsymbol{\omega}} \times \boldsymbol{\beta} + \boldsymbol{\omega} \times (\boldsymbol{\omega} \times \boldsymbol{\beta} + \dot{\boldsymbol{\beta}})\Big) \quad = \quad \boldsymbol{a}_{\mathcal{P}},\end{aligned}$$

where $\boldsymbol{a}_{\mathcal{P}}$ is the acceleration of the plane $\mathcal{P}$. Multiplying both sides of the above equation with R_{po}^T:

$$2\boldsymbol{\omega} \times (\boldsymbol{v} + \boldsymbol{\omega} \times \boldsymbol{\beta}) + \dot{\boldsymbol{v}} + \dot{\boldsymbol{\omega}} \times \boldsymbol{\beta} + \boldsymbol{\omega} \times \dot{\boldsymbol{\beta}} = R_{po}^T \boldsymbol{a}_{\mathcal{P}}. \quad (13)$$

From (12) and (13) we obtain

$$\begin{aligned}\dot{\boldsymbol{\omega}} \quad = \quad D^{-1}\Big(\boldsymbol{\beta} \times R_{po}^T \boldsymbol{a}_{\mathcal{P}} - \boldsymbol{\beta} \times \Big(g\boldsymbol{z} + \boldsymbol{\omega} \times \boldsymbol{v} \\ + 2\boldsymbol{\omega} \times (\boldsymbol{\omega} \times \boldsymbol{\beta}) + \boldsymbol{\omega} \times \dot{\boldsymbol{\beta}}\Big) - \boldsymbol{\omega} \times \frac{I}{m}\boldsymbol{\omega}\Big), \quad (14)\end{aligned}$$

where

$$\begin{aligned} D &= \frac{I}{m} + \|\boldsymbol{\beta}\|^2 I_3 - \boldsymbol{\beta}\boldsymbol{\beta}^T, \qquad (15) \\ \boldsymbol{v} &= R_{po}^T \boldsymbol{v}_{\mathcal{P}} - \boldsymbol{\omega} \times \boldsymbol{\beta}, \qquad \text{from (6).}\end{aligned}$$

In the above I_3 denotes the 3×3 identity matrix. The matrix D is positive definite, following that the angular inertia matrix I is positive definite. More specifically, for any 3×1 vector $\boldsymbol{e} \neq \boldsymbol{0}$, we have

$$\begin{aligned}\boldsymbol{e}^T D \boldsymbol{e} &= \boldsymbol{e}^T\Big(\frac{I}{m} + \|\boldsymbol{\beta}\|^2 I_3 - \boldsymbol{\beta}\boldsymbol{\beta}^T\Big)\boldsymbol{e} \\ &= \boldsymbol{e}^T \frac{I}{m}\boldsymbol{e} + \|\boldsymbol{\beta}\|^2\|\boldsymbol{e}\|^2 - (\boldsymbol{e} \cdot \boldsymbol{\beta})^2 \\ &> 0\end{aligned}$$

Theorem 1 *The nonlinear system consisting of equations*

$$\begin{aligned}\dot{\boldsymbol{u}} &= R_\psi S^{-1}(\boldsymbol{y}, -\boldsymbol{x})^T \boldsymbol{\omega}, \\ \dot{\boldsymbol{s}} &= V^{-1} S^{-1}(\boldsymbol{y}, -\boldsymbol{x})^T \boldsymbol{\omega}, \\ \dot{\psi} &= K_g S^{-1}(\boldsymbol{y}, -\boldsymbol{x})^T \boldsymbol{\omega}, \qquad (16) \\ \dot{\boldsymbol{\omega}} &= D^{-1}\boldsymbol{\beta} \times R_{po}^T \boldsymbol{a}_{\mathcal{P}} - D^{-1}\Big(\boldsymbol{\beta} \times \Big(g\boldsymbol{z} + \boldsymbol{\omega} \times \boldsymbol{v} \\ &\quad + 2\boldsymbol{\omega} \times (\boldsymbol{\omega} \times \boldsymbol{\beta}) + \boldsymbol{\omega} \times \dot{\boldsymbol{\beta}}\Big) + \boldsymbol{\omega} \times \frac{I}{m}\boldsymbol{\omega}\Big),\end{aligned}$$

governs the object motion as well as the contact motions on the object and in the plane.

Both the rolling constraints (4) and (6) were used in deriving system (16) and are therefore implicitly included in the system. Nevertheless, $\boldsymbol{v}$ is a redundant state variable of the system as it depends on $\boldsymbol{\omega}$ and $\boldsymbol{s}$; and (4) induces another redundancy among ω_1, ω_2, and ω_3.

A state of the nonlinear system (16) has eight (scalar) variables including $\boldsymbol{u} = (u_1, u_2)^T$, $\boldsymbol{s} = (s_1, s_2)^T$, ψ, and $\boldsymbol{\omega} = (\omega_1, \omega_2, \omega_3)^T$. Together they determine the object's position and orientation relative to frame $\Pi_{\mathcal{P}}$. Thus given the plane motion, they determine the object's motion relative to frame $\Pi_{\mathcal{B}}$. The one redundancy among ω_1, ω_2, ω_3 will be taken care of when we investigate the local observability of system (16) in the next section. Note there is no need to treat $\boldsymbol{v}_{\mathcal{P}}$ as a state variable since it is known.

The output of the system is $\boldsymbol{u} = (u_1, u_2)^T$. The input of the system is $\boldsymbol{a}_{\mathcal{P}} = (a_{\mathcal{P}_1}, a_{\mathcal{P}_2}, a_{\mathcal{P}_3})^T$ whose components control the input fields

$$\boldsymbol{h}_1 = \begin{pmatrix} \boldsymbol{0} \\ \tilde{\boldsymbol{h}}_1 \end{pmatrix}, \ \boldsymbol{h}_2 = \begin{pmatrix} \boldsymbol{0} \\ \tilde{\boldsymbol{h}}_2 \end{pmatrix}, \ \boldsymbol{h}_3 = \begin{pmatrix} \boldsymbol{0} \\ \tilde{\boldsymbol{h}}_3 \end{pmatrix},$$

respectively, where the vector fields $\tilde{\boldsymbol{h}}_1, \tilde{\boldsymbol{h}}_2, \tilde{\boldsymbol{h}}_3$ together define a 3×3 matrix:

$$\begin{aligned} (\tilde{\boldsymbol{h}}_1, \tilde{\boldsymbol{h}}_2, \tilde{\boldsymbol{h}}_3) &= D^{-1}\boldsymbol{\beta} \times R_{po}^T \\ &= D^{-1}\boldsymbol{\beta} \times \Big((\boldsymbol{x}, \boldsymbol{y})R_\psi, -\boldsymbol{z}\Big), \quad (17) \end{aligned}$$

Denote by $\boldsymbol{f}$ the drift field of the system, which is composed of the right sides of the equations (16) when $\boldsymbol{a}_{\mathcal{P}} = 0$. It measures the rate of state change under zero input.

3 Local Observability

The unknowns of system (16) include the point of contact $\boldsymbol{u}$ in the plane, the point of contact on the object as determined by $\boldsymbol{s}$, the rotation ψ of the object about the contact normal, and the object's angular velocity $\boldsymbol{\omega}$. These variables constitute a state in the system's state space manifold M. This section will look into whether knowing $\boldsymbol{u}$ is sufficient for locally determining $\boldsymbol{s}, \psi$, and $\boldsymbol{\omega}$.

Our approach is to investigate the local observability of system (16). Denote by ξ the current state of rolling so that $\xi = (\boldsymbol{u}, \boldsymbol{s}, \psi, \boldsymbol{\omega})^T$. Essentially, we need to determine if the observability codistribution $d\mathcal{O}$ at ξ equals the cotangent space $T_\xi^* M$ spanned by the differentials $d\boldsymbol{u}, d\boldsymbol{s}, d\psi, d\boldsymbol{\omega}$ at ξ. The codistribution $d\mathcal{O}$ consists of the differentials of u_1, u_2, and their higher order Lie derivatives with respect to $\boldsymbol{f}$, $\boldsymbol{h}_1$, $\boldsymbol{h}_2$, $\boldsymbol{h}_3$. The linear space $\mathcal{O}$ spanned by these functions is called the observation space of the system.

We decompose the cotangent space $T_\xi^* M$ at state ξ into orthogonal subspaces $T_\xi^* M_u$, $T_\xi^* M_s$, $T_\xi^* M_\psi$, $T_\xi^* M_\omega$, with bases $d\boldsymbol{u}, d\boldsymbol{s}, d\psi, d\boldsymbol{\omega}$, respectively. Obviously, $T_\xi^* M_u$ is spanned since $d\boldsymbol{u} \in d\mathcal{O}$. First, we shall show that the cotangent subspace $T_\xi^* M_\omega$ is spanned. Second, we shall derive a sufficient condition about the contact geometry that guarantees the cotangent subspace $T_\xi^* M_s$ to be spanned. Third, we shall extend this sufficient condition for the spanning of $T_\xi^* M_s \times T_\xi^* M_\psi$. Finally, we shall argue that the differentials chosen from the observability codistribution $d\mathcal{O}$ to span these subspaces will span $T_\xi^* M$ under the combined sufficient conditions.

3.1 Angular velocity

As noted before, one of the angular velocity components $\omega_1, \omega_2, \omega_3$ is a redundant state variable under the rolling constraint (4). Let the coordinates of $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$ in frame $\Pi_{\mathcal{B}}$ be (x_1, x_2, x_3), (y_1, y_2, y_3), (z_1, z_2, z_3), respectively. Without loss of generality, we assume $z_3 \neq 0$ in some neighborhood of the state. Immediately from (4)

$$\omega_3 = -\frac{z_1\omega_1 + z_2\omega_2}{z_3}. \quad (18)$$

The new system rewritten from system (16) using (18) has seven state variables $\boldsymbol{u}, \boldsymbol{s}, \psi, \omega_1, \omega_2$. The new observation space is equal to the old one except that every appearance of ω_3 is now replaced with (18). For this reason, we will not distinguish the new observation space from $\mathcal{O}$ unless taking the partial derivatives of functions in $\mathcal{O}$ with respect to ω_1 and ω_2, on which ω_3 has become dependent.

The cotangent subspace $T_\xi^* M_\omega$ is two-dimensional and spanned by $d\omega_1(\xi)$ and $d\omega_2(\xi)$. To show the spanning of $T_\xi^* M_\omega$ by differentials in the codistribution $d\mathcal{O}$, it suffices to prove that the Jacobian matrix $\partial L_f \boldsymbol{u}/\partial(\omega_1, \omega_2)$ has rank 2, where $L_f \boldsymbol{u} = (L_f u_1, L_f u_2)^T$.

This Jacobian is given as

$$\frac{\partial L_f \boldsymbol{u}}{\partial(\omega_1,\omega_2)} = \frac{\partial L_f \boldsymbol{u}}{\partial \boldsymbol{\omega}} \frac{\partial \boldsymbol{\omega}}{\partial(\omega_1,\omega_2)} = \frac{\partial \dot{\boldsymbol{u}}}{\partial \boldsymbol{\omega}} \frac{\partial \boldsymbol{\omega}}{\partial(\omega_1,\omega_2)}$$

$$= R_\psi S^{-1} \begin{pmatrix} y_1 & y_2 & y_3 \\ -x_1 & -x_2 & -x_3 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ -\frac{z_1}{z_3} & -\frac{z_2}{z_3} \end{pmatrix}$$

$$= R_\psi S^{-1} \begin{pmatrix} -\frac{x_2}{z_3} & \frac{x_1}{z_3} \\ -\frac{y_2}{z_3} & \frac{y_1}{z_3} \end{pmatrix}.$$

Computing the determinant of the Jacobian is straightforward:

$$\det\left(\frac{\partial L_f \boldsymbol{u}}{\partial(\omega_1,\omega_2)}\right) = -\frac{1}{K z_3},$$

where $K \neq 0$ is the Gaussian curvature at the contact point $\boldsymbol{q}$.

Proposition 2 *The cotangent subspace $T^*_\xi M_\omega$ is spanned by the differentials $dL_f \boldsymbol{u}$.*

This lemma states that rolling induces a one-to-one mapping from the space of angular velocities to the space of contact velocities in the plane for constant $\boldsymbol{s}$ and ψ. Accordingly, there exists some control that can distinguishes between any two different (but close enough) angular velocities of the object in the same pose.

3.2 Contact on the object

We now proceed to determine whether the cotangent subspace $T^*_\xi M_s$ is spanned by the differentials in $d\mathcal{O}$. This time we choose from $\mathcal{O}$ six (scalar) functions $L_{h_1} L_f \boldsymbol{u}$, $L_{h_2} L_f \boldsymbol{u}$, $L_{h_3} L_f \boldsymbol{u}$ comprising the following 2×3 matrix:

$$\begin{aligned} Q_\psi &= (L_{h_1} L_f \boldsymbol{u},\ L_{h_2} L_f \boldsymbol{u},\ L_{h_3} L_f \boldsymbol{u}) \\ &= R_\psi S^{-1} (\boldsymbol{y}, -\boldsymbol{x})^T D^{-1} \boldsymbol{\beta} \times \Big((\boldsymbol{x}, \boldsymbol{y}) R_\psi, -\boldsymbol{z}\Big). \end{aligned} \tag{19}$$

The cotangent subspace $T^*_\xi M_s$ is spanned provided the partial derivatives $\partial Q_\psi / \partial s_1$ and $\partial Q_\psi / \partial s_2$, viewed as two 6-dimensional vectors, are linearly independent. Introducing a 2×3 matrix:

$$Q = S^{-1} (\boldsymbol{y}, -\boldsymbol{x})^T D^{-1} \boldsymbol{\beta} \times R, \tag{20}$$

where $R = (\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ as given by (1), we have

Lemma 3 *The partial derivatives $\partial Q_\psi / \partial s_1$ and $\partial Q_\psi / \partial s_2$ are linearly dependent if and only if the partial derivatives $\partial Q / \partial s_1$ and $\partial Q / \partial s_2$ are linearly dependent.*

Proof: Suppose there exist c_1 and c_2, not both zero, such that

$$c_1 \frac{\partial Q_\psi}{\partial s_1} + c_2 \frac{\partial Q_\psi}{\partial s_2} = \mathbf{0}.$$

Expand this equation into two:

$$\begin{aligned} & c_1 R_\psi \frac{\partial}{\partial s_1} \Big(S^{-1} (\boldsymbol{y}, -\boldsymbol{x})^T D^{-1} \boldsymbol{\beta} \times (\boldsymbol{x}, \boldsymbol{y}) \Big) R_\psi \\ & \quad + c_2 R_\psi \frac{\partial}{\partial s_2} \Big(S^{-1} (\boldsymbol{y}, -\boldsymbol{x})^T D^{-1} \boldsymbol{\beta} \times (\boldsymbol{x}, \boldsymbol{y}) \Big) R_\psi = \mathbf{0}; \end{aligned} \tag{21}$$

$$\begin{aligned} & c_1 R_\psi \frac{\partial}{\partial s_1} \Big(S^{-1} (\boldsymbol{y}, -\boldsymbol{x})^T D^{-1} \boldsymbol{\beta} \times (-\boldsymbol{z}) \Big) \\ & \quad + c_2 R_\psi \frac{\partial}{\partial s_2} \Big(S^{-1} (\boldsymbol{y}, -\boldsymbol{x})^T D^{-1} \boldsymbol{\beta} \times (-\boldsymbol{z}) \Big) = \mathbf{0}. \end{aligned} \tag{22}$$

Multiply (21) by R_ψ on both the left and the right and (22) by $-R_\psi$ on the left; and merge the two resulting equations into one:

$$\begin{aligned} & c_1 \frac{\partial}{\partial s_1} \Big(S^{-1} (\boldsymbol{y}, -\boldsymbol{x})^T D^{-1} \boldsymbol{\beta} \times (\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) \Big) \\ & \quad + c_2 \frac{\partial}{\partial s_2} \Big(S^{-1} (\boldsymbol{y}, -\boldsymbol{x})^T D^{-1} \boldsymbol{\beta} \times (\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) \Big) = \mathbf{0}. \end{aligned}$$

Hence $\partial Q / \partial s_1$ and $\partial Q / \partial s_2$ are linearly dependent.

Conversely. □

Thus we will focus our investigation on the linear independence of the partial derivatives $\partial Q / \partial s_1$ and $\partial Q / \partial s_2$. Unfortunately, for certain shape $\boldsymbol{\beta}$, these two partial derivatives are linearly dependent. For example, if $\boldsymbol{\beta}$ is a sphere with radius r, one can verify that[2]

$$Q = \begin{pmatrix} -\frac{5}{7} & 0 & 0 \\ 0 & -\frac{5}{7} & 0 \end{pmatrix}.$$

The partial derivatives of Q are zero and obviously linearly dependent. In this example, the contact point can

[2] Use $\beta = r(\cos s_1 \cos s_2, -\cos s_1 \sin s_2, \sin s_1)^T$ as the parametrization, for instance.

be anywhere on the sphere whatever path the sphere rolls along in the supporting plane.

But objects of more general shape are to our real interest. So we aim at establishing some condition on the linear independence of $\partial Q/\partial s_1$ and $\partial Q/\partial s_2$ that can be satisfied by most shapes.

To compute these two partial derivatives, we first obtain the partial derivatives of $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$:

$$\frac{\partial}{\partial s_1}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}) = \|\boldsymbol{\beta}_{s_1}\|\,(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z})\begin{pmatrix} 0 & -\kappa_{g_1} & -\kappa_1 \\ \kappa_{g_1} & 0 & 0 \\ \kappa_1 & 0 & 0 \end{pmatrix};$$

$$\frac{\partial}{\partial s_2}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}) = \|\boldsymbol{\beta}_{s_2}\|\,(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z})\begin{pmatrix} 0 & -\kappa_{g_2} & 0 \\ \kappa_{g_2} & 0 & -\kappa_2 \\ 0 & \kappa_2 & 0 \end{pmatrix}.$$

Let $A = D^{-1}\boldsymbol{\beta}\times$, where

$$\boldsymbol{\beta}\times \;=\; \begin{pmatrix} 0 & -\boldsymbol{\beta}_3 & \boldsymbol{\beta}_2 \\ \boldsymbol{\beta}_3 & 0 & -\boldsymbol{\beta}_1 \\ -\boldsymbol{\beta}_2 & \boldsymbol{\beta}_1 & 0 \end{pmatrix}$$

with $\boldsymbol{\beta} = (\boldsymbol{\beta}_1, \boldsymbol{\beta}_2, \boldsymbol{\beta}_3)^T$ in frame Π_B is the skew-symmetric matrix representing cross product. Then $DA = \boldsymbol{\beta}\times$. Differentiate this equation with respect to s_1 (s_2, respectively) on both sides and solve for the partial derivatives of A:

$$\begin{aligned} \frac{\partial}{\partial s_1}(D^{-1}\boldsymbol{\beta}\times) &= D^{-1}\left(\|\boldsymbol{\beta}_{s_1}\|\boldsymbol{x}\times - \frac{\partial D}{\partial s_1}D^{-1}\boldsymbol{\beta}\times\right); \\ \frac{\partial}{\partial s_2}(D^{-1}\boldsymbol{\beta}\times) &= D^{-1}\left(\|\boldsymbol{\beta}_{s_2}\|\boldsymbol{y}\times - \frac{\partial D}{\partial s_2}D^{-1}\boldsymbol{\beta}\times\right). \end{aligned}$$

With all the above partial derivatives we can write out the partial derivatives of Q:

$$\begin{aligned} \frac{\partial Q}{\partial s_1} = {} & \Big(\frac{\partial S^{-1}}{\partial s_1}(\boldsymbol{y},-\boldsymbol{x})^T - \|\boldsymbol{\beta}_{s_1}\|S^{-1}(\kappa_{g_1}\boldsymbol{x},\ \kappa_{g_1}\boldsymbol{y} \\ & + \kappa_1\boldsymbol{z})^T\Big)D^{-1}\boldsymbol{\beta}\times R + S^{-1}(\boldsymbol{y},-\boldsymbol{x})^T D^{-1} \\ & \Big(-\frac{\partial D}{\partial s_1}D^{-1}\boldsymbol{\beta}\times R + \|\boldsymbol{\beta}_{s_1}\|\Big((0,\boldsymbol{z},-\boldsymbol{y}) \\ & + \boldsymbol{\beta}\times(\kappa_{g_1}\boldsymbol{y}+\kappa_1\boldsymbol{z},\ -\kappa_{g_1}\boldsymbol{x},\ -\kappa_1\boldsymbol{x})\Big)\Big); \quad (23) \end{aligned}$$

$$\begin{aligned} \frac{\partial Q}{\partial s_2} = {} & \Big(\frac{\partial S^{-1}}{\partial s_2}(\boldsymbol{y},-\boldsymbol{x})^T - \|\boldsymbol{\beta}_{s_2}\|S^{-1}(\kappa_{g_2}\boldsymbol{x} - \kappa_2\boldsymbol{z}, \\ & \kappa_{g_2}\boldsymbol{y})^T\Big)D^{-1}\boldsymbol{\beta}\times R + S^{-1}(\boldsymbol{y},-\boldsymbol{x})^T D^{-1} \\ & \Big(-\frac{\partial D}{\partial s_2}D^{-1}\boldsymbol{\beta}\times R + \|\boldsymbol{\beta}_{s_2}\|\Big((-\boldsymbol{z},0,\boldsymbol{x}) \\ & + \boldsymbol{\beta}\times(\kappa_{g_2}\boldsymbol{y},\ -\kappa_{g_2}\boldsymbol{x}+\kappa_2\boldsymbol{z},\ -\kappa_2\boldsymbol{y})\Big)\Big). \quad (24) \end{aligned}$$

It seems very difficult to say anything about the linear independence of $\partial Q/\partial s_1$ and $\partial Q/\partial s_2$ directly from their complicated forms above. So we intend to find a matrix A and a sufficient condition under which $(\partial Q/\partial s_1)A$ and $(\partial Q/\partial s_2)A$ will be linearly independent. The same condition will then ensure the linear independence between $\partial Q/\partial s_1$ and $\partial Q/\partial s_2$. The matrix of our choice is $R^T\boldsymbol{\beta}$, which, when multiplied on (23) and (24) to the right, gets rid of those terms in $\partial Q/\partial s_1$ and $\partial Q/\partial s_2$ that have $\boldsymbol{\beta}\times R$ on the right. We can further simplify the remaining terms in the products using the following equations:

$$\begin{aligned} (0,\boldsymbol{z},-\boldsymbol{y})R^T &= \boldsymbol{x}\times; \\ (-\boldsymbol{z},0,\boldsymbol{x})R^T &= \boldsymbol{y}\times; \\ (\kappa_{g_1}\boldsymbol{y}+\kappa_1\boldsymbol{z},-\kappa_{g_1}\boldsymbol{x},-\kappa_1\boldsymbol{x})R^T &= \kappa_{g_1}\boldsymbol{z}\times - \kappa_1\boldsymbol{y}\times; \\ (\kappa_{g_2}\boldsymbol{y},-\kappa_{g_2}\boldsymbol{x}+\kappa_2\boldsymbol{z},-\kappa_2\boldsymbol{y})R^T &= \kappa_{g_2}\boldsymbol{z}\times + \kappa_2\boldsymbol{x}\times. \end{aligned}$$

In the end, we obtain

$$\frac{\partial Q}{\partial s_1}R^T\boldsymbol{\beta} = \|\boldsymbol{\beta}_{s_1}\|S^{-1}(\boldsymbol{y},-\boldsymbol{x})^T D^{-1}\boldsymbol{\beta}\times\boldsymbol{b}_1; \quad (25)$$

$$\frac{\partial Q}{\partial s_2}R^T\boldsymbol{\beta} = \|\boldsymbol{\beta}_{s_2}\|S^{-1}(\boldsymbol{y},-\boldsymbol{x})^T D^{-1}\boldsymbol{\beta}\times\boldsymbol{b}_2, \quad (26)$$

where

$$\boldsymbol{b}_1 = -\boldsymbol{x} + (\kappa_{g_1}\boldsymbol{z} - \kappa_1\boldsymbol{y})\times\boldsymbol{\beta}; \quad (27)$$

$$\boldsymbol{b}_2 = -\boldsymbol{y} + (\kappa_{g_2}\boldsymbol{z} + \kappa_2\boldsymbol{x})\times\boldsymbol{\beta}. \quad (28)$$

Lemma 4 *The vectors $(\partial Q/\partial s_1)R^T\boldsymbol{\beta}$, $(\partial Q/\partial s_2)R^T\boldsymbol{\beta}$ are linearly dependent if and only if $\boldsymbol{\beta}\cdot(I\boldsymbol{z}) = 0$ or* $\det(\boldsymbol{b}_1,\boldsymbol{b}_2,\boldsymbol{\beta}) = 0$.

Proof: We need to conduct a few steps of reasoning:

$$\frac{\partial Q}{\partial s_1}R^T\boldsymbol{\beta} \quad\text{and}\quad \frac{\partial Q}{\partial s_2}R^T\boldsymbol{\beta} \quad\text{are linearly dependent}$$

if and only if

$$(\boldsymbol{y}, -\boldsymbol{x})^T D^{-1}\boldsymbol{\beta} \times \boldsymbol{b}_1 \quad \text{and} \quad (\boldsymbol{y}, -\boldsymbol{x})^T D^{-1}\boldsymbol{\beta} \times \boldsymbol{b}_2$$

are linearly dependent

if and only if

$D^{-1}\boldsymbol{\beta} \times \boldsymbol{b}_1$, $D^{-1}\boldsymbol{\beta} \times \boldsymbol{b}_2$, and $\boldsymbol{z}$ are linearly dependent

if and only if

$\boldsymbol{\beta} \times \boldsymbol{b}_1, \boldsymbol{\beta} \times \boldsymbol{b}_2$, and $D\boldsymbol{z}$ are linearly dependent,

where

$$\begin{aligned} D\boldsymbol{z} &= \left(\frac{I}{m} + \|\boldsymbol{\beta}\|^2 I_3 - \boldsymbol{\beta}\boldsymbol{\beta}^T\right)\boldsymbol{z} \\ &= \frac{I}{m}\boldsymbol{z} + \boldsymbol{\beta} \times (\boldsymbol{z} \times \boldsymbol{\beta}). \end{aligned}$$

There are two cases: (1) $\boldsymbol{\beta}\cdot(I\boldsymbol{z}) = 0$ and (2) $\boldsymbol{\beta}\cdot(I\boldsymbol{z}) \neq 0$. In case (1), $\boldsymbol{\beta}\times\boldsymbol{b}_1, \boldsymbol{\beta}\times\boldsymbol{b}_2, D\boldsymbol{z}$ are linearly dependent. In case (2), they are linearly dependent if and only if $\boldsymbol{\beta}\times\boldsymbol{b}_1$ and $\boldsymbol{\beta}\times\boldsymbol{b}_2$ are linearly dependent, which happens if and only if $\det(\boldsymbol{b}_1, \boldsymbol{b}_2, \boldsymbol{\beta}) = 0$ □

Combining Lemmas 3 and 4, we arrive at a sufficient condition for the spanning of $T^*_\xi M_s$.

Proposition 5 *The cotangent subspace $T^*_\xi M_s$ is spanned by the differentials of six functions $L_{h_1}L_f\boldsymbol{u}$, $L_{h_2}L_f\boldsymbol{u}$, $L_{h_3}L_f\boldsymbol{u}$, all in the observation space of system (16), if (1) $\boldsymbol{\beta}\cdot(I\boldsymbol{z}) \neq 0$ and (2) $\det(\boldsymbol{b}_1, \boldsymbol{b}_2, \boldsymbol{\beta}) \neq 0$.*

When $\boldsymbol{\beta}$ is a sphere, $\boldsymbol{b}_1 = \boldsymbol{b}_2 = \boldsymbol{0}$, violating condition (2) in Proposition 5. For general $\boldsymbol{\beta}$, conditions (1) and (2) in Proposition 5 are satisfied at all but at most a one-dimensional set of contact points.

Up until now we have selected from the observation space $\mathcal{O}$ a total of 10 scalar functions: $\boldsymbol{u}$, $L_f\boldsymbol{u}$, $L_{h_1}L_f\boldsymbol{u}$, $L_{h_2}L_f\boldsymbol{u}$, and $L_{h_3}L_f\boldsymbol{u}$. The first two functions depend on only $\boldsymbol{u}$, while the last six functions on only $\boldsymbol{s}$ and ψ. Hence we have shown that the cotangent subspace $T^*_\xi M_u \times T^*_\xi M_s \times T^*_\xi M_\omega$ is spanned unless $\boldsymbol{\beta}\cdot(I\boldsymbol{z}) = 0$ or $\det(\boldsymbol{b}_1, \boldsymbol{b}_2, \boldsymbol{\beta}) = 0$.

3.3 Rotation about the contact normal

The rolling object has three degrees of freedom: the point of contact on the object $\boldsymbol{s}$ and the rotation ψ of the object about the contact normal. The last proposition states that the contact point alone can be distinguished under a sufficient condition. Such condition essentially ensures the differentials of $L_{h_1}L_f\boldsymbol{u}$, $L_{h_2}L_f\boldsymbol{u}$, and $L_{h_3}L_f\boldsymbol{u}$ to span the cotangent subspace $T^*_\xi M_s$. Now we shall show that these differentials indeed span the cotangent subspace $T^*_\xi M_s \times T^*_\xi M_\psi$ except for special shapes such as a sphere. That is, we shall show that the 2×3 matrices $\partial Q_\psi/\partial s_1$, $\partial Q_\psi/\partial s_2$, and $\partial Q_\psi/\partial\psi$ are linearly independent for general shape $\boldsymbol{\beta}$.

Let Q_{12} be the 2×2 matrix formed by the first two columns in Q:

$$Q_{12} = S^{-1}(\boldsymbol{y}, -\boldsymbol{x})^T D^{-1}\boldsymbol{\beta} \times (\boldsymbol{x}, \boldsymbol{y}).$$

Construct a 2×3 matrix:

$$\tilde{Q} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} Q + Q_{12} \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \end{pmatrix}.$$

Lemma 6 *The partial derivatives $\partial Q_\psi/\partial s_1, \partial Q_\psi/\partial s_2$, $\partial Q_\psi/\partial\psi$ are linearly dependent if and only if $\tilde{Q}$ and the partial derivatives $\partial Q/\partial s_1$, $\partial Q/\partial s_2$ are linearly dependent.*

The proof of Lemma 6 is similar to that of Lemma 3. This lemma, along with Propositions 2 and 5, suggest the weak dependency, if any, of system (16)'s local observability on the object's rotation angle ψ.

In this section, assuming the linear independence of $\partial Q/\partial s_1$ and $\partial Q/\partial s_2$, we want to determine if $\tilde{Q}$ is linearly independent of them. If not always, we would like to seek some sufficient condition on the object shape $\boldsymbol{\beta}$. Let us start by offering a rather restrictive necessary condition for $\tilde{Q}$ to vanish.

Lemma 7 *The matrix $\tilde{Q}$ is equal to $\boldsymbol{0}$ only if the following three conditions all hold:*

$$\begin{aligned} \boldsymbol{\beta} \times \boldsymbol{z} &= \boldsymbol{0}; \\ \kappa_1 \boldsymbol{x}^T D^{-1}\boldsymbol{x} &= \kappa_2 \boldsymbol{y}^T D^{-1}\boldsymbol{y}; \\ \boldsymbol{x}^T D^{-1}\boldsymbol{y} &= 0. \end{aligned} \tag{29}$$

Proof: There are two cases: (1) $\boldsymbol{\beta} \times \boldsymbol{z} \neq \mathbf{0}$, and (2) $\boldsymbol{\beta} \times \boldsymbol{z} = \mathbf{0}$.

In the first case, we have $D^{-1}\boldsymbol{\beta} \times \boldsymbol{z} \neq \mathbf{0}$ due to that D^{-1} is positive definite. The third column of $\tilde{Q}$ is not zero, for we know:

$$\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} S^{-1}(\boldsymbol{y}, -\boldsymbol{x})^T D^{-1}\boldsymbol{\beta} \times \boldsymbol{z} = \mathbf{0}$$

if and only if

$$(\boldsymbol{y}, -\boldsymbol{x})^T D^{-1}\boldsymbol{\beta} \times \boldsymbol{z} = \mathbf{0}$$

if and only if

$$D^{-1}\boldsymbol{\beta} \times \boldsymbol{z} = c\boldsymbol{z}, \qquad \text{for some } c \neq 0$$

if and only if

$$\boldsymbol{\beta} \times \boldsymbol{z} = cD\boldsymbol{z}, \qquad \text{for some } c \neq 0,$$

only if

$$0 = \boldsymbol{z} \cdot (\boldsymbol{\beta} \times \boldsymbol{z}) = \boldsymbol{z}^T D\boldsymbol{z}.$$

But $\boldsymbol{z}^T D\boldsymbol{z} > 0$, as D is positive definite.

In the second case, the third column of Q is zero. Since $\boldsymbol{\beta} = \|\boldsymbol{\beta}\|\boldsymbol{z}$, we have

$$\begin{aligned} Q_{12} &= \|\boldsymbol{\beta}\| S^{-1}(\boldsymbol{y}, -\boldsymbol{x})^T D^{-1}\boldsymbol{z} \times (\boldsymbol{x}, \boldsymbol{y}) \\ &= \|\boldsymbol{\beta}\| S^{-1}(\boldsymbol{y}, -\boldsymbol{x})^T D^{-1}(\boldsymbol{y}, -\boldsymbol{x}). \end{aligned}$$

The 2×2 matrix $(\boldsymbol{y}, -\boldsymbol{x})^T D^{-1}(\boldsymbol{y}, -\boldsymbol{x})$ is positive definite, so we write

$$Q_{12} = \|\boldsymbol{\beta}\| S^{-1} \begin{pmatrix} a & b \\ b & d \end{pmatrix}, \; a > 0 \text{ and } ad - b^2 > 0.$$

Now we are able to simplify the first two columns of $\tilde{Q}$:

$$\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} Q_{12} + Q_{12} \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} = \|\boldsymbol{\beta}\| \begin{pmatrix} -\frac{b}{\kappa_1} - \frac{b}{\kappa_2} & \frac{a}{\kappa_1} - \frac{d}{\kappa_2} \\ \frac{a}{\kappa_1} - \frac{d}{\kappa_2} & \frac{b}{\kappa_1} + \frac{b}{\kappa_2} \end{pmatrix}.$$

The above 2×2 matrix is zero if and only if $b = 0$ and $\frac{a}{\kappa_1} = \frac{d}{\kappa_2}$, which imply conditions (29). □

Suppose the necessary conditions in Lemma 7 does not hold. Then $\tilde{Q} \neq \mathbf{0}$. Suppose again $\partial Q/\partial s_1$ and $\partial Q/\partial s_2$ are linearly independent. The matrices $\partial Q/\partial s_1$, $\partial Q/\partial s_2$, and $\tilde{Q}$ are linearly dependent if and only if there exist c_1, c_2, not all zero, such that

$$\begin{aligned} c_1 \frac{\partial Q}{\partial s_1} + c_2 \frac{\partial Q}{\partial s_2} &= \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} Q \\ &\quad + Q_{12} \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \end{pmatrix}. \end{aligned} \tag{30}$$

Multiply both sides of (30) by $R^T\boldsymbol{\beta}$ on the right, plug (25) and (26) in, and then multiply both sides of the resulting equation by S on the left:

$$\begin{aligned} c_1 \|\boldsymbol{\beta}_{s_1}\| (\boldsymbol{y}, -\boldsymbol{x})^T D^{-1}\boldsymbol{\beta} \times \boldsymbol{b}_1 + c_2 \|\boldsymbol{\beta}_{s_2}\| (\boldsymbol{y}, -\boldsymbol{x})^T D^{-1}\boldsymbol{\beta} \times \boldsymbol{b}_2 \\ = (\boldsymbol{y}, -\boldsymbol{x})^T D^{-1}\boldsymbol{\beta} \times (\boldsymbol{\beta} \times \boldsymbol{z}). \end{aligned}$$

Solving the above equation for c_1 and c_2 and substituting the solutions into (30), we end up with a system of third order partial differential equations of $\boldsymbol{\beta}$, four of which are independent. We postulate that this PDE system has at most one solution.

Proposition 8 *Assume $dL_{h_1}L_f\boldsymbol{u}$, $dL_{h_2}L_f\boldsymbol{u}$, and $dL_{h_3}L_f\boldsymbol{u}$ span the cotangent subspace $T^*_\xi M_s$. Furthermore, assume the necessary condition in Lemma 7 does not hold. The same differentials will also span the cotangent subspace $T^*_\xi M_s \times T^*_\xi M_\psi$ except for at most one (local) shape of $\boldsymbol{\beta}$.*

3.4 Sufficient conditions for local observability

To summarize, we have chosen from the observation space $\mathcal{O}$ a total of ten functions: two scalar functions $\boldsymbol{u}$, which are the system outputs; two first order Lie derivatives $L_f\boldsymbol{u}$, which involve ω_1, ω_2, s_1, s_2, ψ; and six second order Lie derivatives $L_{h_1}L_f\boldsymbol{u}$, $L_{h_2}L_f\boldsymbol{u}$, $L_{h_3}L_f\boldsymbol{u}$, which involve s_1, s_2, ψ only. The differentials of these functions (except $L_f\boldsymbol{u}$) live in orthogonal subspaces of the cotangent space $T^*_\xi M$. If these subspaces are spanned, given that $T^*_\xi M_\omega$ is spanned by $dL_f\boldsymbol{u}$, $T^*_\xi M$ is also spanned and local observability of system (16) follows.

Combining Proposition 2 through Proposition 8, we obtain a sufficient condition for the local observability of rolling:

Theorem 9 *The rolling system (16) is locally observable except for at most one object shape $\boldsymbol{\beta}$ if*

$$\det(\boldsymbol{b}_1, \boldsymbol{b}_2, \boldsymbol{\beta}) \neq 0 \tag{31}$$

and

$$\boldsymbol{\beta} \cdot (I\boldsymbol{z}) \neq 0, \qquad \text{when } \boldsymbol{\beta} \times \boldsymbol{z} \neq \mathbf{0}; \tag{32}$$

$$\kappa_1 \boldsymbol{x}^T D^{-1} \boldsymbol{x} \neq \kappa_2 \boldsymbol{y}^T D^{-1} \boldsymbol{y} \text{ or } \boldsymbol{x}^T D^{-1} \boldsymbol{y} \neq 0, \qquad \text{when } \boldsymbol{\beta} \times \boldsymbol{z} = \mathbf{0}, \tag{33}$$

where $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$ are given by (27) and (28), respectively.[3]

Condition (31) depends on the differential geometry of contact. It is only violated by at most a one-dimensional set of points in a general patch $\boldsymbol{\beta}$. Condition (32) states that the contact normal transformed by the angular inertia matrix should not be perpendicular to the contact location vector. It is only violated by at most a one-dimensional set of points in a general patch. Condition (33), only for $\boldsymbol{\beta} \times \boldsymbol{z} = \mathbf{0}$, can hardly be violated by any shape other than a sphere.

4 Summary

This paper deals with local observability of a three-dimensional object rolling on a translating plane. The object is bounded by a smooth surface that makes point contact with the plane, which is free to accelerate in any direction. The plane is rough enough to allow only the pure rolling motion of the object. As the object rolls, the contact traces out a path in the plane, which can be detected by a tactile array sensor embedded in the plane.

Utilizing Montana's equations for contact kinematics, we described the kinematics and dynamics of this task by a nonlinear system, of which the output is the contact location in the plane. Then we established a sufficient condition for the pose and motion of the object to be locally observable. This was done by decomposing the cotangent space at the current state into orthogonal subspaces associated with the object's pose and angular velocity, respectively, and by later combining the sufficient conditions on the spanning of these subspaces. The combined condition depends only on the object's contact geometry and angular inertia matrix. And it is general enough to be satisfied almost everywhere on any object surface that does not assume certain degeneracies as a sphere does.

From a broader perspective, we expect the paper's result on local observability from contact to generalize to many other manipulation tasks. The presented work is part of our larger goal to understand how much information about a manipulation task is encoded in a small amount of tactile data and to develop methods for retrieving such information.

One direction of future work will be to design a nonlinear observer that can asymptotically estimate the pose and motion of a rolling object from the path of contact. Further along this line of research will be to implement (tactile) sensors and mount them on a palm to estimate poses and motions of real objects.

Another direction is to study the controllability of the rolling object on a palm and to combine the result with the present local observability result for developing parts orienting and dextrous manipulation strategies.

Acknowledgments

Support for this research was provided in part by Carnegie Mellon University, and in part by the National Science Foundation through the following grants: NSF Presidential Young Investigator award IRI-9157643 and NSF Grants IRI-9213993 and IRI-9503648.

References

[1] K.-F. Böhringer, B. R. Donald, R. Mihailovich, and N. C. MacDonald. Sensorless manipulation using massively parallel microfabricated actuator arrays. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pages 826–833, 1994.

[3] Note that $\beta \times z = 0$ implies $\beta \cdot (Iz) \neq 0$ given that I is positive definite.

[2] R. C. Brost and M. T. Mason. Graphical analysis of planar rigid-body dynamics with multiple frictional contacts. In *The Fifth International Symposium on Robotics Research*, pages 293–300, 1990.

[3] C. Cai and B. Roth. On the spatial motion of a rigid body with point contact. In *Proceedings of the 1987 IEEE International Conference on Robotics and Automation*, pages 686–695, 1987.

[4] P. E. Crouch. Dynamical realizations of finite Volterra series. *SIAM Journal on Control and Optimization*, 19(1):177–202, 1981.

[5] M. Erdmann. An exploration of nonprehensile two-palm manipulation: Planning and execution. In *The Seventh International Symposium on Robotics Research*, October 21–25 1995. To appear in *International Journal of Robotics Research*.

[6] M. Erdmann and M. Mason. An exploration of sensorless manipulation. *IEEE Journal of Robotics and Automation*, 4(4):369–379, 1988.

[7] P. C. Gaston and T. Lozano-Pérez. Tactile recognition and localization using object models: The case of polyhedra on a plane. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(3):257–266, 1984.

[8] S. Goyal, A. Ruina, and J. Papadopoulos. Planar sliding with dry friction. *Wear*, 143:307–352, 1991.

[9] W. E. L. Grimson and T. Lozano-Pérez. Model-based recognition and localization from sparse range or tactile data. *International Journal of Robotics Research*, 3(3):3–35, 1984.

[10] D. D. Grossman and M. W. Blasgen. Orienting mechanical parts by computer-controlled manipulator. *IEEE Transaction on Systems, Man, and Cybernetics*, 5(5):561–565, 1975.

[11] R. Hermann and A. J. Krener. Nonlinear controllability and observability. *IEEE Transactions on Automatic Control*, AC-22(5):728–740, 1977.

[12] R. D. Howe and M. R. Cutkosky. Dynamic tactile sensing: Perception of fine surface features with stress rate sensing. *IEEE Transactions on Robotics and Automation*, 9(2):140–151, 1993.

[13] A. Isidori. *Nonlinear Control Systems: An Introduction*. Springer-Verlag, 1985.

[14] Y.-B. Jia and M. Erdmann. Pose from pushing. In *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*, pages 165–171, 1996.

[15] Y.-B. Jia and M. Erdmann. Observing pose and motion through contact. To appear in *Proceedings of the 1998 IEEE International Conference on Robotics and Automation*, 1998.

[16] J. Kerr and B. Roth. Analysis of multifingered hands. *International Journal of Robotics Research*, 4(4):3–17, 1986.

[17] Z. Li and J. Canny. Motion of two rigid bodies with rolling constraint. *IEEE Transactions on Robotics and Automation*, 6(1):62–72, 1990.

[18] D. G. Luenberger. An introduction to observers. *IEEE Transactions on Automatic Control*, AC-16(6), 1971.

[19] K. Lynch, H. Maekawa, and K. Tanie. Manipulation and active sensing by pushing using tactile feedback. In *Proceedings of the 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 416–421, 1992.

[20] W. D. MacMillan. *Dynamics of Rigid Bodies*. McGraw-Hill Inc., 1936. Dover reprint, 1960.

[21] D. J. Montana. The kinematics of contact and grasp. *International Journal of Robotics Research*, 7(3):17–32, 1988.

[22] H. Nijmeijer and A. van der Schaft. *Nonlinear Dynamical Control Systems*. Springer-Verlag, New York, 1990.

[23] J.-S. Pang and J. C. Trinkle. Complementary formulations and existence of solutions of dynamic multi-rigid-body contact problems with Coulomb friction. *Mathematical Programming*, 73:199–226, 1996.

[24] K. Salisbury. Interpretation of contact geometries from force measurements. In M. Brady and R. Paul, editors, *Robotics Research*, pages 565–577. The MIT Press, 1984.

[25] M. Spivak. *A Comprehensive Introduction to Differential Geometry*, volume 3. Publish or Perish, Inc., Boston, 1975.

Taylor & Francis
Taylor & Francis Group
http://taylorandfrancis.com

Geometry for Modeling Biomolecules

Herbert Edelsbrunner, *University of Illinois, Urbana, Illinois, USA*

Space filling diagrams are geometric models of molecular conformations in three-dimensional space. Each atom is a location is space and a quantitative expression of influence on the immediate surrounding. This paper surveys the basic types of space filling diagrams with a focus on the dual alpha shape. Properties of those diagrams that relate to questions of connectivity, size, shape and symmetry, and metamorphosis are discussed.

1 Introduction

The concept of a molecule seems to be an essential and stable ingredient in scientific studies of microscopic events. Still, people's understanding and image of what molecules are has changed with time. Even today the question whether there are atoms can still be asked, Bader [3]. We rephrase and ask whether it is *useful* to postulate the existence of atoms, and if yes, what model of an atom is most productive. This is the utilitarian viewpoint which is adopted in this paper, but the question is directed towards molecules. Since even the utilitarian goal is rather ambitious, we claim without justification that some of the most useful models are geometric. We then go ahead and survey a class of geometric models known as space filling diagrams.

Geometry in biology. There is in fact a large body of evidence in favor of the thesis that geometry plays an important role in life on the microscopic level. One of the most striking insights generated by decades of research is that geometric shape decides how biomolecules function. We quote Rose [39]: "What role a protein takes in the grand biological opera depends on exactly one thing: its shape. For a protein molecule, function follows form." It thus seems imperative to design and fabricate refined tools that permit focussed probes into the geometry of this opera. It is likely that such tools will find use in the design of drugs, in the classification of molecular components, in simulating docking and folding processes, etc.

Purpose and scope. The goal of this paper is a biased survey of space filling diagrams used to model large molecules. The bias follows the author's background and is directed towards topological, geometric, and algorithmic properties of these models. Even within this narrow scope, the reader will find a bias towards the author's own work, for which the author apologizes.

The methods used and discussed in this paper come primarily from combinatorial disciplines within topology, geometry, and algorithms. Among the many textbooks in these areas we recommend the following: the short monograph by Alexandrov [2] for an easy and intuitive introduction to topology, the text by Guillemin and Pollack [24] for an introduction to differential ideas in topology, the two-volume handbook edited by Gruber and Wills [23] for a broad coverage of discrete geometry, and the text by Cormen, Leiserson and Rivest [9] for a comprehensive introduction to combinatorial algorithms.

Outline. Section 2 describes four types of space filling diagrams. Section 3 introduces the alpha shape, which is dual to these diagrams. Section 4 focuses on holes in molecules. Section 5 studies formulas for measuring the geometric size of a molecule. Section 6 considers questions of shape and symmetry. Section 7 discusses the phenomena of change under growth and motion. Section 8 concludes the paper.

2 Space Filling Models

The step from molecule to geometric shape is facilitated by space filling diagrams. This section introduces four types: the Van der Waals model, the solvent accessible model, the molecular surface model, and the molecular skin model.

Van der Waals model. A molecule is a collection of atoms. Each atom, A_i, is specified by its type and a location in space: $z_i \in R^3$. The Van der Waals radius maps A_i to a positive real r_i that is the same for all atoms of the same type. The *Van der Waals* model of the molecule is a union of balls:

$$\mathsf{VW} \quad = \quad \{x \in R^3 \mid \exists i : \|x - z_i\| \leq r_i\},$$

see Figure 1. The Van der Waals radius is assigned

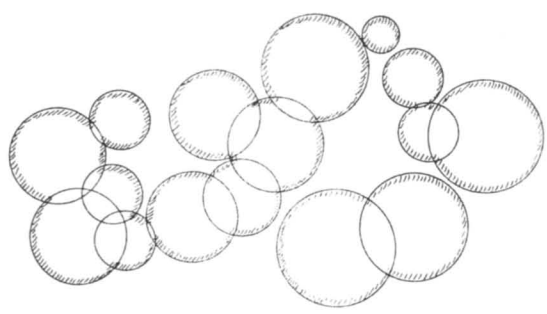


Figure 1: *Van der Waals model: each disk represents an atom specified by its center and radius.*

so atoms in non-binding position do not overlap, see Creighton [10]. Since overlap depends on two atoms and not on one, we can imagine a certain amount of ambiguity in the assignment of radii and it seems indeed common that different research schools in molecular biology use slightly different maps.

Solvent accessible model. Interaction between the Van der Waals model and a solvent represented as an omnipresent sphere of radius ϱ is studied by increasing the radius of every atom ball by ϱ. A solvent sphere is disjoint from VW iff its center lies outside the *solvent accessible* model define by Lee and Richards [30, 38] as the union of enlarged balls:

$$\mathsf{SA} \quad = \quad \{x \in R^3 \mid \exists i : \|x - z_i\| \leq r_i + \varrho\},$$

see Figure 2. The overlap free positions of the solvent

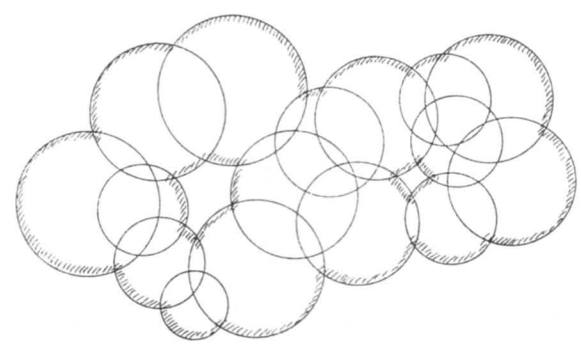


Figure 2: *Solvent accessible model: the radius of each disk in Figure 1 is increased by a constant, which is the radius of the solvent.*

are exactly the centers in the complement of SA. It follow that a solvent particle can move between locations $x, y \in R^3$ without touching VW iff x and y belong to the same component of the complement of SA.

Molecular surface model. A model that resembles in size the Van der Waals and in connectivity the solvent accessible model is the *molecular surface* model. It consists of all points $x \in R^3$ that lie outside all solvent spheres disjoint from VW:

$$\mathsf{MS} \quad = \quad \{x \in R^3 \mid \forall y \notin \mathsf{SA} : \|x - y\| > \varrho\},$$

see Figure 3. According to Michael Connolly, the molecular surface model was originally introduced by Greer and Richards. The first computer program for constructing its surface is due to Connolly [7]. The surface is generated by a sphere of radius ϱ rolling about VW. The envelope of this motion is a surface that bounds MS. It consists of sphere and torus patches connected in a tangent continuous manner. Because of

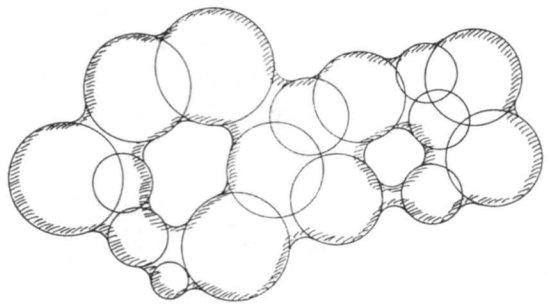


Figure 3: *Molecular surface model: a circle representing the solvent rolls about the Van der Waals model.*

occasional self-intersections, this surface is in general different from the boundary of MS. Two consequences are that the boundary is not everywhere tangent continuous, and that MS and SA do not necessarily have the same homotopy type.

Molecular skin model. Think of the torus patches of MS as blending surfaces that connect neighboring spheres in VW. A similar but different blending effect is achieved by shrinking an infinite family of spheres. The family, F, is obtained by increasing each radius by a factor $\sqrt{2}$ and adding combinations of 2, 3, and 4 balls. The *molecular skin* model is the union of the family after shrinking the radius of every ball by a factor $\frac{\sqrt{2}}{2}$:

$$\mathsf{SK} \quad = \quad \{x \in R^3 \mid \exists (z,r) \in F : \|x - z\| \leq \frac{\sqrt{2} \cdot r}{2}\}.$$

Details of this construction including a precise specification of F can be found in section 6. The surface of SK consists of sphere and hyperboloid patches that meet in a tangent continuous manner. Similar to MS, the molecular skin model is not homotopy equivalent to SA, but it is homotopy equivalent to the union of atom balls whose radii are increased by a factor of $\sqrt{2}$. A property unique to SK is a symmetry between inside and outside useful in studies of complementarity.

3 Dual Models

The space filling models have a common dual counterpart. To construct it we introduce the Voronoi diagram that decomposes the union of balls into convex pieces. This union could either be a Van der Waals or a solvent accessible model. The overlap between the pieces defines what we call the dual complex.

Voronoi cells. The *weighted distance* of a point $x \in R^3$ from A_i is $\pi_i(x) = \|x - z_i\|^2 - r_i^2$. For $r_i = 0$ this is the same as the square of the Euclidean distance. Weighted distance is defined for any real so possibly also negative square radius. The corresponding extension from real to imaginary radii and balls is useful in sections 6 and 7 when we talk about the complement and metamorphosis of a molecule. The *Voronoi cell* of A_i is the region of points whose weighted distance to A_i is at least as small as to any other ball:

$$V_i \quad = \quad \{x \in R^3 \mid \forall j : \pi_i(x) \leq \pi_j(x)\},$$

see Figure 4. Each Voronoi cell is an intersection of

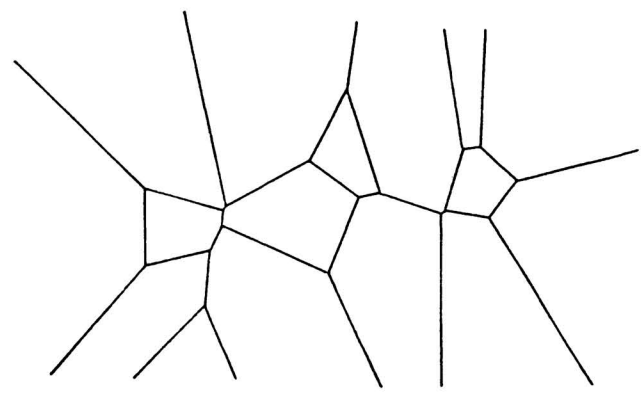


Figure 4: *Voronoi cells: each atom generates a convex polyhedron.*

closed half-spaces and therefore a convex polyhedron. Any two cells overlap at most along some piece of their boundary, and together the collection of cells covers the entire R^3. To simplify the exposition we assume throughout this paper that the balls are in nondegenerate position. With this assumption the common intersection of $k + 1 \geq 1$ Voronoi cells is either

empty or a common $(3-k)$-dimensional face. For $k \geq 4$ the common intersection is always empty.

The cells are named after the Russian mathematician Voronoi who studied the geometry of numbers [40]. The terminology in the literature is not entirely uniform and Voronoi cells are also known as power cells, Dirichlet cells, Thiessen polygons, etc.

Delaunay complex. The Delaunay complex is a collection of simplices that records the overlap pattern among Voronoi cells. Specifically, the complex contains the convex hull of $k+1$ ball centers iff the corresponding $k+1$ Voronoi cells have a non-empty common intersection. To develop the necessary mathematical notation let B be a subset of $k+1$ atoms and define $z_B = \{z_i \mid A_i \in B\}$ and $V_B = \{V_i \mid A_i \in B\}$. The *Delaunay complex* is

$$\mathcal{D} \quad = \quad \{\sigma = \operatorname{conv} z_B \mid \bigcap V_B \neq \emptyset\},$$

see Figure 5, where $\operatorname{conv} z_B$ denotes the convex hull of the points in z_B. The non-degenerate position assumption implies that the points in z_B are affinely independent and their convex hull is a simplex of dimension $\operatorname{card} B - 1$.

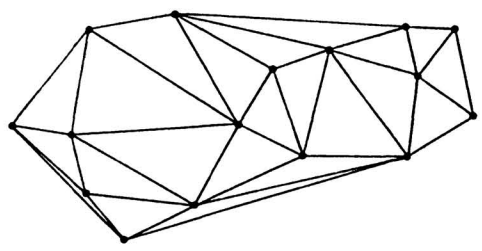


Figure 5: *Delaunay complex: Voronoi cells are represented by vertices and the pairwise and triplewise overlap among cells is represented by edges and triangles.*

The complex is named after the Russian mathematician Delaunay, also Delone, who defines the complex as the collection of cells that satisfy the empty sphere property [11]. For a set of points the condition is expressed in terms of spheres passing through points. The generalization to a set of balls is expressed in terms of spheres that are *orthogonal* to balls, which means the square distance between their centers equals the sum of square radii.

Fact 1 *$\sigma = \operatorname{conv} z_B$ belongs to $\mathcal{D}$ iff there is a ball orthogonal to all $A_i \in B$ and further than orthogonal from all other balls.*

Dobkin and Laszlo [13] design a data structure for storing a 3-dimensional Delaunay complex and Facello [21] describes the implementation of an incremental algorithm for constructing it.

Alpha shape. In order to apply duality to molecules it is necessary to restrict the construction of simplices to the portions of Voronoi cells covered by the atom balls. This leads to the definition of alpha complexes and alpha shapes. Observe first that the Voronoi cells decompose the union of balls into convex regions: $V_i \cap \mathsf{VW} = V_i \cap A_i$. The *dual* or *alpha complex* records the overlap pattern among these regions:

$$\mathcal{K} \quad = \quad \{\sigma = \operatorname{conv} z_B \mid \bigcap V_B \cap \bigcap B \neq \emptyset\},$$

see Figure 6. The dual complex is a subcomplex of the Delaunay complex: $\mathcal{K} \subseteq \mathcal{D}$. The *alpha shape* is the union of simplices in $\mathcal{K}$. It is often convenient to ignore

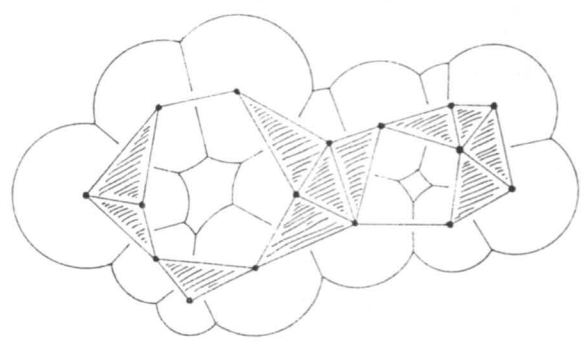


Figure 6: *Dual or alpha complex: it contains the subset of simplices in $\mathcal{D}$ that record overlap among Voronoi cells that reach into the union of balls.*

the difference between alpha shape and alpha complex, but sometimes it is essential to remember that the former is a polyhedron and thus a subset of space while the latter is a complex and therefore a combinatorial structure.

Alpha shapes have been introduced for points in the plane by Edelsbrunner, Kirkpatrick and Seidel [19]. They have been generalized to points in three dimensions by Edelsbrunner and Mücke [20] and to balls in general dimensions by Edelsbrunner [14]. The alpha is a parameter that expresses growth of balls. When we talk about a fixed set of balls this parameter is irrelevant, but it will be used in section 7 where simultaneous growth of atom balls is discussed.

Duality. As a first application we mention the use of the alpha shape in constructing the dual union of balls. Consider for example the Van der Waals model: $\mathsf{VW} = \bigcup_i A_i$. The boundary of VW consists of sphere patches that meet along circular arcs and corner points. We call these k-patches, for $k = 2, 1, 0$. The direct construction of the patches is made difficult by numerical inaccuracies resulting from the use of floating-point computation. A more robust strategy is to first compute the alpha complex and to use the dual correspondence for all decisions on patch overlap. This idea has been implemented by Akkiraju and Edelsbrunner [1] who construct a triangulation of the surface. Another fast algorithm for constructing the surface, but not a triangulation, is described in Halperin and Overmars [25].

The particular dual correspondence useful in this context relates the boundary of VW with the boundary of the alpha shape, see Figure 6. To express the correspondence let B be a set of $k+1$ balls and define S_B as the set of spheres bounding the balls in B.

Fact 2 *The intersection of the spheres, $\bigcap S_B$, contains a $(2-k)$-patch of the boundary of* VW *iff the simplex $\sigma = \operatorname{conv} z_B$ belongs to the boundary of the alpha shape. Furthermore, a patch is face of another patch iff the corresponding simplex of the latter is a face of the corresponding simplex of the former patch.*

4 Connectivity

A topologically interesting aspect of molecules is their connectivity. It is fairly common for biomolecules to have tunnels and voids. A tunnel is a hole through the molecule that is accessible from the outside whereas a void is a hole that is completely surrounded by atoms. Classifying and counting holes is one of the major topics in topology. A particularly useful formalism is that of homology groups and Betti numbers, see e.g. Munkres [33]. Curiously, the type of holes that is most important in the study of biomolecules is not a hole in the topological sense at all. These holes are cavities or depressions through which biomolecules interact. They are called hollows in the philosophical treatment of holes by Casati and Varzi [5] and pockets in this paper.

Voids and dual simplex sets. A *void* of a shape $M \subseteq R^3$ is a bounded component of the complement, $R^3 - M$. For example, the solvent accessible model $M = \mathsf{SA}$ in Figure 2 has two voids, and so does the corresponding alpha shape in Figure 6. That this is not a coincidence follows from a general result implied by the nerve theorem first proved by Leray [31].

Fact 3 *A union of balls and the corresponding alpha shape are homotopy equivalent.*

In a nutshell this means that the two are connected the same way: they have the same type, number, and arrangement of holes. The correspondence between the two shapes can be made geometrically more specific, see Figure 6.

Fact 4 *A union of balls contains the corresponding alpha shape, and each void of the alpha shape contains the corresponding void of the union of balls.*

A void of the alpha shape is covered by a collection of simplices that belong to $\mathcal{D}$ but not to $\mathcal{K}$. We call this the *dual simplex set* of the void. Each dual simplex set is open at the boundary. Applications of these sets will be discussed in section 5.

Pockets. Cavities that are not voids are more difficult to define because it is not immediately clear where

they should start and what criterion can be used to differentiate shallow depressions from ones that should be taken seriously. Connolly [8] sought to visualize such cavities indirectly through highlighting the skeleton of the complement. Kuntz [27] identifies a region of the complement as a cavity if a sphere can get stuck there.

We follow the approach of Edelsbrunner, Facello and Liang [18] who use an acyclic relation of the tetrahedra in the Delaunay complex to define pockets. Specifically, let the *orthosphere* of a tetrahedron $\sigma = \operatorname{conv} z_B$ be the unique sphere orthogonal to all four balls in B. Two tetrahedra form a pair in the relation, $\sigma \preceq \tau$, if they share a common triangle and the center of the orthosphere of σ lies on τ's side of the triangle. Similarly, we set $\sigma \preceq \infty$ if one of the triangles of σ belongs to the convex hull and the center of the orthosphere lies on the other side of that triangle. Take all tetrahedra in $\mathcal{D} - \mathcal{K}$, arrange them in groups defined by shared descendents, and remove the group containing ∞. For each remaining group add the faces that do not belong to $\mathcal{K}$ and combine two groups if they share at least one triangle. The result is what we call the *dual simplex sets* of pockets. A *pocket* is the part of space outside the ball union covered by simplices in its dual simplex set. In the end, the notion of pocket is very similar to Kuntz's ideas about cavities.

The dual simplex set of a pocket is typically partially open and partially closed. Define the *mouth* as a component of boundary simplices. Each mouth describes a connection between the pocket and the space outside the molecule. A void is a pocket without mouth. Figure 7 shows a large pocket formed by the HIV-1 protease molecule. Pockets of one molecule naturally interact with protrusions of another. It is desirable to define a protrusion in a symmetric manner, such as the pocket of the complement. This is indeed possible once we understand that the structure of the complement can also be expressed in terms of finite sets of balls, see section 6.

Computing and counting holes. The three types of topological holes of a molecule are counted by Betti numbers, which are the ranks of the three possibly nontrivial homology groups. Definitions of these concepts are technically fairly involved and can be found in every text on algebraic topology. Intuitively, the Betti number β_0 counts the components, β_1 counts the independent tunnels, and β_2 counts the voids. Traditional algorithms, such as the matrix manipulation method described in Munkres [33], are impractical for the size of complexes that arise as duals of biomolecules. An algorithm that is efficient in R^3 is developed by Delfinado and Edelsbrunner [12]. It computes all three Betti numbers using a filter of the Delaunay complex. This is a linear ordering of the simplices so every prefix is a subcomplex, and the alpha complex is one such prefix. The simplices following the alpha complex in the filter form the dual simplex sets of all voids and of the outside of the molecule.

The filter can be extended to also represent the relation over the tetrahedra needed to define dual simplex sets of pockets. The Betti number algorithm can then be extended and compute pockets by collecting simplices of $\mathcal{D} - \mathcal{K}$ in groups as determined by the relation, see Edelsbrunner, Facello and Liang [18].

5 Geometric Size

There is a connection between the topological complex representation of a molecule and its geometric size. We restrict the discussion to the volume of a union of balls and a void, but similar formulas also exist for surface area and length of arcs.

Inclusion-exclusion. This combinatorial principle expresses the volume of a union of balls as an alternating sum of volumes of common intersections:

$$\operatorname{vol} \bigcup A \;=\; \sum_{B \subseteq A} (-1)^{\operatorname{card} B - 1} \operatorname{vol} \bigcap B.$$

The number of terms is exponential in the number of balls, and the volume of $\bigcap B$ is progressively more difficult to evaluate as the cardinality of B increases. Kratky [26] observed that most of the terms in the inclusion-exclusion formula are redundant, and this includes the non-zero terms many of which cancel. Specifically, he considered the common intersection of a finite collection of disks and proves that there is an

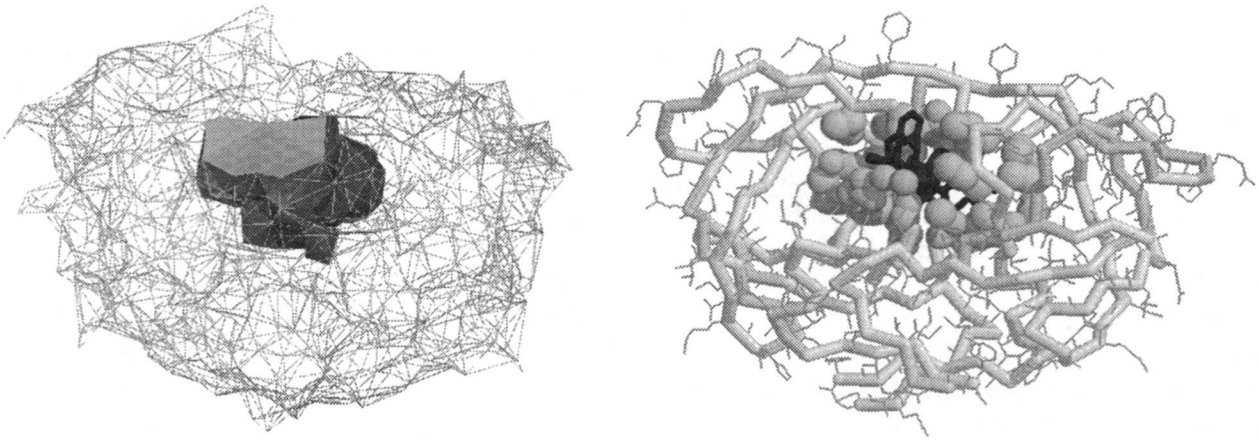


Figure 7: *HIV-1 protease: dual simplex set inside alpha shape to the left, and pocket indicated by balls defining the wall of the pocket to the right.*

exact inclusion-exclusion formula that contains only terms for sets B of cardinality 1, 2 and 3. Scheraga and collaborators use the generalization of Kratky's result to balls in R^3 in their software for volume computation [36].

Truncated formulas. A weakness of Kratky's result is that it does not specify which terms of the basic inclusion-exclusion formula are redundant. For example, the formula that simply takes all terms for sets B of cardinality 1, 2, 3 is not correct. Naiman and Wynn [34] observe that the formula that has a term for each Delaunay simplex is exact. Even this formula contains redundant information, and the smallest inclusion-exclusion formula that gives the exact volume contains a term for each simplex in the alpha complex.

Fact 5 *The volume of a union of balls is*

$$\mathrm{vol}\bigcup A \;=\; \sum_{\sigma\in\mathcal{K}}(-1)^{\mathrm{card}\,B-1}\mathrm{vol}\bigcap B,$$

where B is the set of balls so that $\sigma = \mathrm{conv}\, z_B$.

This result is due to Edelsbrunner [14] who proves it by integration over Euler characteristics. All intersections of $k+1$ balls that correspond to k-simplices in $\mathcal{K}$ have the same combinatorial type, see Figure 8. For this

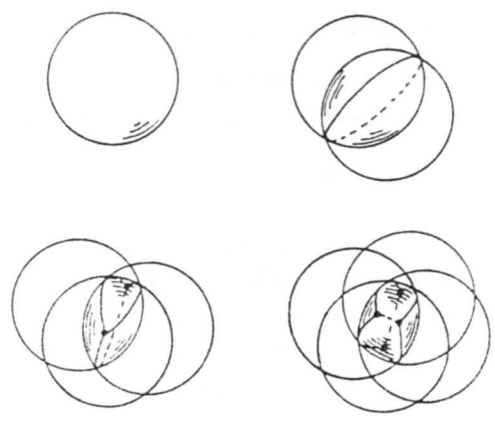


Figure 8: *The common intersection of 1, 2, 3, 4 spherical balls.*

reason the formula in Fact 5 is significantly easier to implement than that over simplices in $\mathcal{D}$.

The proof in [14] reveals a relationship between the union of balls in R^3 and a convex polyhedron in R^4.

More generally, one can consider the union of intersections of balls, which corresponds to a possibly non-convex polyhedron in R^4. Edelsbrunner [15] shows that the volume can be computed with a truncated inclusion-exclusion formula derived from the boundary complex of that polyhedron.

Angle-weight formulas. The volume of a void can be computed using Fact 5 if we first fill the void with balls and then subtract from the total volume the volume of the original molecule. This seems roundabout and indeed there is a direct formula based on the dual simplex set. For a tetrahedron σ and a face τ define the *angle*, $\varphi_{\tau,\sigma}$, equal to $\frac{1}{2}$ if τ is a triangle, equal to the dihedral angle at τ if it is an edge, and equal to the solid angle at τ if it is a vertex. The volume is the total volume of the dual simplex set minus its intersection with the ball union.

Fact 6 *The volume of void V with dual set of tetrahedra T is*

$$\begin{aligned} \operatorname{vol} V &= \sum_{\sigma \in T} \operatorname{vol} \sigma \\ &\quad + \sum_{\sigma \in T} \sum_{\sigma \supseteq \tau \in \mathcal{K}} (-1)^{\operatorname{card} B} \cdot \varphi_{\tau,\sigma} \cdot \operatorname{vol} \bigcap B, \end{aligned}$$

where B is such that $\tau = \operatorname{conv} z_B$.

The dimension of any simplex τ in the above formula is at most 2. The terms are therefore restricted to intersections of at most 3 spheres.

Experimental results. The alpha shapes software includes the implementation of inclusion-exclusion formulas for volume and surface area and has been publically available for many year at `ftp.ncsa.uiuc.edu`. Its functionality with a focus on geometric size is described in Edelsbrunner, Facello, Fu and Liang [17]. Peters, Fauck and Frömmel use the software to automatically identify ligand binding sites near the boundary of proteins [37].

The more recent facility to identify and compute pockets is used by Liang and McGee to estimate hydration change due to osmotic stress [32]. Edelsbrunner, Liang and Woodward [29] use the software and extensions of the inclusion-exclusion formulas to pockets for the purpose of gathering statistical data about average sizes of pockets and mouths and computing correlation to the size of ligands, etc. For general binding sites they observe a remarkable variety in size, and for the HIV-1 protease molecule they measure a surprisingly large range of sizes for a flexible pocket that adjusts to the binding ligand.

6 Shape and Symmetry

Questions of complementarity are common in biology, and Blaney and Dixon [4] study the problem of molecular docking where complementarity assumes a concrete geometric meaning. An interesting special case arises when one type of molecule forms aggregates where the molecule docks with a copy of itself. To do this the molecule must be locally self-complementary. This section studies the molecular skin model, which provides a means to realize perfect complementarity.

Body and skin. Given a finite set of atom balls, A_i, the skin is obtained as the envelope of an infinite family, F. We generate this family through addition and scaling. A ball is the preimage of the weighted distance function $\pi_i : R^3 \to R$: $A_i = \pi_i{}^{-1}(-\infty, 0]$. Addition and multiplication with a scalar are well defined and give rise to the vector space of functions. We borrow the vector space for the balls and define

$$\sum \gamma_i \cdot A_i \quad = \quad \pi^{-1}(-\infty, 0],$$

where $\pi(x) = \sum \gamma_i \cdot \pi_i(x)$. In this vector space we define the convex hull of the A_i as the set of combinations $\sum \gamma_i \cdot A_i$ with $\sum \gamma_i = 1$ and $\gamma_i \geq 0$ for every i, as usual. For each ball (z, r) in the convex hull, the infinite family F contains the ball $(z, \frac{\sqrt{2}r}{2})$, which is a shrunken copy that shares the same center. The *body* is the union of balls in F, and the *skin* is the envelope, see Figure 9. The only difference between the body and the molecular skin model in section 2 is that the latter first grows the balls to compensate for the shrinking effect inherent in the construction. The definition of body and skin is due to Edelsbrunner [16]. Some of the important properties are:

Fact 7 *The body and the union of the A_i are homotopy equivalent.*

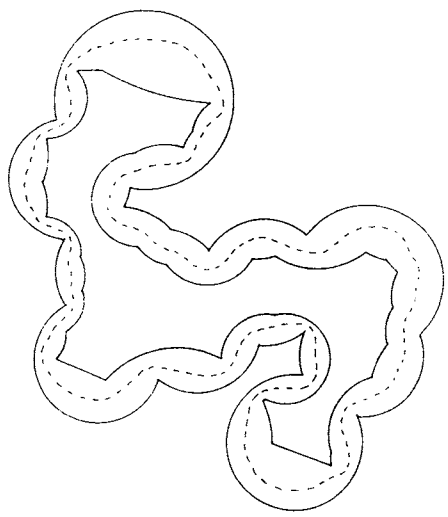


Figure 9: *From outside in: boundaries of the union, the body, and the complement of the union of orthogonal disks.*

Together with Fact 3 this implies that the body is homotopy equivalent to the alpha shape.

Fact 8 *The skin consists solely of sphere and hyperboloid patches, and in the non-degenerate case it is everywhere tangent continuous.*

The exception to tangent continuity happens when two of the atom balls touch and the point where they touch lies on the boundary of the union. In this case the hyperboloid that blends between the two spheres degenerates to a double-cone and tangent continuity is violated at the apex of that double-cone.

Complementarity. We use a fundamental symmetry between Voronoi and Delaunay complexes to represent the complement of a molecule by another finite set of balls. The idea is similar to the computer program DOCK by Kuntz and collaborators where a pocket is filled with balls to get a positive imprint [28]. We replace the heuristic filling process by a canonical construction.

Recall that each tetrahedron $\sigma \in \mathcal{D}$ corresponds to a vertex shared by four Voronoi cells. Let $B_j = (y_j, q_j)$ be the ball whose center, y_j, is that Voronoi vertex and whose square radius, q_j^2, is the weighted distance of y_j from any one of the four balls that generate the four Voronoi cells. By construction, B_j is orthogonal to the four balls and further than orthogonal from all other A_i. To complete the construction we add an infinitely large ball (a half-space) with center at the infinite end of every unbounded Voronoi edge. An example is shown in Figure 9 where the innermost curve is the boundary of the union of the B_j. The new set of balls has some possibly surprising properties.

Fact 9 *The complement of the interior of the $\bigcup_j B_j$ is contained in and homotopy equivalent to $\bigcup_i A_i$.*

This implies that the A_i and B_j together cover the entire R^3.

Fact 10 *The skin of the B_j is the same as the skin of the A_i. The bodies of the two sets are complementary and intersect in the common skin.*

Protrusions. One of the applications of the set of balls B_j is that it facilitates the definition and construction of protrusions. Recall the definition of a pocket in section 4 and think of a pocket as a protrusion of the complement. Symmetrically, we can think of a protrusion as a pocket of the complement.

Consider the Delaunay complex of the B_j, which is the same as the set of Voronoi cells of the A_i together with their faces. The dual complex of the B_j is a subcomplex of that Delaunay complex. Define a relation over the cells, as in section 4, and group the cells and their faces outside the dual complex to form dual cell sets of pockets. One difficulty with this approach is the asymmetry with respect to ∞, which is part of the old but not of the new relation. We can either manually pick cells and remove the groups that contain them, or we can construct another set of balls C_k from the non-imaginary B_j, the same way as the B_j are derived from the A_i. Because we select only non-imaginary balls, the set of C_k is not equal to the set of A_i, but their union and body are the same. The cell generated by the largest of the C_k would be the best candidate to play the role of ∞ in the construction of protrusions.

While the Delaunay complex of the A_i is simplicial, if we assume non-degenerate position, the Delaunay

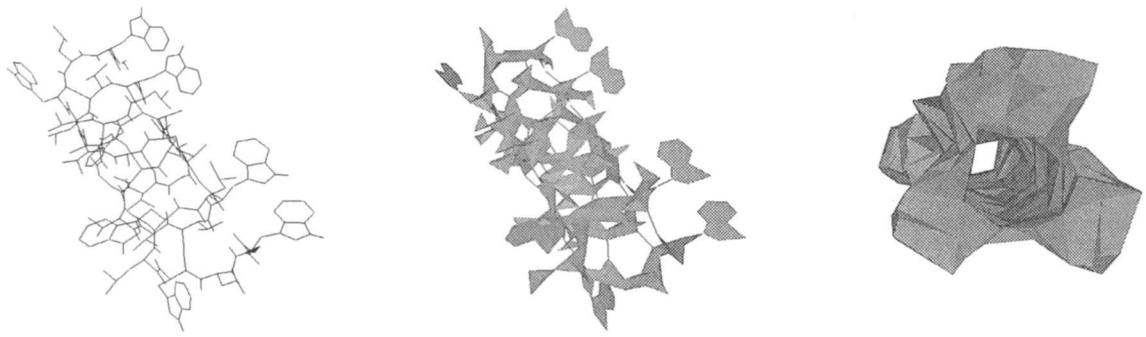


Figure 10: *Three alpha shapes of the Gramicidin A molecule. The parameter α grows from left to right.*

complex of the B_j is simple. This is because it is not defined by input data but rather by balls derived from the input data.

7 Metamorphosis

The assumption of static molecules with fixed atoms is convenient but certainly not realistic and therefore of limited use.

Growth. A simple form of change is growth, but even in this framework there is more freedom than can be controlled. Alpha shapes are designed to control simultaneous growth with only one parameter: α^2. Similar to the square radii, α^2 can be negative in which case α is imaginary. For a given $\alpha^2 \in R$ we define

$$A_i(\alpha) \quad = \quad (z_i, \sqrt{r_i^2 + \alpha^2}).$$

For vanishing α we have the original balls, and for vanishing r_i we have all balls of the same radius, α. The *α-complex*, $\mathcal{K}_\alpha$, is the dual complex of the union of the $A_i(\alpha)$. An example is shown in Figure 10.

In this framework smaller balls grow faster than larger ones and as α grows all balls converge to radius α. The ramification of the quadratic growth formula is computational efficiency. The weighted distance of $x \in R^3$ from $A_i(\alpha)$ is

$$\|x - z_i\|^2 - r_i^2 + \alpha^2 \quad = \quad \pi_i(x) + \alpha^2.$$

In words, the ordering of the balls by weighted distance of x does not change with changing α. It follows that the Voronoi cells remain unchanged and so does the Delaunay complex. The only thing that changes with growing α is that $\mathcal{K}_\alpha$ contains more and more simplices of $\mathcal{D}$ until, for large enough α, we have $\mathcal{K}_\alpha = \mathcal{D}$. An interesting application that still needs to be explored is the use of the hierarchy of α-complexes in algorithmic approaches to shape matching as it occurs in the simulation of docking processes.

The difference between the Van der Waals and the solvent accessible models of the A_i can generally not be bridged by quadratic growth. In this case we observe linear growth defined by

$$A_i(\alpha) \quad = \quad (z_i, r_i + \alpha).$$

A computationally reasonable efficient approach to representing this one-parametric family constructs the Delaunay complex for the $A_i(0) = A_i$ and adopts the complex through edge and triangle flips as α changes continuously. Details are described by Facello [22].

Motion. A molecular shape represented by a space filling diagram deforms if the atom balls move about. Possibly the geometrically cleanest deformations are observed when the molecular skin model is used. It remains tangent continuous at all times except at points in time and space where it changes topology. The

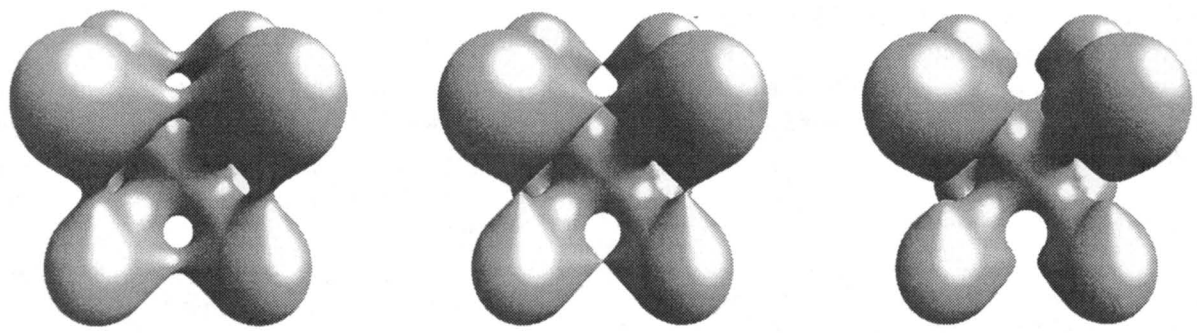


Figure 11: *Three skins of the same nine balls that go through a small amount of shrinking.*

breaking of a connection is locally realized by a hyperboloid of one sheet that gets narrower, becomes a double cone, and flips over to a hyperboloid of two sheets, see Figure 11. The creation of a tunnel goes through the same sequence backwards and with inside and outside exchanged.

A possible application of general motion is, for example, the visualization of the folding process. The driving force of the motion could be a molecular dynamics software, such as the one created by Schulten and collaborators [35]. The numerical simulation produces snap shots of the computed motion that need to be connected by small interpolating homotopies. A full implementation of such a system could be used to efficiently and accurately track any size and topology change during the motion. This information can be fed back into the software that computes the motion for the next time step.

Space of shapes. The ability to canonically deform shapes opens up the possibility to construct spaces of shapes. This idea is pursued by Cheng, Edelsbrunner, Fu and Lam who use a single parameter to control the deformation from one shape to another [6]. To sketch the particulars of the method, let the two shapes be the skins of two sets of balls, X and Z. At time $t \in [0,1]$ the set of moving balls is

$$Y(t) \quad = \quad \{(1-t) \cdot X_i + t \cdot Z_j \mid X_i \in X, Z_j \in Z\}.$$

Even far apart balls X_i and Z_j are matched, but due to the vector space of balls their combination is usually redundant as it shrinks away on its trip from X_i to Z_j. The time parameter defines a 1-dimensional space of shapes. Members of this space can be combined with a third shape, etc. With $k+1$ shapes, each the skin of a set of balls, we get a k-dimensional space of shapes. Any shape in this space is indexed by its barycentric coordinates that specify the fractions in which the given shapes contribute to the mixing of shapes.

It would be interesting to test this idea of shape indexing in the construction of databases for small drug compounds. To get the method off the ground we need, in addition to the synthesis algorithm outlined above, and analysis algorithm that finds for a given skin the best approximation within the space.

8 Conclusion

The author's personal interest in molecular biology is based on the strong connection to geometry and topology. He is still amazed, in fact, of the extent in which biological questions influenced his thinking about problems in quite a few seemingly unrelated areas.

Acknowledgements

Research by the author of this paper is partially supported by grant NSF-CCR-97-12088. The author thanks all collaborators on projects that lead to the

results and insights surveyed in this paper: Nataraj Akkiraju, Chandrajit Bajaj, Ho-Lun Cheng, Siu-Wing Cheng, Cecil Delfinado, Michael Facello, Ping Fu, David Kirkpatrick, Ka-Po Lam, Jie Liang, Ernst Mücke, Valerio Pascucci, Raimund Seidel, Shankar Subramaniam, Clare Woodward.

References

[1] N. Akkiraju and H. Edelsbrunner. Triangulating the surface of a molecule. *Discrete Appl. Math.* **71** (1996), 5-22.

[2] P. S. Alexandrov. *Elementary Concepts of Topology.* Translated from Russian by A. E. Farley, Dover, New York, 1961.

[3] R. F. W. Bader. *Atoms in Molecules. A Quantum Theory.* Clanrendon Press, Oxford, England, 1994.

[4] J. M. Blaney and J. S. Dixon. A good ligand is hard to find: automated docking methods. *Perspectives in Drug Discovery and Design* **1** (1993), 301–319.

[5] R. Casati and A. C. Varzi. *Holes and Other Superficialities.* MIT Press, Cambridge, Massachusetts, 1994.

[6] S.-W. Cheng, H. Edelsbrunner, P. Fu and K.-P. Lam. Design and analysis of planar shape deformation. *In* "Proc. 14th Ann. Sympos. Comput. Geom., 1998", to appear.

[7] M. L. Connolly. Analytical molecular surface calculation. *J. Appl. Cryst.* **16** (1983), 548–558.

[8] T. H. Connolly. Molecular interstitial skeleton. *Computer Chem.* **15** (1991), 37–45.

[9] T. H. Cormen, Ch. E. Leiserson and R. L. Rivest. *Introduction to Algorithms.* MIT Press, Cambridge, Mass., 1990.

[10] T. E. Creighton. *Proteins. Structures and Molecular Principles.* Freeman, New York, 1984.

[11] B. Delaunay. Sur la sphère vide. *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk* **7** (1934), 793–800.

[12] C. J. A. Delfinado and H. Edelsbrunner. An incremental algorithm for Betti numbers of simplicial complexes on the 3-sphere. *Comput. Aided Geom. Design* **12** (1995), 771–784.

[13] D. P. Dobkin and M. J. Laszlo. Primitives for the manipulation of three-dimensional subdivisions. *Algorithmica* **4** (1989), 3–32.

[14] H. Edelsbrunner. The union of balls and its dual shape. *Discrete Comput. Geom.* **13** (1995), 415–440.

[15] H. Edelsbrunner. Algebraic decomposition of nonconvex polyhedra. *In* "Proc. 36th Ann. IEEE Sympos. Found. Comput. Sci. 1995", 248–257.

[16] H. Edelsbrunner. Deformable smooth surface design. *Discrete Comput. Geom.*, to appear.

[17] H. Edelsbrunner, M. A. Facello, P. Fu and J. Liang. Measuring proteins and voids in proteins. *In* "Proc. 28th Ann. Hawaii Internat. Conf. System Sciences, 1995", vol. V: Biotechnology Computing, 256–264.

[18] H. Edelsbrunner, M. A. Facello and J. Liang. On the definition and the construction of pockets in macromolecules. *In* "Proc. Pacific Sympos. Biocomputing 1996", World Scientific, Singapore, 272–281.

[19] H. Edelsbrunner, D. G. Kirkpatrick and R. Seidel. On the shape of a set of points in the plane. *IEEE Trans. Inform. Theory* **IT-29** (1983), 551–559.

[20] H. Edelsbrunner and E. P. Mücke. Three-dimensional alpha shapes. *ACM Trans. Graphics* **13** (1994), 43–72.

[21] M. A. Facello. Implementation of a randomized algorithm for Delaunay and regular triangulations in three dimensions. *Comput. Aided Geom. Design* **12** (1995), 349–370.

[22] M. A. Facello. Geometric techniques for molecular shape analysis. Report UIUCDCS-R-96-1967, Dept. Comput. Sci., Univ. Illinois, Urbana, 1996.

[23] P. M. Gruber and J. Wills, eds. *Handbook of Convex Geometry, volumes A and B.* North-Holland, Amsterdam, 1993.

[24] V. Guillemin and A. Pollack. *Differential Topology.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1974.

[25] D. Halperin and M. H. Overmars. Spheres, molecules, and hidden surface removal. *In* "Proc. 10th Ann. Sympos. Comput. Geom., 1994", 113–122.

[26] K. W. Kratky. The area of intersection of n equal circular disks. *J. Phys. A: Math. Gen.* **11** (1978), 1017–1024.

[27] I. D. Kuntz. Structure-based strategies for drug design and discovery. *Science* **257** (1992), 1078–1082.

[28] I. D. Kuntz, J. M. Blaney, S. J. Oatley, R. Langridge and T. E. Ferrin. A geometric approach to macromolecule-ligand interactions. *J. Mol. Biol.* **161** (1982), 269–288.

[29] J. Liang, H. Edelsbrunner and C. Woodward. Anatomy of protein pockets and cavities: measurement of binding site geometry and implications for ligand design. *Protein Science*, to appear.

[30] B. Lee and F. M. Richards. The interpretation of protein structures: estimation of static accessibility. *J. Mol. Biol.* **55** (1971), 379–400.

[31] J. Leray. Sur la forme des espaces topologiques et sur les point fixes des représentations. *J. Math. Pures Appl.* **24** (1945), 95–167.

[32] J. Liang and M. P. McGee. Mechanisms of coagulation factor Xa inhibition by antithrombin: Correlation between hydration structures and water transfer during reactive loop insertion. *Biophys. J.*, submitted, 1997.

[33] J. R. Munkres. *Elements of Algebraic Topology.* Addison-Wesley, Redwood City, California, 1984.

[34] D. Q. Naiman and H. P. Wynn. Inclusion-exclusion-Bonferroni identities and inequalities for discrete tube-like problems via Euler characteristics. *Ann. Statist.* **20** (1992), 43–76.

[35] M. Nelson, W. Humphrey, A. Gürsoy, A. Dalke, L. V. Kalé, R. Skeel, K. Schulten and R. Rufkin. MDScope – a visual computing environment for structural biology. *Comput. Phys. Commun.* **91** (1995), 111–134.

[36] G. Perrot, B. Cheng, K. D. Gibson, J. Vila, A. Palmer, A. Nayeem, B. Maigret and H. A. Scheraga. MSEED: a program for rapid determination of accessible surface areas and their derivatives. *J. Comput. Chem.* **13** (1992), 1–11.

[37] K. P. Peters, J. Fauck and C. Frömmel. The automatic search for ligand binding sites in proteins of known three-dimensional structure using only geometric criteria. *J. Mol. Biol.* **256** (1996), 201–213.

[38] F. M. Richards. Areas, volumes, packing, and protein structure. *Ann. Rev. Biophys. Bioeng.* **6** (1977), 151–176.

[39] G. D. Rose. No assembly required. *The Sciences* **36** (1996), 26–31.

[40] G. Voronoi. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. *J. Reine Angew. Math.* **133** (1907), 97–178, and **134** (1908), 198–287.

Taylor & Francis
Taylor & Francis Group
http://taylorandfrancis.com

Algorithms for Fence Design

Robert-Paul Berretty, *University of Utrecht, Utrecht, The Netherlands*
Ken Goldberg, *University of California at Berkeley, CA, USA*
Mark H. Overmars, *University of Utrecht, Utrecht, The Netherlands*
A. Frank van der Stappen, *University of Utrecht, Utrecht, The Netherlands*

A common task in automated manufacturing processes is to orient parts prior to assembly. We address sensorless orientation of a polygonal part on a conveyor belt by a sequence of stationary fences across this belt. Since fences can only push against the motion of the belt, it is a challenging problem to compute fence designs which orients a given part. In this paper, we give several polynomial-time, algorithms to compute fence designs which are optimal with respect to various criteria. We address both frictionless and frictional fences. We also compute modular fence designs in which the fence angles are restricted to a discrete set of angles instead of an interval.

1 Introduction

Many automated manufacturing processes require parts to be oriented prior to assembly. A part feeder takes in a stream of identical parts in arbitrary orientations and outputs them in a uniform orientation. Part feeders often use data obtained from some kind of sensing device to accomplish their task. We consider the problem of *sensorless orientation* of parts, in which the initial pose of the part is assumed to be unknown. In sensorless manipulation, parts are positioned and/or oriented using passive mechanical compliance. The input is a description of the part shape and the output is a sequence of open-loop actions that moves a part from an unknown initial pose into a unique final pose. Among the sensorless part feeders considered in literature are the parallel-jaw gripper [9, 13], a single pushing jaw [2, 14, 15, 17], a conveyor belt with a sequence of (stationary) fences placed along its sides [7, 18, 21], a conveyor belt with a single rotational fence (1JOC) [1], a tilting tray [12, 16], and vibratory plates and programmable vector fields [5, 6].

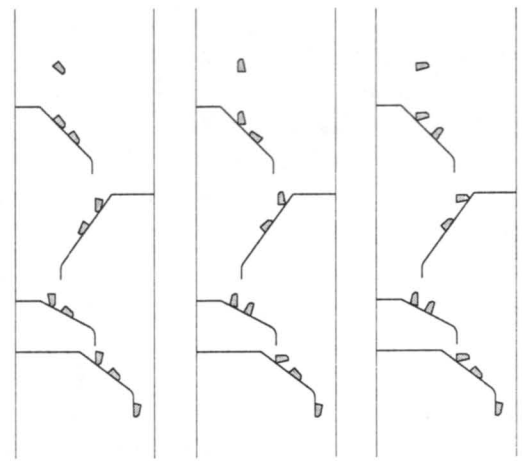


Figure 1: *Three overhead views of the same conveyor belt and fence design. The traversals for three different initial orientations of the same part are displayed. The traversals show that the part ends up in the same orientation in each of the three cases.*

The pushing jaw [2, 14, 15, 17] orients a part by an alternating sequence of pushes and jaw reorientations. The problem of sensorless orientation by a pushing jaw is to find a sequence of push directions that will move the part from an arbitrary initial orientation into a single known final orientation. Such a sequence is referred to as a *push plan.* Goldberg [13] showed that any polygonal part can be oriented by a sequence of pushes. Chen and Ierardi [9] proved that any polygonal part with n vertices can be oriented by $O(n)$ pushes. They showed that this bound is tight by constructing (pathological) n-gons that require $\Omega(n)$ pushes to be oriented. Goldberg gave an $O(n^2)$ algorithm for computing the shortest push plan for a polygon.

The problem of *fence design* is to determine a sequence of fence orientations (see Figure 1) such that the fences with these orientations align the part as it moves down a conveyor belt and slides along these fences [7, 18, 21]. The motion of the belt effectively turns each slide into a push action by the fence in the direction normal to the fence. The fact that the direction of the push, i.e., the normal at the fence, must have a non-zero component in the direction opposite to the motion of the belt imposes a restriction on successive push directions. Fence design can be regarded as finding a constrained sequence of push directions (see Subsection 2.2 for the actual constraints). The additional constraints make fence design considerably more difficult than sensorless orientation by a pushing jaw. Wiegley *et al.* [21] conjectured that a fence design exists for any polygonal part. They gave an exponential algorithm for computing the shortest sequence of fences for a given part.

In [4], we proved the conjecture that a fence design exists for any polygonal part. A more eleborate version of the proof can be found in [3]. Also, we gave an $O(n^3 \log n)$ algorithm for computing a fence design of minimal length (in terms of the number of fences used). In this paper, we first review this algorithm, which is easy to implement. The program can be tuned to take into account certain quality requirements on the fence design, like minimum and maximum (successive) fence angles to prevent impractical steep and shallow fences and a long series of fences on a single side of the belt, which would require an impractically wide conveyor belt. The cost of the incorporation of quality measures is an increase of the algorithm's running time to $O(n^4)$.

The algorithm forms the basis for the new algorithms to compute fence designs, which we present in this paper. The graph based algorithm can be modified to compute frictional fence designs, i.e. when there is friction between the part and the fences. Also, we can change the algorithm to compute fence designs which use modular fences of fixed angles. The running time of the algorithm is then $O(mn^2)$, with m the number of available fence angles.

In [4], we showed that fence designs of length $O(n)$ exist for a large class of parts. In this paper, we give a new, output-sensitive algorithm which computes the shortest fence design. This algorithm runs in time $O(kn \log n)$, with k the number of fences in the design. We can adopt this output sensitive algorithm to compute modular fence designs in similar time bounds.

In the first part of the paper, we assume zero friction between the part and the fences. Later, we incorporate Coulomb friction (see e.g. [15]) between the part and the fence. The friction between the part and the belt is assumed to be uniform. We assume quasi-static motion, i.e. the belt speed is low enough so the parts do not bounce when they hit a fence. Since any push action acts on the convex hull of the part rather than on the part itself, we assume without loss of generality that the part under consideration is convex. The fences are assumed to be long enough so that the part rotates into a stable pose [17]. Furthermore, we assume that the the parts do not have meta-stable edges, i.e. the perpendicular projection of the center-of-mass on an edge does not intersect a vertex of the edge.

This paper is organized as follows. In Section 2, we first review the key notion of a push plan, and identify the constraints on the relative push angles for fence design. Section 3 reviews the $O(n^3 \log n)$ algorithm for computing the shortest fence design for a part. In Section 4 we present the output sensitive algorithm which runs in $O(kn \log n)$ time. In Section 5 we investigate frictional fences, and in Section 6 we look at modular fences. In Section 7 we draw some conclusions and give several interesting open questions.

2 Push Plans and Fence Designs

2.1 The push function

In this section we focus on the push function of a part. Let P a convex polygonal part with n vertices and center-of-mass c. We assume that a fixed coordinate frame is attached to P. Directions are expressed relative to this frame. The contact direction of a supporting line l of P is uniquely defined as the direction of the normal of l pointing into P. The radius function $r : [0, 2\pi) \to \mathbb{R}^+$ maps a direction ϕ onto the distance from c to the supporting line of P with contact direction ϕ. For a polygonal part, the radius function is

a continuous piecewise sinusoidal function, and can be computed in $O(n)$ time by checking each vertex [15]. The final orientation of a part that is being pushed can be determined from its radius function.

In most cases, parts will start to rotate when pushed. If pushing in a certain direction does *not* cause the part to rotate, then the contact normal at one of its points of contact with the jaw passes through the center-of-mass [15]. We refer to the corresponding direction of the contact normal as an *equilibrium* push direction or orientation. If pushing does change the orientation, then this rotation changes the orientation of the pushing device relative to the part. We assume that pushing continues until the part stops rotating and settles in a stable equilibrium pose, which is an equilibrium with a preimage of non-zero length.

The *push function* $p : [0, 2\pi) \rightarrow [0, 2\pi)$ links every orientation ϕ to the orientation $p(\phi)$ in which the part P settles after being pushed by a jaw with initial contact direction ϕ (relative to the frame attached to P). The rotation of the part due to pushing causes the contact direction of the jaw to change. The final orientation $p(\phi)$ of the part is the contact direction of the jaw after the part has settled. The equilibrium push directions are the fixed points of p.

The push function p of a polygonal part consists of steps, which are intervals $I \subset [0, 2\pi)$ for which $p(\phi) = C$ for all $\phi \in I$ and some constant $C \in I$. The steps of the push function are easily constructed from the radius function r. If the part is pushed in a direction corresponding to a point of non-horizontal tangency of the radius function then the part will rotate in the direction in which the radius decreases. The part finally settles in an orientation corresponding to a local minimum of the radius function. As a result, all points in the open interval I bounded by two consecutive local maxima of the radius function r map onto the orientation $\phi \in I$ corresponding to the unique local minimum of r on I. (Note that ϕ itself maps onto ϕ because it is a point of horizontal tangency.) This results in the steps of the push function. In preparation for the next sections, we define two open intervals $l(v) = \{\phi < v | p(\phi) = v\}$ and $r(v) = \{\phi > v | p(\phi) = v\}$ for each fixed point—or equilibrium orientation—v of

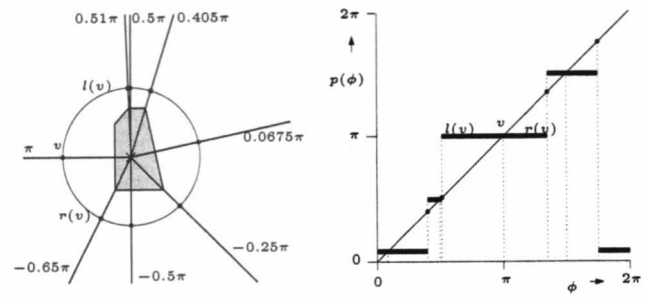


Figure 2: *A polygonal part and its push function. The minima correspond to normals to polygon edges that intersect the center-of-mass. The maxima correspond to tangents to polygon vertices whose normals intersect the center-of-mass. The horizontal steps of the push function are angular intervals between two successive maxima.*

the push function. We refer to these intervals as v's left and right environment respectively. The interval $l(v)$ corresponds to the half-step left of $v = f(v)$ and $r(v)$ corresponds to the half-step right of $v = f(v)$ (see Figure 2). Note that an equilibrium v is stable if $l(v) \cup r(v) \neq \emptyset$. We denote by $\mathcal{A} = a_0, \ldots, a_{m_s-1}$ the cyclic sequence of stable equilibria, cut at some arbitrary orientation, and ordered by increasing angle. Besides the steps, there are isolated points satisfying $p(\phi) = \phi$ in the push function, corresponding to local maxima of the radius function. Figure 2 shows a polygonal part and its push function. The final orientation of the part while being pushed is described by the push function. In [3] we showed that any step function having only non-zero-length half-steps represents a polygonal part, and is therefore a valid push function.

A push function p is said to have period d if $p(\phi) = p((\phi + d) \bmod 2\pi)$ for all $\phi \in [0, 2\pi)$. Any part can only be oriented up to (periodic) symmetry in its push function. The sequence $\mathcal{A}$ of stable equilibra is said to have cycle d if and only if $|r(a_i)| = |r(a_{i+d})|$, and $|l(a_i)| = |l(a_{i+d})|$ for all $0 \leq i < m_s$. (Indexing is modulo m_s.)

We use the abbreviation p_α to denote the (shifted) push function defined by

$$p_\alpha(\phi) = p((\phi + \alpha) \bmod 2\pi),$$

for all $\phi \in [0, 2\pi)$. Note that $p_\alpha(\phi)$ is the final orientation of a part in initial orientation ϕ after a reorientation by α followed by a push. We can now define a *push plan.*

Definition 1 *A* push plan *is a sequence* $\alpha_1, \ldots, \alpha_m$ *such that* $p_{\alpha_m} \circ \ldots \circ p_{\alpha_1}(\phi) = \Phi$ *for all* $\phi \in [0, 2\pi)$ *and some fixed* $\Phi \in [0, 2\pi)$.

An important property of the push function is monotonicity, or the order preserving property [16]. A sequence $s_1, \ldots, s_n$ of elements of a set S is *ordered* if $s_1, \ldots, s_n$ are encountered in order when the generating cycle of S is traced *once*, starting from s_1. A function $p : S \to S$ is *monotonic* if for any ordered sequence $s_1, \ldots, s_n$ the sequence $p(s_1), \ldots, p(s_n)$ is also ordered.

2.2 Fence design

In this section we address the problem of designing a series of fences $f_1, \ldots, f_m$ that will orient P when it moves down a conveyor belt and slides along these fences $f_1, \ldots, f_m$. Let us assume that the conveyor belt moves from top to bottom, as indicated in the overhead view in Figure 4. We distinguish between left fences, which are placed along the left belt side, and right fences, which are placed along the right side. The fence angle β_i of a fence f_i denotes the angle between the upward pointing vector opposing the motion of the belt and the normal to the fence with a positive component in upward direction. The motion of the belt turns the sliding of the part along a fence into a push by the fence. The direction of the push is—by the zero friction assumption—orthogonal to the fence with a positive component in the direction opposing the motion of the belt. Thus, the motion of the belt causes any push direction to have a positive component in the direction opposing the belt motion. We now transform this constraint on the push direction relative to the belt into a constraint on successive push directions relative to the part.

Sliding along a fence f_i causes one of P's edges e to align with the fence. The curved tip of the fence [7] guarantees that e is aligned with the belt sides as P leaves the fence. If f_i is a left fence then e faces the left belt side (see Figure 3). If f_i is a right fence, it faces the right side. Assume f_i is a left fence. At the moment of leaving f_i, hence, after the push, the contact direction of f_i is perpendicular to the belt direction and towards the right belt side. So, the reorientation of the push is expressed relative to this direction. Figure 3 shows that the reorientation α_{i+1} is in the range $(0, \pi/2)$ if we choose f_{i+1} to be a left fence. If we take a right fence f_{i+1} then the reorientation is in the range $(\pi/2, \pi)$. A similar analysis can be done when P leaves a right fence and the edge e faces the left belt side. The result is given in the following table.

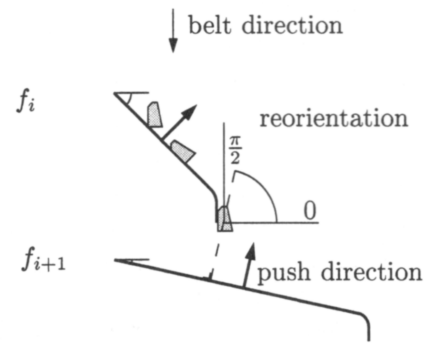


Figure 3: *For two successive left fences, the reorientation of the push direction lies in the range* $(0, \pi/2)$.

f_i	$\alpha_{i+1} \in I_{f_i, f_{i+1}}$	f_{i+1}
left	$(0, \pi/2)$	left
left	$(\pi/2, \pi)$	right
right	$(-\pi, -\pi/2)$	left
right	$(-\pi/2, 0)$	right

The table shows that the type of fence f_i imposes a bound on the reorientation α_{i+1}. Application of the same analysis to fences f_{i-1} and f_i and reorientation α_i leads to the following definition of a valid fence design [21].

Definition 2 *A* fence design *is a push plan* $\alpha_1, \ldots, \alpha_m$ *satisfying for all* $1 \leq i < m$*:*

- $\alpha_i \in (0, \frac{\pi}{2}) \cup (-\pi, -\frac{\pi}{2}) \Rightarrow \alpha_{i+1} \in (0, \frac{\pi}{2}) \cup (\frac{\pi}{2}, \pi)$
- $\alpha_i \in (-\frac{\pi}{2}, 0) \cup (\frac{\pi}{2}, \pi) \Rightarrow \alpha_{i+1} \in (-\frac{\pi}{2}, 0) \cup (-\pi, -\frac{\pi}{2})$.

The push plan on the left in Figure 4 satisfies the constraints of Definition 2, and is therefore also a fence design.

The relation between push plans and fence design immediately gives a worst case lower bound on the length of fence designs of $\Omega(n)$, which is presented in the paper of Chen and Ierardi [9]. The left and right environments of the stable equilibria of the parts of [9] which lead to this lower bound have the same length, with only a few exceptions. In their paper, Chen and Ierardi also give a linear upper bound on the length of a general push plan. Unfortunately, in general, this upper bound does not carry over to fence design, since the reorientations of the jaw do not satisfy the constraints of fence design. For a large class of parts, of which the lengths of the left and right environments are unique to some extent, the linear upper bound is valid for fence design, though.

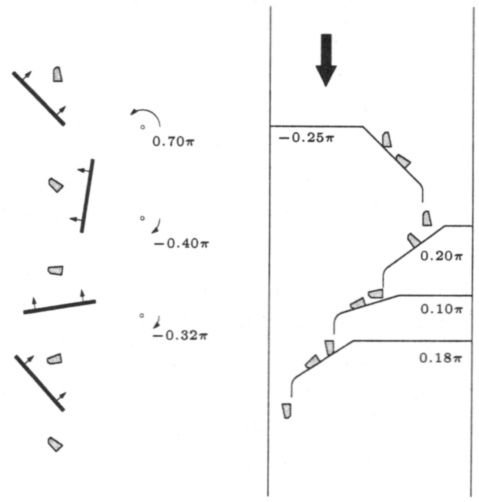


Figure 4: *The left picture shows a plan for a pushing jaw. The jaw reorientations are (from top to bottom)* 0.70π, -0.40π, *and* -0.32π. *The conveyor belt on the right orients the same part. The fence angles, which are measured relative to the upward vector opposing the direction of belt motion, are* -0.25π, 0.20π, 0.10π, *and* 0.18π. *(Note that the sequence of fence orientations is not the same as the sequence of orientations of the pushing jaw because the curved fence tip rotates the stable edge to become aligned with the belt direction.)*

3 An Algorithm for Computing Fence Designs

In this section we give an algorithm [4] which enables us to compute fence designs in time $O(n^3 \log n)$. This algorithm is based upon a discretization of the fence design problem, which leads to a polynomial-size graph representing the fence design problem. A search in this graph gives a feasible fence design. The algorithm is used as a basis for Sections 4 to 6. In the Section 3.1, we discuss the algorithm. In Section 3.2 its implementation.

3.1 A graph based algorithm

As every fence puts the part in a *stable* equilibrium orientation, the part is in one of these $m_s = O(n)$ orientations as it travels from one fence to another. This transforms the fence design problem into a discrete problem. After a first fence, the part can be in any of the stable orientations $a_0, \ldots, a_{m_s-1}$. The problem is to reduce the set of possible orientations of P to one stable orientation a_i by a sequence of fences. We build a directed graph on all possible *states* of the part as it travels from one fence to a next fence. A state consists of a set of possible orientations of the part plus the type (left or right) of the last fence, as the latter imposes a restriction on the reorientation of the push direction. Although there are $O(2^{m_s})$ subsets of $\{a_0, \ldots, a_{m_s-1}\}$, we shall see that we can restrict ourselves to subsets consisting of intervals of adjacent stable equilibria. Any such interval can be represented by the closed interval s defined by the first and last stable orientation a_i and a_j of the interval. The resulting graph has $O(m_s^2)$ nodes.

Consider two graph nodes (s, t) and (s', t'), where $s = [a_i, a_j]$ and s' are intervals of stable equilibria and t and t' are fence types. Let $I_{t,t'}$ be the open interval (of length $\pi/2$) of reorientations admitted by the successive fences of types t and t' according to the table in Subsection 2.2. There is a directed edge from (s, t) to (s', t') if there is an angle $\alpha \in I_{t,t'}$ such that a reori-

entation of the push direction by α followed by a push moves any stable orientation in s into a stable orientation in s'. To check this condition, we determine the preimage $(u, w) \supseteq s'$ of s' under the (monotonic) push function. Observe that if $|s| = a_j - a_i < w - u$, any reorientation in the open interval $r = (u - a_i, w - a_j)$ followed by a push will map s into s'. We add an edge from (s, t) to (s', t') if the intersection of r and the interval $I_{t,t'}$ of admitted reorientations is non-empty, and label this edge with $r \cap I_{t,t'}$. For convenience, we add a source and a sink to the graph. We connect the source to every node (s, t) in the graph for which s contains all m_s stable equilibria, and we connect every node (s, t) with $s = [a_i, a_i]$ to the sink, if the part has no symmetry in its push function. If the parts push function is symmetric, then the parts stable orientations have a cycle. Let d denote the cycle of the parts push function. The part is oriented if the interval of possible orientations is $[a_i - d, a_i]$, for any $i \in [0, \ldots, m_s]$ (indexing is modulo m_s). For symmetric parts, we connect these nodes to the sink. Every path from the source to the sink now represents a fence design; a fence design of minimum length corresponds to the shortest such path and can be found by a breadth-first search from the source.

Theorem 3 *The shortest path in the graph corresponds to the shortest fence design, if a fence design exists.*

Proof: We prove that each path corresponds to a fence design and vice versa. The theorem follows immediately if we realize that every edge in the path corresponds to a fence in the design.

($\Rightarrow$) If there is a path, we must prove that there is a corresponding fence design. Since there is an edge from (s, t) to (s', t') if and only if the successive fences t and t' allow for a reorientation that maps s into s', this follows immediately from the construction of the graph.

($\Leftarrow$) If there is a fence design, we prove there is a path in the graph that represents this fence design. Let $f_1, \ldots, f_N$ be a fence design. We track the possible orientations of the fence design, and prove by induction that for every set of possible orientations, there is a node in the graph, and furthermore, there is a path from the source to such a node. Let A_i denote the set of all possible orientations of P leaving f_i. It is easy to see that for each A_i there are multiple nodes that include the set of possible orientations.

After the first fence f_1, all stable orientations are possible. Since we added edges from the source to all nodes containing all stable orientations, these nodes are reachable.

We now assume that for fence f_i having type t in our fence design the nodes (s, t) with $s \supseteq A_i$ are reachable from the source. Let t' be the fence type of f_{i+1}. Let (s', t') be a node such that $s' \supseteq A_{i+1}$. Let (u, w) denote the preimage of s'. Since the push function is monotonic and by existence of the fence design which maps A_i onto A_j, there is an admitted reorientation α_{i+1} by f_{i+1} such that $(u - \alpha_{i+1}, w - \alpha_{i+1}) \supset A_i$. Therefore, let (s, t) be a node such that $(u - \alpha_{i+1}, w - \alpha_{i+1}) \supset s \supseteq A_i$. There is an edge from (s, t) to (s', t'), and there is a path from the source to (s, t). Since $s' \supseteq A_{i+1}$ is arbitrary, all (s', t') with $s' \supseteq A_{i+1}$ are reachable from the source. □

An important observation is that some graph edges are redundant if we are just interested in a fence design of minimum length. Consider a node (s, t) and all its outgoing edges to nodes $(s' = [a_i, a_j], t')$ for a fixed left endpoint a_i and a fixed fence type t'. We show that only one of these outgoing edges is sufficient. The following lemma is the key to this result.

Lemma 4 *Let (s, t), (s', t'), and (s'', t') be nodes with $s' = [a_i, a_j]$ and $s'' = [a_i, a_k]$ and let $s' \supset s''$. If there are edges from (s, t) to both (s', t') and (s'', t'), then the edge from (s, t) to (s', t') can be deleted without affecting the length of the shortest path in the graph.*

Proof: Assume that τ is a path from source to sink containing the edge $((s, t), (s', t'))$ and assume $((s', t'), (s''', t'''))$ succeeds this edge in τ. Because $s' \supset s''$, there must also be an edge $((s'', t'), (s''', t'''))$ in the graph. Hence, we can replace the edges $((s, t), (s', t'))$ and $((s', t'), (s''', t'''))$ in τ by $((s, t), (s'', t'))$ and $((s'', t'), (s''', t'''))$ without affecting the length of τ. □

The repeated application of Lemma 4 to the graph (until no more edges can be deleted) leads to a reduced graph in which every node has just one outgoing edge per set of nodes with intervals with a common left endpoint and with a common fence type. The single edge from the initial graph that remains after the repeated application of Lemma 4 is the one to the node corresponding to the shortest interval. Since there are $O(m_s) = O(n)$ possible left endpoints and just two fence types, the number of outgoing edges from one node is reduced to $O(n)$. The total number of edges of the reduced graph is therefore $O(n^3)$.

Theorem 5 *The reduced graph contains $O(n^2)$ nodes and $O(n^3)$ edges.*

The (reduced) graph can be constructed in the following way. First we compute the push function and store it in such a way that preimages can be found in $O(1)$ time. For each node (s,t), left endpoint a_i, and fence type t', we must determine the shortest interval $s' = [a_i, a_j]$ such that an edge exists between (s,t) and (s',t'). We can do this by a binary search on j. Since checking whether an edge exists between a pair of nodes corresponds to computing the preimage of an interval (which can be done in constant time), this binary search takes $O(\log n)$ time. As a similar binary search must be performed for each combination of a node (s,t), a left endpoint a_i, and a fence type t', the total time required to compute the graph edges is $O(n^3 \log n)$. A breadth-first search of the graph takes $O(n^3)$ time (see e.g. [10]). This yields the following theorem.

Theorem 6 *The computation of the optimal fence design for a polygonal part P with n vertices takes $O(n^3 \log n)$ time.*

Let τ be a path in the graph from the source to the sink. Every edge of τ corresponds to a non-empty angular interval of possible reorientations of the push direction. We simply pick the midpoint of every such interval as the reorientation, and get a push plan which is a fence design. We can easily compute the fence angles from the reorientation angles on the path, using the properties of the fence framework [21].

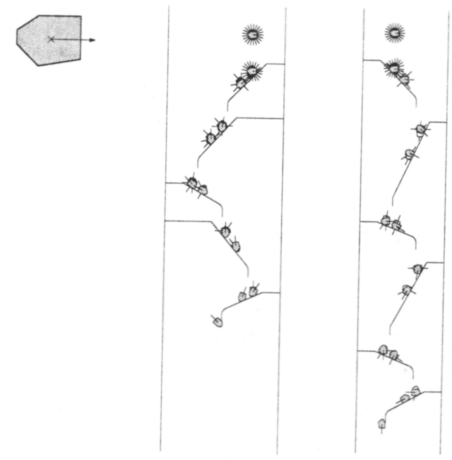


Figure 5: *An optimal design of five fences, and a design of six alternating fences allowing for a narrower belt. Every line segment emanating from the part represents a possible orientation of the part.*

3.2 Implementation

We implemented the above algorithm to test its behavior in practice. This turned out to be rather easy, using only some basic geometric computations for the push function, and some standard graph algorithms. The resulting code is rather fast; it returned fence designs within a fraction of a second for all parts we tried. (It should be noted that all parts we used here have only ten edges. When the number of edges goes up the cubic behavior of the algorithm will blow up the time required.) All fence designs shown in this paper were generated by the program. Our implementation offers the user the additional possibility of adding costs to graph edges. By doing so, the user can prevent the algorithm from outputting certain types of fence designs. Assigning high costs to edges between any pair of nodes of the same fence type t, for example, will cause the algorithm to output a sequence of alternating (left and right) fences if such a sequence exists. Alternating sequences are often preferred over sequences containing cascades of left (or right fences), as they generally allow for narrower conveyor belts (see Figure 5). Different cost assignments can be found to prevent e.g. unwanted steep and shallow fences. The costs make it

impossible to apply Lemma 4 to reduce the graph size, as this may cause the removal of equally long or longer paths with lower cost from the graph. The size of the resulting graph is therefore $O(n^4)$. Dijkstra's algorithm (see e.g. [10]) has been used to find the minimal cost path through the graph in time $O(n^4)$.

4 An Output-Sensitive Algorithm

A disadvantage of the algorithm of the previous section is the high running time of $O(n^3 \log n)$, even if the computed fence design turns out to be short. In [4, 20] we showed that, for a large class of parts, the fence designs have linear length, and for most long and thin parts, the fence design has constant length. This suggests that an output sensitive algorithm, whose running time depends on the length, is to be preferred. In this section we present such an algorithm which calculates the shortest fence design in $O(kn \log n)$ time, with n the combinatorial complexity of the part and k the number of fences of the computed design. Using this algorithm, most fence designs can be computed in $O(n^2 \log n)$, or even $O(n \log n)$ time.

4.1 Maintaining the best intervals

The main idea of the algorithm is to maintain the shortest interval of possible orientations after k fences, instead of precomputing the whole graph of all possible intervals of orientations. This is basically the same technique as used by Goldberg's algorithm to compute push plans [13]. Goldberg maintains the interval of possible orientations, and greedily shrinks this interval per application of the pushing jaw. We, however, must take into account the constraints of fence design. It is not sufficient to maintain a single shortest interval of possible orientations. Lemma 4 indicates that it is sufficient to maintain for each stable orientation the shortest interval starting at this orientation. Also, to be able to compute the possible reorientation of the jaw, we have to have two copies, one for the shortest interval with a plan that ends with a left fence, and another copy for a plan ending with a right fence. We denote the last orientation in the interval starting with a_i on a fence of type t after k fences by $a_i^t(k)$. Let us denote the sets containing all intervals after k fences ending with a left or a right fence by $\mathcal{L}(k)$ and $\mathcal{R}(k)$ respectively. After one fence, we have the following sets of shortest intervals:

$$\begin{aligned}
\mathcal{L}(1) &= \{[a_0, a_0^{\mathcal{L}}(1)], [a_1, a_1^{\mathcal{L}}(1)], \ldots, [a_{m_s}, a_{m_s}^{\mathcal{L}}(1)]\} \\
&= \{[a_0, a_{m_s}], [a_1, a_0], \ldots, [a_{m_s}, a_{m_s-1}]\} \\
\mathcal{R}(1) &= \{[a_0, a_0^{\mathcal{R}}(1)], [a_1, a_1^{\mathcal{R}}(1)], \ldots, [a_{m_s}, a_{m_s}^{\mathcal{R}}(1)]\} \\
&= \{[a_0, a_{m_s}], [a_1, a_0], \ldots, [a_{m_s}, a_{m_s-1}]\}
\end{aligned}$$

We shall now explain how to compute the sets of intervals $\mathcal{L}(k)$ and $\mathcal{R}(k)$, given $\mathcal{L}(k-1)$ and $\mathcal{R}(k-1)$. We show how to compute the intervals in $\mathcal{L}(k)$, since computing the intervals in $\mathcal{R}(k)$ is analogous. Let us consider the interval starting with orientation a_i and suppose that we want to compute the orientation $a_i^{\mathcal{L}}(k)$ such that the length of $[a_i, a_i^{\mathcal{L}}(k)]$ is minimal. Since the last fence we want to use is a left fence, we are allowed to change the push direction by angles in $(0, \frac{\pi}{2})$ and $(\frac{\pi}{2}, \pi)$, if the $k-1$'th fence was a left or right fence respectively. This constrains the possible alignment of the previous calculated shortest intervals in $\mathcal{L}(k-1)$ and $\mathcal{R}(k-1)$ with a_i. The preimage of a_i under the push function is $(a_i - |l(a_i)|, a_i + |r(a_i)|)$. Only intervals which can be mapped onto this preimage can possibly result in an interval of the form $[a_i, b]$ and therefore are *candidates* for the shortest interval $[a_i, a_i^{\mathcal{L}}(k)] \in \mathcal{L}(k)$. We have to determine the best interval among the candidate intervals from $\mathcal{L}(k-1)$ and $\mathcal{R}(k-1)$, i.e. the interval with the shortest possible image, which has to start with a_i, under the shifted pushfunction.

A single interval's image which starts with a_i can vary in length, depending on the position of the interval's left endpoint before applying the (shifted) push function. Any position of this left endpoint in $l(a_i) \cup \{a_i\} \cup r(a_i)$ results in an image starting with a_i. The image is shortest, though, when aligned as close as possible to $a_i - |l(a_i)|$. Therefore, our first goal is to determine the intervals which can align with $a_i - |l(a_i)|$. Also, we want to determine intervals which can be mapped onto angles in $(a_i - |l(a_i)|, a_i + |r(a_i)|)$. These two sets of intervals form the afore-mentioned candidates for the shortest interval $[a_i, a_i^{\mathcal{L}}(k)] \in \mathcal{L}(k)$. The candidate of which the right endpoint, after proper alignment with a_i, is leftmost determines the shortest

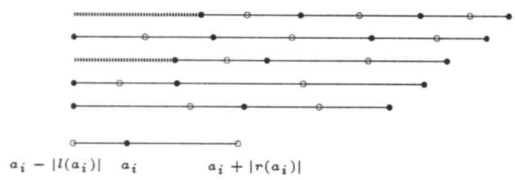


Figure 6: *The candidate intervals for $[a_i, a_i^{\mathcal{L}}(k)]$. The two intervals with the trailing dashed line cannot be mapped onto $a_i - |l(a_i)|$ by a valid fence design. The best interval is the interval with the leftmost right endpoint in the figure.*

interval $[a_i, a_i^{\mathcal{L}}(k)] \in \mathcal{L}(k)$, and is therefore the best candidate. The position of the right endpoint of a candidate interval is dependent on the length of the interval, as well as on the leftmost position onto which we can possibly map the left endpoint of this interval with a valid change of the push direction. Figure 6 depicts the orientation a_i together with five candidate intervals, of which two cannot be mapped onto $a_i - |l(a_i)|$. The alignment of the intervals closest to $a_i - |l(a_i)|$ is depicted in the picture. The two dashed lines correspond to the difference between $a_i - |l(a_i)|$ and the best alignment of the intervals.

A naive, but simple, algorithm to compute the sets $\mathcal{L}(k)$ and $\mathcal{R}(k)$ is to:

1. Test for each pair of an orientation a_i and a fence type, all intervals in $\mathcal{L}(k-1)$ and $\mathcal{R}(k-1)$ and check if there is a valid reorientation of the jaw such that these intervals align with $a_i - |l(a_i)|$, or possibly map onto $(a_i - |l(a_i)|, a_i + |r(a_i)|)$ otherwise.

2. Determine the interval with the smallest possible right end point with respect to its best alignment with $a_i - |l(a_i)|$.

3. Compute the image of the push function of this interval and store this image in appropriate set $\mathcal{L}(k)$ or $\mathcal{R}(k)$.

It takes linear time to determine the best best candidate per interval. Therefore, this naive approach leads to an $O(kn^2)$ algorithm.

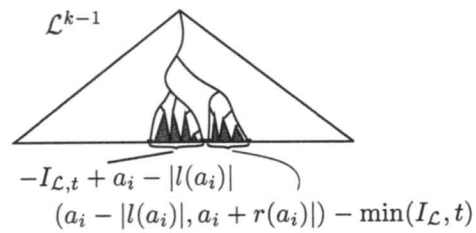


Figure 7: *The two queries from the range tree $\mathcal{L}(k-1)$ for $a_i^t(k)$.*

4.2 Updating in $O(n \log n)$ time

We now show how to speed up the part which determines the best candidate to logarithmic time per interval. To achieve this, we build a *range tree* (see e.g. [11]) on the starting orientations of the intervals in $\mathcal{L}(k-1)$ and $\mathcal{R}(k-1)$, which allows us to find in logarithmic time sets of the candidate intervals for any starting orientation after k fences. A range tree is a balanced binary search tree. The leaves of the tree contain the intervals of the previous step, ordered on starting orientation. The nodes of the tree contain splitting values, which guide the search. The subset of keys which are contained in the subtree is referred to as the *canonical subset* of this subtree. Let us store in each node of the range tree, the position of the left-most right endpoint of the canonical subset of intervals of the subtree rooted at this node, and furthermore, the length of the shortest interval of the same canonical subset. We can now easily derive the candidate interval with the leftmost right-interval, thereby reducing the number of candidates to $O(\log n)$, as we shall outline in the next three paragraphs.

Let us first try to understand which range queries are necessary and sufficient to compute a new interval. Assume that we want to compute $a_i^t(k)$, i.e. the right endpoint of the shortest interval starting with a_i after k fences; the k'th fence being of type t. We first analyze which intervals from $\mathcal{L}(k-1)$ align with $a_i - |l(a_i)|$, and which intervals map onto $(a_i - |l(a_i)|, a_i + |r(a_i)|)$. The interval from $\mathcal{L}(k-1)$ must be reoriented by an angle in $I_{\mathcal{L},t}$—$I_{\mathcal{L},t}$ being the angular interval of possible reorientations of the jaw by a left fence and a fence of type

t, see the table in Section 2.2. Intervals $[a_j, a_{j'}]$ with $a_j \in a_i - |l(a_i)|$ minus angles in $I_{\mathcal{L},t}$ can be mapped onto $a_i - |l(a_i)|$. If, on the other hand, the minimal angle $\min(I_{\mathcal{L},t})$ in $I_{\mathcal{L},t}$, cannot map a_j onto $a_i - |l(a_i)|$, a_j can be mapped onto $(a_i - |l(a_i)|, a_i + |r(a_i)|)$, if $a_j + \min(I_{\mathcal{L},t}) \in (a_i - |l(a_i)|, a_i + |r(a_i)|)$. Querying from $\mathcal{R}(k-1)$ is similar. Therefore, we query as follows from $\mathcal{L}(k-1)$ and $\mathcal{R}(k-1)$:

The intervals which align with $a_i - |l(a_i)|$ correspond to two range queries of length $\frac{\pi}{2}$; $-I_{\mathcal{L},t} + a_i - |l(a_i)|$ in $\mathcal{L}(k-1)$, and $-I_{\mathcal{R},t} + a_i - |l(a_i)|$ in $\mathcal{R}(k-1)$. Since these intervals align with $a_i - |l(a_i)|$, the shortest such interval (which is stored with the nodes) is the candidate per canonical subset. There is a logarithmic number of canonical subsets, and consequently a logarithmic number of candidates.

The intervals which can only be mapped onto $(a_i - |l(a_i)|, a_i + |r(a_i)|)$ receive a penalty which is added to the length of this interval. In the end, the interval which has the leftmost right endpoint is the best candidate of a canonical subset. There is a minor problem with calculating the leftmost right endpoint of a set of intervals. Since the push function is periodic, the leftmost point is dependent on the choice of the start of the 2π wraparound. The penalty of two consecutive intervals might differ 2π plus the desired difference. To overcome this problem, we compute two copies of the push function and store the both of them in the range tree—in the second copy of the push function the values are 2π higher than in the first copy. This way, any range of intervals can be found as one piece of the range tree. We query $(a_i - |l(a_i)|, a_i + |r(a_i)|) - \min(I_{\mathcal{R},t})$ in $\mathcal{R}(k-1)$ and $(a_i - |l(a_i)|, a_i + |r(a_i)|) - \min(I_{\mathcal{L},t})$ in $\mathcal{L}(k-1)$.

To overcome wraparound problems, we make sure that $a_i + |r(a_i)| > a_i - |l(a_i)| > \min(I_{\mathcal{R},t})$, by adding 2π to some of the values, if necessary. We derive a logarithmic number of canonical subsets. The values of their leftmost right endpoints are too high to compete with the lengths of the candidates of the former two queries. We want the difference between $a_i - |l(a_i)|$, and the right endpoint. Therefore, we subtract $a_i - |l(a_i)|$ from the found right endpoints, and derive a logarithmic number of appropriate candidates.

In Figure 7 the queries from $\mathcal{L}(k-1)$ for $a_i^t(k)$ are depicted. The query on the left corresponds to intervals of which the starting orientation can be mapped onto $a_i - |l(a_i)|$. The length of the shortest interval of the canonical subsets of this query determines the best candidate for this query. The query on the right corresponds to intervals of which the starting orientation can be mapped onto $(a_i - |l(a_i)|, a_i + |r(a_i)|)$. The difference of the leftmost right endpoint of the canonical subsets of this query and $a_i - |l(a_i)|$ determines the best candidate for this query. Together with the corresponding queries from $\mathcal{R}(k-1)$, the shortest interval starting with a_i on a fence of type t is determined. The total number of candidates resulting from the four range queries is $O(\log n)$. The best candidate of the four queries is the interval which determines the value of $a_i^t(k)$, and is computed in $O(\log n)$ time. This leads to the following lemma.

Lemma 7 *Let there be given two range trees; one in which two copies on two folds of the unit circle of the intervals in $\mathcal{L}(k-1)$ are stored; and one for the intervals in $\mathcal{R}(k-1)$, stored in a similar way. Both trees are ordered on the left endpoints of the intervals. If in the nodes the shortest interval as wel as the leftmost right endpoint of the corresponding canonical subsets are stored, then, the sets $\mathcal{L}(k)$ and $\mathcal{R}(k)$ can be computed in $O(n \log n)$ time.*

The building time of the interval tree is $O(n \log n)$ time. First, we presort the intervals on their length, and their right endpoint. Then, we build the range-tree buttom up in linear time. The query takes $O(\log n)$ per calculated new shortest interval. We have $O(n)$ new intervals. Per fence we spend $O(n \log n)$ time.

4.3 Incrementally computing the fence design

We want to use the described method to compute fence designs. This implies that the algorithm has to terminate when the part is oriented up to symmetry. If the part has no rotational symmetry, then the part is oriented if there is one stable orientation in an interval of possible orientations. If, on the other hand the part has rotational symmetry, then we have to detect if the part is oriented up to symmetry. This can be done

in constant time as well, by comparing the indices of the computed intervals. We can easily precompute the desired difference of the indices.

Also, we want to know which fence design corresponds to our finally derived interval which corresponds to the oriented part. This can be accomplished by incrementally building the same graph as in Section 3, storing the used fences with the edges of the graph. We can find the fence design, by tracking back the path from the 'oriented' node to the source of the graph. The size of the graph increases by $O(n)$ per step of the algorithm. Tracking back the path takes $O(k)$ time.

The algorithm, therefore, runs in $O(kn \log n)$ time, k the number of steps of the algorithm. The following theorem summarizes the result of this section.

Theorem 8 *Let P be a polygonal part. A fence design which orients P up to symmetry can be computed in $O(kn \log n)$ time, with n the combinatorial complexity of P, and $k = O(n^2)$ the number of fences in the resulting design.*

Proof: A fence design which orients P up to symmetry exists [4, 3]. We can, by subtracting the indices of a current shortest interval, check in constant time if the part is oriented up to symmetry. Extending the fence design by one fence takes $O(n \log n)$ time. Maintaining the used fences takes $O(n)$ time per extension. The total running time is, therefore, $O(kn \log n)$ time, with k the number of fences in the design. □

Although we did not actually implement this output-sensitive algorithm, the algorithm is more or less as easy to implement as the graph based algorithm. The main difference between the two algorithms is the use of a range tree in the latter, which is basically a binary search tree. Another difference is the incremental construction of the graph, which is not any harder than precomputing the nodes and the edges. We believe the resulting code of the algorithm of this algorithm is faster than the code of the previous section, provided that any part can be oriented by a linear length fence design.

5 Friction

Up till now, we assumed that there was no friction between the fences and the part. In this section, we shall incorporate friction into the model.

5.1 The model

There are several ways to look at friction. We will use the simplified model of Coulomb friction (see e.g. [15]). Imagine a part which slides along a fence on a stable edge. The frictional force is a force opposite to the motion of the part. According to Coulomb's law, the frictional force is a fixed fraction of normal force acting on the part—the normal force is induced by the belt which presses the part onto the fence. This fraction is given by the *coefficient of friction* μ. A simple statement of Coulomb's law for a moving part is:

$$|f_{friction}| = \mu |f_{normal}|$$

If $\mu|f_{normal}|$ is larger than the resulting force in the direction of the part, then the part will remain at rest. The frictional force now balances the applied force by the belt, and Coulomb's law boils down to:

$$|f_{friction}| \leq \mu |f_{normal}|$$

Suppose we have a part on an fence, and we apply a downward force induced by the conveyor. In Figure 8, we have on the left, a part which remains at rest. The frictional force combined with the normal force cancel out the force of the belt. On the right, the part slides, the frictional force combined with the normal force to not balance the belt force. A graphical interpretation of the Coulomb friction is given by the cone with dihedral angle $2 \arctan \mu$. If the normal force combined with the frictional force is in the interior of the cone, the part remains stable, otherwise, the part will slide. It easy to see whether or not the combined normal and frictional force is in the cone. If we negate the belt force, we check whether or not this force is in the cone. In the former case, the part remains stable, in the latter, the part slides.

In his thesis, Mason [15] describes an easy test to predict the rotation of the part which has a vertex in

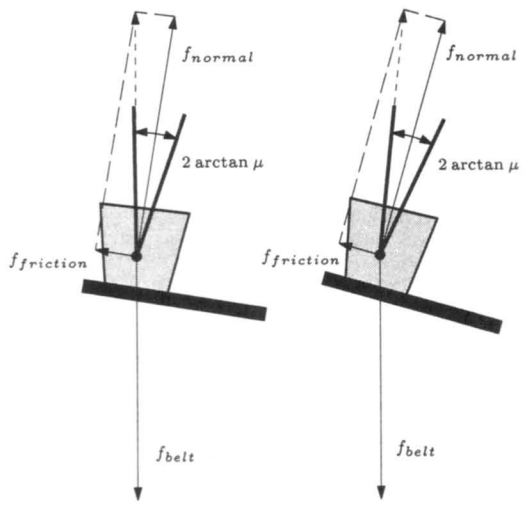


Figure 8: *A part on a fence with friction. Left, the part does not slide, due to friction. Right, the part slides.*

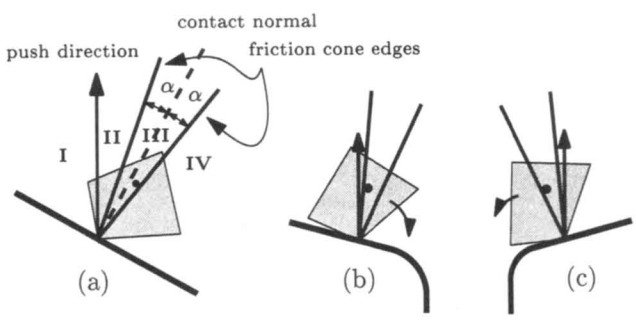


Figure 9: *(a) The rotation of the part in vertex-contact with the fence. (b) and (c) The same part with the same orientation with respect to the fence. However, due to friction, the left fence rotates the part to another edge than the right fence.*

contact with the fence. He gives an analysis which boils down to a very simple geometric check. Given the orientation of the part and the fence angle, we can draw three half lines emanting from the contact vertex. One is the line of pushing, which is opposite to the direction of the belt. The two other lines are the boundary edges of the friction cone. The area above the fence is now divided into four areas, which we enumerate from left to right (see Figure 9(a)). The position of the center-of-mass now determines the rotation of the part. In area's **I** and **II**, the part will rotate counterclockwisely. In area's **III** and **IV**, the part will rotate clockwisely. If the center-of-mass is on the line separating area **II** from area **III**, then the part is in unstable equilibrium, and will neither rotate clockwisely, nor rotate counterclockwisely. Though, the part might either remain in the same position if the separating line is the line of pushing, or slide towards the end of the fence, otherwise.

To determine whether or not an edge is stable we test for both vertices of the edge if they predict counterclockwise or clockwise rotation. If the vertices disagree, the edge is stable, otherwise, the edge is unstable and the part will rotate as predicted.

We now incorporate the developed theory into the fence design model. The first observation we make is that we cannot make fences which are very shallow, because these fences do no longer allow the parts to slide off the fences. The (absolute) minimum fence angle must be strictly greater than $\arctan \mu$. Under this assumption, the line which separates area **II** from area **III** is the left edge of the friction cone, if the part is on a left fence, and the right edge of the friction cone otherwise. This implies that the push function, which predicts the rotation of the part, is no longer symmetric. There are orientations for which a left fence rotates the part clockwisely, and a right fence rotates the part counterclockwisely (see Figure 9(b),(c)). However, the push function for a part on a left fence is still monotonous, as is the push function for a part on a right fence. Moreover, one could look at the frictional fence as a frictionless fence which is $\arctan \mu$ shallower. The normal of this shallower frictionless fence coincides with described edge of the friction cone. However, the steepest frictionless fence which corresponds to a frictional fence is $\frac{\pi}{2} - \arctan \mu$ for a right fence, and $\arctan \mu - \frac{\pi}{2}$ for a left fence. Unfortunately, this implies that frictional fences are not able to orient any part up to symmetry. It is possible to construct parts which can only be oriented with a reorientation of the push direction push arbitrarily close to π [1]. Since the reorientation of the jaw closest to π is $\pi - \arctan \mu$ for

a frictional fence, we can always construct a part which cannot be oriented by frictional fences.

5.2 Algorithms for frictional fence designs

In this section we discuss two algorithms to compute frictional fence designs, if one exists. Firstly, we discuss a modification of the graph based algorithm, and secondly we give a modification of the output sensitive algorithm. Let us first redefine the intervals of possible reorientations of the jaw for frictional fences.

f_i	$\alpha_{i+1} \in I^{\mu}_{f_i,f_{i+1}}$	f_{i+1}
left	$(0, \pi/2 - \arctan\mu)$	left
left	$(\pi/2, \pi - \arctan\mu)$	right
right	$(-\pi + \arctan\mu, -\pi/2)$	left
right	$(-\pi/2 + \arctan\mu, 0)$	right

This means that we can still use our graph based algorithm. However we have to modify the part which decides if a node can be connected to another node in the graph.

Consider two graph nodes (s,t) and (s',t'), where $s = [a_i, a_j]$ and s' are intervals of stable equilibria and t and t' are fence types. Let $I^{\mu}_{t,t'}$ be the open interval (of length $\pi/2 - \arctan\mu$) of reorientations admitted by the successive fences of types t and t' according to the table above. There is a directed edge from (s,t) to (s',t') if there is an angle $\alpha \in I^{\mu}_{t,t'}$ such that a reorientation of the push direction by α followed by a push moves any stable orientation in s into a stable orientation in s', and furthermore α is constructed by a frictional fence. To check this condition, we determine the preimage $(u,w) \supseteq s'$ of s', under the push function. Observe that if $|s| = a_j - a_i < w - u$, any reorientation in the open interval $r = (u - a_i, w - a_j)$ followed by a push will map s into s'. We add an edge from (s,t) to (s',t') if the intersection of r and the interval $I^{\mu}_{t,t'}$ of admitted reorientations is not empty. We label this edge with this intersection. Also, we add a source and a sink to the graph. We connect the source to every node (s,t) in the graph for which s contains all m_s stable equilibria, and we connect every node (s,t) with s an interval which is reduced up to symmetry to the sink. Every path from the source to the sink now represents a fence design; a fence design of minimum length corresponds to the shortest such path and can be found by a breadth-first search from the source.

Let τ be a path in the graph from the source to the sink. Every edge of τ corresponds to a non-empty angular interval of possible reorientations of the push direction. We pick the midpoint ϕ of every such interval, and determine if this reorientation is by left or a right fence. For a left fence, we add $\arctan\mu$ to the corresponding fence angle. For a right fence, we subtract $\arctan\mu$. We get a frictional fence design.

Theorem 9 *Let P be a polygonal part, with coefficient of friction μ with respect to the fences. There is an algorithm which computes a shortest fence design in time $O(n^3 \log n)$, if there is a fence design which orients P. If no fence designs exists, the algorithm detects failure in the same time bound.*

Also, we can modify the output-sensitive algorithm to compute the shortest fence design for frictional fences, if one exists. The modification by itself is quite straightforward. Instead of using the table of Section 2.2, we use the modified table of this section to determine the possible reorientations of the jaw to extend the fence designs. A problem, however, is to determine when to stop augmenting the fence designs. Therefore, we have to add an extra stop criterion to ensure the algorithm does not try to extend the fence design when it becomes clear that this no longer makes sense.

Let us concentrate on the array which stores the current set of possible intervals. During one step of the algorithm, we replace items of the array by shorter items. We argue that, if an interval was not replaced in a step of the algorithm, then we need not to compute the new intervals derived from this interval in the succeeding step. This is because intervals monotonously decrease, and the intervals which would have been derived from this interval in the succeeding step, were already computed in the former step. The second stop criterion is to stop when no interval can be improved on in a single step. Checking if an interval is shrunk in a step of the algorithm can be done by storing a flag which is reset at the start of a step, and set if the interval is improved. Checking if any interval needs improvement,

can be done on the fly. Storing and updating the flags does not influence the asymptotic running time. Since there are at most $O(n^2)$ intervals of stable equilibria, the algorithm terminates after at most $O(n^2)$ steps. This leads to an upper bound on the running time of $O(n^3 \log n)$. While improving the intervals, we also incrementally construct the graph corresponding to the found fence designs. At the end, deriving a frictional fence design from a path in the graph is accomplished in the same way as in the graph based method.

Theorem 10 *Let P be a polygonal part, having combinatorial complexity n. If there is a fence design that orients P, then a design can be computed in $O(kn \log n)$ time, with k the number of required fences, which orients the part up to symmetry. If no fence design exists, the algorithm reports failure in $O(n^3 \log n)$ time.*

If we could prove that any fence design has linear length, then this result would most likely carry over to the upper bound on the running time of the algorithm reporting failure. This would result in an upper bound of the running time of $O(n^2 \log n)$ of the frictional fence design algorithm.

6 Modular Fences

In this section, we discuss modular fence designs, i.e., designs for which the fence angles are to be taken from a discrete and finite set of angles. Modular fences can be useful in an industrial application of a conveyor to orient parts. There is no need for 'expensive' adjustable fences. The conveyor operator picks prefabricated fences out of a box and just attaches them along the sides of the belt. Figure 10 shows an example of a fence design constructed with modular fences of $\frac{\pi}{4}$ and $-\frac{\pi}{4}$, together with a fence design build with arbitrary fence angles. As is to be expected, the modular fence design requires more fences.

6.1 The model

We first discuss how the discretization of the possible fence angles influences the possible reorientations of the jaw, and therefore the algorithms to compute fence designs.

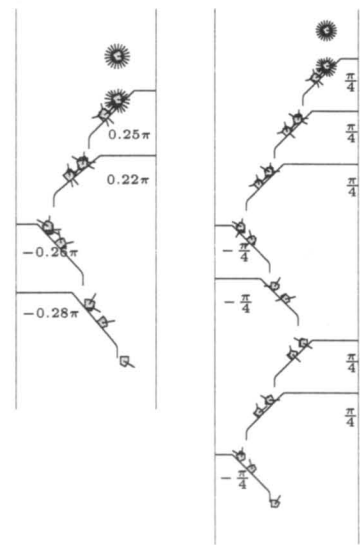


Figure 10: *Two fence designs for the same part. On the left, any fence angle is available. On the right, only fences of $\frac{\pi}{4}$ and $-\frac{\pi}{4}$ are used.*

Let m denote the number of fence angles. Every fence can correspond to two reorientations of the jaw. A left fence with fence angle $\phi \in (-\frac{\pi}{2}, 0)$ corresponds to a reorientation of the jaw of $\phi + \frac{\pi}{2}$ if it succeeds a left fence in the fence design, and $\phi - \frac{\pi}{2}$ if it succeeds a right fence. A right fence with fence angle $\phi \in (0, \frac{\pi}{2})$ corresponds to a reorientation of the jaw of $\phi + \frac{\pi}{2}$ if it succeeds a left fence in the fence design, and $\phi - \frac{\pi}{2}$ if it succeeds a right fence. The total number of possible reorientations of the jaw is bounded by $2m$.

Still, it is possible to construct parts which can only be oriented with a reorientation of the push direction push arbitrarily close to π [1]. This implies that one can always construct a part that cannot be oriented, given a set of modular fence angles.

The modular fences we treat in this section are friction free. Using the techniques of Section 5, one can easily modify the algorithms of this section to compute frictional modular fence designs as well.

6.2 Algorithms for modular fence designs

In this section we give two algorithms to compute modular fence designs (if they exist for the part). As in the section on frictional fences, the first algorithm is based on the graph based algorithm of Section 3, and the second is based on the output-sensitive algorithm of Section 4. However, the running times of the algorithms are surprisingly different. The former runs in $O(mn^2)$ and the latter in $O(mkn)$ time, m denoting the number of available fence angles, k the number of fences in the design. Since k can vary from $O(1)$ to $O(n^2)$ in the worst case, it is interesting to present both algorithms.

In the algorithms of this section, we want to be able to quickly compute the image under the push function of a value of the form $a_i + \alpha$, a_i a stable equilibrium, and α in the set of reorientations of the jaw by modular fences. Therefore, we precompute these values using the push function. The number of possible reorientations of the jaw is $2m$. These images are computed by tracing the push function, in time $O(mn)$. We store these values in a two dimensional array. We can now determine in constant time the value of $p(a_i + \alpha)$, for each stable equilibrium a_i and each modular reorientation of the jaw α.

The nodes and the edges of the graph based algorithm are encoded the same way as in Section 3. The graph is built in a somewhat different way than before, though. A node of the graph again corresponds to a state of the part, encoded as an interval of possible orientations. The edges between the nodes are added a bit differently than before though. Let (s, t) be a node in the graph, $s = [a_i, a_j]$ is an interval of stable equilibria and t is a fence type. Since each modular fence has a fixed angle, we can for *each* fence angle compute the image of s under the push function. If we add edges from (s, t) to the nodes in the graph corresponding to these images, Lemma 4 states that the resulting graph suffices to compute shortest paths in the graph, corresponding to shortest fence designs, although there might be redundant edges.

We compute the images as follows. For each modular fence f, having type t', we compute the reorientation of the jaw $\alpha_{t,f}$ between a fence of type t, and the fence f. The interval of orientations after appending fence f to a design is $s' = [p(a_i + \alpha_{t,f}), p(a_j + \alpha_{t,f})]$. We add a directed edge in the graph from (s, t) to (s', t'). We label this edge with this modular push angle. For each node, we spend $O(m)$ time computing the outgoing edges.

Also, we add a source and a sink to the graph. We connect the source to every node (s, t) in the graph for which s contains all m_s stable equilibria, and we connect every node (s, t) with s an interval which is reduced up to symmetry to the sink. Every path from the source to the sink now represents a fence design; a fence design of minimum length corresponds to the shortest such path and can be found by a breadth-first search from the source.

The building time and the size of the graph are $O(mn^2)$. The search through the graph takes $O(mn^2)$ as well. We can, therefore, compute a fence design in $O(mn^2)$ time, if one exists. If no fence design exists which orients the part up to symmetry, then there is no path from the source to the sink in the graph, which is detected after $O(mn^2)$ as well. In the given bounds, m is the number of available fence angles. If we assume that a constant number of fixed angles is available, then the running time is $O(n^2)$.

Theorem 11 *Let P be a polygonal part, having combinatorial complexity n. Let m be the number of different available fence angles. If there is a fence design that orients P, then a shortest fence design which orients this part can be computed in $O(mn^2)$. Otherwise failure is detected in the same time bound.*

We now present an output sensitive algorithm which computes a fence design in $O(mkn)$ time, with k the number of fences. The algorithm is based on the output-sensitive algorithm of Section 4. If $k = O(1)$, this algorithm is asymptotically faster than the previously presented algorithms.

We store $O(n)$ intervals of possible orientations in an array. These intervals are the shortest intervals reachable with a fence design which is incrementally constructed. We have to take into account the fence type, and store therefore for each tuple of a stable orientation

v and a fence type t the shortest interval starting with orientation v , with the last push by a fence of type t. When we augment the fence design with one fence, we can compute the new shortest intervals as follows. For each interval, we compute the m images of this interval. We check in the array of shortest intervals, if the new interval is shorter than the currently stored interval, and, if so, we replace the stored interval by the new interval. Looking up the image of an interval of stable orientations can be done in constant time, using our precomputed two-dimensional array with stored values of the push functions. Therefore, given the intervals after $k-1$ fences, computing the intervals after k fences is accomplished in $O(nm)$ time.

We stop when the part is oriented up to symmetry. This can be checked by comparing the indices of the intervals of possible orientations. Unfortunately, not every part can be oriented up to symmetry using a modular fence design. This means we have to add an extra stop criterion which determines if the intervals keep improving. We already discussed how to add this criterion without extra time overhead in Section 5, and we shall not repeat this discussion here, but just give the result in the following theorem.

Theorem 12 *Let P be a polygonal part, having combinatorial complexity n. Let m be the number of different available fence angles. If there is a fence design that orients P, then a design can be computed in $O(kmn)$, with $k = O(n^2)$ the number of required fences, which orients the part up to symmetry. If no fence design exists, the algorithm terminates in $O(mn^3)$ time.*

Since the best known upper bound on the length of a fence design is $O(n^2)$, the latter algorithm might be slower than the graph based algorithm. However, we believe that fence designs, like push plans [9], have at most linear length. It is an interesting open problem to prove or disprove this.

7 Conclusion

In this paper we addressed the algorithmic issues of sensorless part orientation by a sequence of fences across a conveyor belt. Frictionless fences can orient any polygonal part up to symmetry. In this paper, we gave several algorithms to compute fence designs. We first reviewed an easy to implement algorithm which runs in $O(n^3 \log n)$ time. This algorithm can be modified to compute fence designs which take into account various quality measures, such as avoidance of very steep or shallow fences. This modification increases the running time of the algorithm to $O(n^4)$.

The structure of the algorithm yields an $O(n^2)$ bound on the length of the shortest fence design. But for a large class of parts it is known that fence designs have linear length and for most long and thin parts, a feasible fence design consists of only a constant of number fences. In this paper, we presented the first output sensitive fence design algorithm which benefits from these bounds on the length of the plan. This algorithm runs in $O(kn \log n)$ time, with k the number of fences in the final design.

Furthermore, we gave algorithms to compute fence designs for frictional fences and modular fences, in which we allow only a discrete set of fence angles. The former turned out to be just a minor modification to the algorithms for computing frictionless fence designs. The latter, however, gave rise to a different approach to computing fence designs in time $O(mn^2)$, or $O(mkn)$, k the number of fences, m the number of fence angles. Unfortunately, not every part can be oriented with frictional or modular fences.

The results largely settle the algorithmic questions in computing fence designs, although some improvements in the running time might still be possible. The main interesting open problem left is the question which parts can be oriented with friction of with modular fences. Experiments show that for many parts a valid fence design still exists. An other open problem is the bound on the length of the fence design. Only an $O(n^2)$ upper bound is known at the moment. It is unknown whether this is indeed required ar whether linear length designs always exist. Also, looking into orienting algebraic parts [19] using fences is of interest. Finally, we want to mention the problems of dealing with uncertainty (see [8] for a first treatment) and generalization of fence design to 3D.

Acknowledgements

This research was supported in part by NATO Collaborative Research Grant CRG 951224. Berretty's research is supported by the Dutch Organization for Scientific Research (N.W.O.). Goldberg's research is supported in part by the National Science Foundation under IRI-9612491 and by Presidential Faculty Fellow Award IRI-9553197.

References

[1] S. Akella, W. Huang, K.M. Lynch, and M.T. Mason. Sensorless parts feeding with a one joint manipulator. In *IEEE International Conference on Robotics and Automation*, 1997. See also *Alg. for Robotic Motion and Manipulation* (J.-P. Laumond and M. Overmars (Eds.)), A.K. Peters, pages 229–238, 1996.

[2] S. Akella and M.T. Mason. Posing polygonal objects in the plane by pushing. In *IEEE International Conference on Robotics and Automation, Nice*, pages 2255–2268, 1992.

[3] R.-P. Berretty, K. Goldberg, M. Overmars, and A.F. van der Stappen. Computing fence designs for orienting parts. Technical report, UU-CS-1997-41, Dept. of Comp. Science, Utrecht University, 1997.

[4] R.-P. Berretty, K.Y. Goldberg, M.H. Overmars, and A.F. van der Stappen. On fence design and the complexity of push plan for orienting parts. In *Proc. 13th Ann. ACM Symp. on Computational Geometry.*, pages 21–29, 1997.

[5] K.-F. Böhringer, V. Bhatt, and K.Y. Goldberg. Sensorless manipulation using transverse vibrations of a plate. In *Proc. IEEE Int. Conf. on Robotics and Automation, Nagoya, Japan*, pages 1989–1996, 1995.

[6] K.-F. Böhringer, B.R. Donald, and N.C. MacDonald. Upper and lower bounds for programmable vector fields with applications to mems and vibratory plate part feeders. *Alg. for Robotic Motion and Manipulation,* J.-P. Laumond and M. Overmars (Eds.), A.K. Peters, pages 255–276, 1996.

[7] M. Brokowski, M.A. Peshkin, and K. Goldberg. Optimal curved fences for part alignment on a belt. *ASME Transactions of Mechanical Design*, 117, March 1995.

[8] J. Chen, K. Goldberg, M. Overmars, D. Halperin, K.-F. Böhringer, and Y. Zhuang. Shape tolerance in feeding and fixturing. em This proceedings, 1998.

[9] Y.-B. Chen and D.J. Ierardi. The complexity of oblivious plans for orienting and distinguishing polygonal parts. *Algoritmica*, 14:367–397, 1995.

[10] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms.* The MIT Press, Cambridge, Massachusetts; London, England, 1990.

[11] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry, Algorithms and Applications.* Springer, Berlin Heidelberg, 1997. ISBN 3-540-61270-X.

[12] M.A. Erdmann and M.T. Mason. An exploration of sensorless manipulation. *IEEE Journal of Robotics and Automation*, 4:367–379, 1988.

[13] K. Goldberg. Orienting polygonal parts without sensors. *Algoritmica*, 10(2):201–225, 1993.

[14] K. M. Lynch and M.T. Mason. Stable pushing: Mechanics, controllability, and planning. *Int. J. of Robotics Research*, 6(15):533–556, 1996.

[15] M. Mason. *Manipulator grasping and pushing operations.* PhD thesis, MIT, 1982. Published in *Robot Hands and the Mechanics of Manipulation*, MIT Press, Cambridge, MA, 1985.

[16] B.K. Natarajan. An algorithmic approach to the automated design of parts orienters. In *IEEE Annual Symposium on Foundations of Computer Science*, pages 132–142, 1986.

[17] M.A. Peshkin and A.C. Sanderson. The motion of a pushed sliding workpiece. *IEEE Journal of Robotics and Automation*, 4(6):569–598, 1988.

[18] M.A. Peshkin and A.C. Sanderson. Planning robotic manipulation strategies for workpieces that slide. *IEEE Journal of Robotics and Automation*, 4(5):524–531, 1988.

[19] A. Rao and K. Goldberg. Manipulating algebraic parts in the plane. *IEEE Transactions on Robotics and Automation*, 11, 1995.

[20] A.F. van der Stappen, K. Goldberg, and M. Overmars. Geometric eccentricity and the complexity of manipulation plans. Technical report, UU-CS-1996-49, Dept. of Computer Science, Utrecht University, 1996. To appear in *Algorithmica.*

[21] J. Wiegley, K. Goldberg, M. Peshkin, and M. Brokowski. A complete algorithm for designing passive fences to orient parts. *Assembly Automation*, 17(2):129–136, August 1997.

Taylor & Francis
Taylor & Francis Group
http://taylorandfrancis.com

Shape Tolerance in Feeding and Fixturing

Jingliang Chen, *University of California at Berkeley, USA*
Ken Goldberg, *University of California at Berkeley, USA*
Mark H. Overmars, *University of Utrecht, The Netherlands*
Dan Halperin, *Tel Aviv University, Isreal*
Karl F Böhringer, *University of California at Berkeley, USA*
Yan Zhuang, *University of California at Berkeley, USA*

Parts are not ideal. Designers and machinists cope with variation in part shape by specifying tolerance zones around a nominal part geometry: all parts that fit within the zone form a tolerance class. In this paper we consider the issue of shape tolerance in two contexts. First, we consider the problem of feeding *(orienting) convex polygonal parts on a conveyor belt with a part-specific sequence of fence angles [4]. Second, we consider the problem of* fixturing *convex polygonal parts using a right angle fixture and one clamp [15]. The challenge is to define appropriate tolerance classes and to argue that a solution – a feeding strategy or a fixture – is guaranteed for all parts in the tolerance class.*

We propose two new parametric tolerance classes. For each, we give an $O(n)$ time algorithm for testing if an n-sided part is in the class. For feeding *we give an $O(n^2)$ time algorithm to compute the maximum radius of a circular tolerance zone around each vertex. Numerical experiments provide evidence that this bound is tight. For* fixturing *we give an $O(1)$ time algorithm to compute the maximum dimensions of rectangular tolerance zones. We implemented both algorithms and report experimental results.*

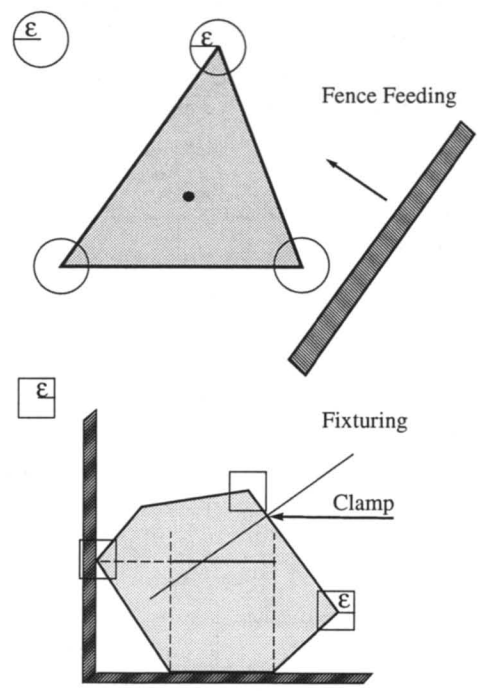


Figure 1: *Tolerance classes for feeding and fixturing.*

1 Introduction

Geometric tolerance is increasingly important as parts and products shrink, yielding relatively larger variations in part shape. Such variations are inevitable due to physical fabrication; all parts vary at some microscopic level. New processes such as MEMS (Micro Electro Mechanical Systems) and stereo lithography impose relatively large variations in part geometry. Characterizing allowable shape tolerance is important for product functionality; it is less widely recognized that variations in part shape can also cause manufacturing processes to fail. In this paper we consider two specific manufacturing processes, fence feeding and modular fixturing, and develop new algorithms for characterizing shape tolerance in these contexts.

We build on the geometric algorithms for feeding and fixturing that assume exact part geometry as input. For each process, we define a tolerance class $\Delta(P, \varepsilon)$ based on a nominal part P and a variation parameter

ε as shown in Figure 1. Both classes are defined so that we can check membership in linear time. Given a nominal part P, we compute a solution, a feeding strategy or fixture. We then compute the largest allowable ε such that the solution will work for all parts in $\Delta(P, \varepsilon)$.

2 Related Work

A part feeder is a device that accepts parts in random orientation and outputs them in a unique orientation. Currently, the design of parts feeders is a black art that is responsible for up to 30% of the cost and 50% of workcell failures [14, 5]. An algorithm for feeding polygonal parts was proposed in [9] and subsequently refined in a series of papers [17, 7, 4]. The model we consider here orients parts of a given shape as they are pushed by a sequence of fences. Thus a solution is a sequence of fence angles.

A fixture is a device that locates and holds parts during assembly, inspection, or machining. Modular fixturing systems typically include a lattice of holes with precise spacing and an assortment of precision locating and clamping modules that can be rigidly attached to the lattice. The model we consider here clamps parts with a right angle fixture and one translating clamp [15]. For fixturing, a solution is an orientation for the part and a position for the clamp.

For reviews of semantic issues in tolerancing, see [18, 19, 20]. Parametric tolerance models allow certain parameters to vary within a given range. On the other hand, geometric tolerance zones require only that the part fit within a defined region or zone [18]. Actual part geometry can be quite gnarly while remaining within the zone. It is extremely difficult to characterize part behavior under these conditions although bounds on behavior of kinematic pairs has been the subject of a study by Joskowicz, Sacks and Srinivasan [10].

Yap and Chang [22] consider geometric metrology: given a set of sample points on the boundary of a closed curve, decide if the curve lies within a given geometric tolerance zone. Even for a (one-dimensional) line segment, conservative classification policies may not exist; the authors illustrate that deterministic policies and statistical decision rules can be non-trivial even in one dimension. The parametric tolerance models defined below permit exact classification in linear time.

Latombe, Wilson and Cazals consider tolerancing in the context of assembly [12]. They propose a parametric tolerance model similar to ours in that they assume part edges are straight. To make the analysis manageable, they add the requirement that edges/faces maintain their relative orientation: i.e., edges in the tolerance class always remain parallel to the nominal edge. For a given set of tolerance specifications, the authors give polynomial time algorithms to decide if an assembly sequence exists for an assembly of polygonal or polyhedral parts. For our applications, we explicitly compute the radius of an allowable tolerance zone around each vertex.

Perhaps the work closest to ours is Akella and Mason [1, 3]. They consider fence and push-squeeze plans for orienting polygonal parts based on a similar framework. One difference is that their parametric tolerance class defines a circular tolerance zone around the center of mass. They note that determining a bound on the possible center of mass of a uniform mass polygon, in terms of bounds on its vertices, is an open problem. We finesse this problem by defining a tolerance class whereby vertices are defined relative to a coordinate frame at the center of mass.

Given the radius of a circular tolerance zone around each vertex and the center of mass, Akella and Mason compute action ranges and uses breadth-first search of an AND/OR tree to check if a solution exists. We extend their framework for feeding with a new model for shape tolerance; rather than testing if a given radius will succeed, we estimate the maximum radius of the tolerance zones.

Donald [8] considered parametric shape variation in the context of motion planning by adding dimensions to configuration space. Our work is a step toward bounding variation in these dimensions. Brost and Peters [6] propose a different model of shape tolerance for modular fixturing where contact normals are allowed to vary within a cone of given half angle. Kavraki considered variation in part shape in the context of ori-

enting parts with an elliptical planar force field. She was able to bound the final orientation of a part in terms of the shape difference between parts P and P': $P - P' \cup P' - P$ [11].

In this paper, we define parametric tolerance classes for feeding and fixturing. We give algorithms for finding bounds on the size of the tolerance zone and algorithms for checking if a part is in the class.

3 Shape Tolerance for Feeding

Polygonal parts can be fed (oriented) as they come into contact with fences on a conveyor belt [16, 2, 21, 4]. Let P be the nominal convex polygonal part with n vertices and center of mass c. We assign a fixed coordinate frame to P, with c as the origin of the coordinate frame and the x-axis aligned with the desired nominal final orientation of the part after feeding.

All directions are expressed relative to this coordinate frame as shown in Figure 2. Vertices are counterclockwise labeled $v_1 \ldots v_n$ and e_i is the edge connecting v_i and v_{i+1}; the edge e_n connects v_n and v_1. Let α_{v_i} be the direction of the ray cv_i in the coordinate system and α_{e_i} be the direction of the normal from c to the line containing the edge e_i. Let $\alpha_{r_i} = \alpha_{e_i} - \alpha_{v_i}$, $\alpha_{l_i} = \alpha_{v_{i+1}} - \alpha_{e_i}$; these angles play a crucial role in our analysis.

3.1 A tolerance class for feeding

For feeding, we define a parametric tolerance class $\Delta(P, \varepsilon)$ based on a nominal part P with center of mass c, as shown in Figure 3. We assume that part edges are always straight. Each vertex of P has a circular tolerance zone of radius ε. A part P' with center of mass c' is in $\Delta(P, \varepsilon)$ if we can position and orient P' such that c' is aligned with c and all of P''s vertices are inside the tolerance zones around the corresponding vertices of P. (Note that we assume a given assignment of vertices.) We assume that topologies of the convex hull of all the parts in $\Delta(P, \varepsilon)$ are the same. We can test whether or not a polygon P' is in $\Delta(P, \varepsilon)$ by computing an angular interval Ω_i' for each vertex v_i' such that rotating P' by $\omega \in \Omega_i'$ will move v_i' into the ε disk around v_i, or will

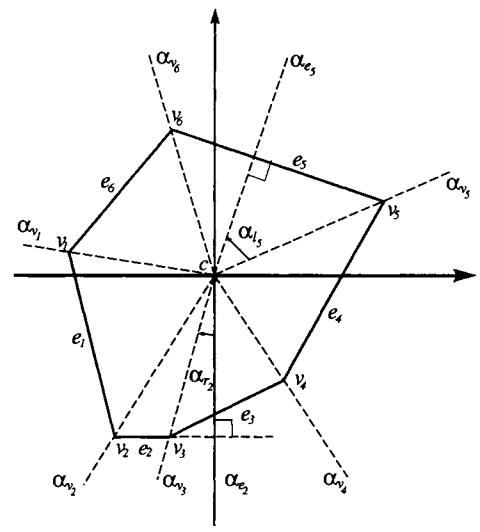


Figure 2: *Notation for feeding.*

keep it inside the disk if it is already inside the disk. If $\cup_i \Omega_i' \neq \emptyset$, then $P' \in \Delta(P, \varepsilon)$. The complexity of this test is $O(n)$.

3.2 The push function and push plans

As shown in 4, the *push function* characterizes the mechanics of a part pushed by a frictionless planar fence [13, 9]. We adopt the notation of [4]. In most cases, parts will start to rotate when pushed. We define the push direction as the direction of the normal from c to the pushing fence. If pushing in a certain direction does *not* cause the part to rotate, we refer to the corresponding push direction of as an *equilibrium* push direction. There are two types of equilibrium push direction, stable equilibrium push direction and unstable equilibrium push direction. If pushing does change the orientation, then this rotation changes the orientation of the pushing device relative to the part. We assume that pushing continues until the part stops rotating and settles in a stable equilibrium pose.

Edge e_i is called a *stable edge* if α_{e_i} corresponds to a stable equilibrium push direction, and e_i is called unstable otherwise. Similarly vertex v_j is called an *equilibrium vertex* if α_{v_j} corresponds to an equilibrium push direction. In the case of a convex polygon, all stable push directions correspond to stable edges and

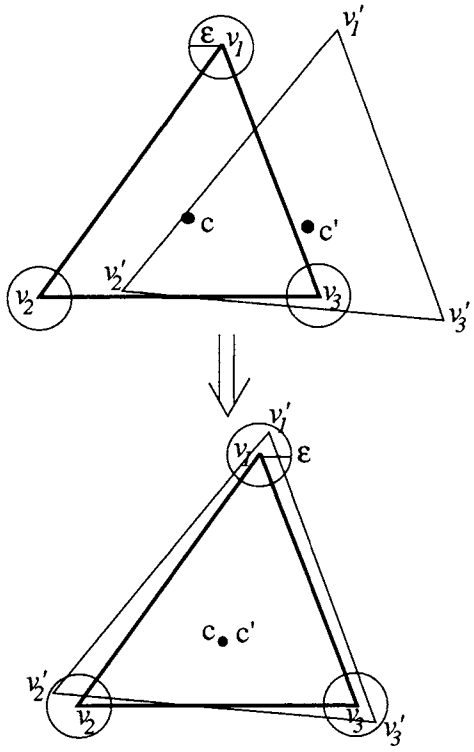


Figure 3: *Check if candidate part $P' \in \Delta(P, \varepsilon)$.*

all the equilibrium push directions corresponding to vertices are unstable. Edge e_i is stable if and only if $\alpha_{r_i} > 0$ and $\alpha_{l_i} > 0$.

The push function $p : [0, 2\pi) \to [0, 2\pi)$ links every orientation ϕ to the orientation $p(\phi)$ in which the part P settles after being pushed by a jaw with initial push direction ϕ. During the pushing, the rotation of the part causes the push direction of the jaw to change. The final orientation $p(\phi)$ of the part is the push direction of the jaw after the part has settled. The equilibrium push directions are the fixed points of p and, as we stated above, they all correspond to stable edges.

The push function p of a polygonal part consists of steps, which are intervals $I \subset [0, 2\pi)$ such that for all $\phi \in I$, $p(\phi) = C$ for some constant $C \in I$; see Figure 4. The steps of the push function are easily constructed from the α_{l_i}'s and α_{r_i}'s. The interval I is bounded by two consecutive unstable equilibrium push directions. All directions strictly inside the interval map onto the stable equilibrium push directions corresponding to a stable edge. (Note that the direction C itself maps onto C because it is the direction of the stable edge.) As a result, the number of steps in the push function equals the number of stable edges.

In preparation for the next section, we define for each stable edge e_i two open intervals $l(e_i) = \{\phi < \alpha_{e_i} | p(\phi) = \alpha_{e_i}\}$ and $r(e_i) = \{\phi > \alpha_{e_i} | p(\phi) = \alpha_{e_i}\}$. We refer to these intervals as e_i's left and right environment respectively. The interval $l(e_i)$ corresponds to the half-step left of α_{e_i} and $r(e_i)$ corresponds to the half-step right of α_{e_i} (Figure 4). The open intervals $l(e_i)$ and $r(e_i)$ are maximal intervals without equilibrium orientation.

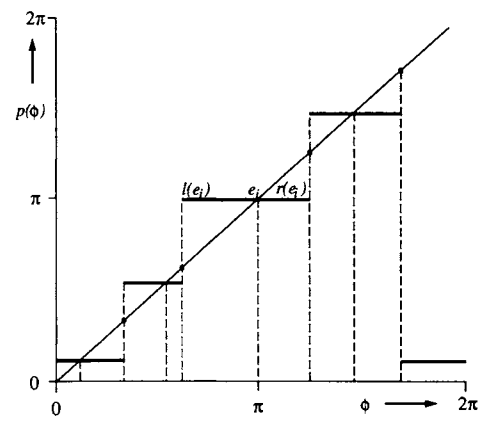


Figure 4: *The push function for a polygonal part. Each step corresponds to a stable edge and is bounded by two consecutive unstable equilibrium push directions which correspond to the directions of equilibrium vertices.*

We use the abbreviation p_α to denote the (shifted) push function defined by

$$p_\alpha(\phi) = p((\phi + \alpha) \bmod 2\pi)$$

for all $\phi \in [0, 2\pi)$. Note that $p_\alpha(\phi)$ is the final orientation of a part in initial orientation ϕ after a reorientation by α followed by a push. We can now define a *push plan.*

Definition 3.1 *A push plan is a sequence $\alpha_1, \ldots, \alpha_m$ such that $p_{\alpha_m} \circ \ldots \circ p_{\alpha_1}(\phi) = \Phi$ for all $\phi \in [0, 2\pi)$ and some fixed $\Phi \in [0, 2\pi)$.*

3.3 Push planning algorithms

Goldberg [9] showed that a push plan exists for any polygonal part and gave an $O(n^2)$ algorithm for finding the shortest such plan. Chen and Ierardi [7] gave a different algorithm based on finding a basic pushing action that repeatedly applies one critical pushing angle to orient the part. The critical pushing angle depends on the length of the two largest open left half intervals in the push function. The same is true for right intervals but, without loss of generality, we only consider left push plans here. Let Ψ be a range of angles defined by the difference in length of these half intervals. Any angle in Ψ will serve as the critical pushing angle. This range can compensate for variation in part shape.

Consider the open intervals $l(e_i)$ defined in the previous section. Let α be the length of the longest interval in the set. We assume, without loss of generality, that e_1 in P is a stable edge with $|l(e_1)| = \alpha$, and we only consider the case in which P has no second stable edge e_i $(i \neq 1)$ satisfying $|l(e_i)| = \alpha$. Each basic action consists of a reorientation of the jaw by an angle of $-(\alpha - \mu)$, with $\mu > 0$ such that $\alpha - \mu > |l(e_i)|$ for any stable edge $e_i \neq e_1$, and a subsequent application of the jaw. (The same arguments will apply for $r(e_i)$ with reorientation by an angle of $\alpha - \mu$ instead of $-(\alpha - \mu)$.)

Note that a reorientation of the jaw by $-(\alpha - \mu)$ corresponds to a change of the push direction by $\alpha - \mu$. Every basic action puts the part into a stable equilibrium orientation. If P is a polygonal part, then the stable equilibrium orientations occur at isolated points in $[0, 2\pi)$. After each basic action, the number of possible part orientation is finite. Let us label the m stable edges $e_{s_1}, \ldots, e_{s_m}$ in order of *increasing* edge index with $s_1 = 1$. After the first application of the jaw, the part P can be in the orientation of any stable edge $e_{s_1}, \ldots, e_{s_m}$. Chen and Ierardi show that every next basic action eliminates the last stable edge in the sequence. So an $m + 1$ step push plan $p_{\alpha-\mu} \circ \ldots \circ p_{\alpha-\mu}$ suffices to put P into orientation α_{e_1}.

3.4 Effect of shape tolerance on the push function

When part geometry changes, it affects the push function in two ways. First, it will change the value of α_{l_i} and α_{r_i}, that is, it will change the length of one or more open half intervals. Second, some critical change on the value of α_{l_i} and α_{r_i} may cause a stable edge in the nominal part to become unstable. Also, it may cause an unstable edge to become stable. Since the number of steps of the push function is equal to the number of stable edges, this will combine several half intervals into a bigger one or break one half interval into several smaller ones. We refer to this as a *topological change* of the push function. It may dramatically change the ranking of the $|l(e_i)|$'s. An example of a polygon with tight topological constraints is shown in Figure 5. e_1 is the stable edge with the largest α_{l_i}. However, if the coordinates of v_1 undergo a small change, it may cause e_1 to become unstable. A possible topological change of the push function for this polygon is shown in Figure 6.

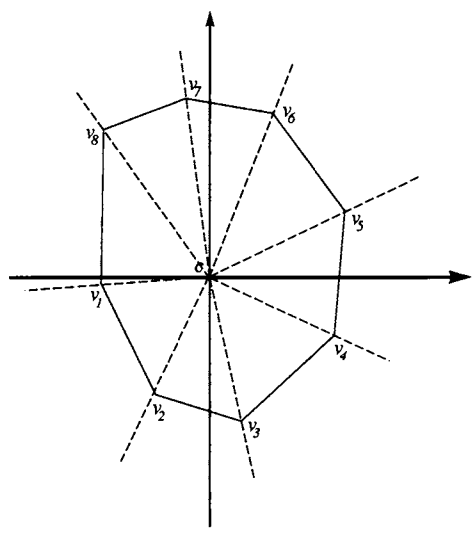


Figure 5: *Example of a polygon with tight topological constraint.*

Using Chen and Ierardi's algorithm, we can find a push plan that is valid for all the parts in the tolerance class $\Delta(P, \varepsilon)$ if

- for every part in $\Delta(P, \varepsilon)$, e_1 is the stable edge with the largest left half interval, and

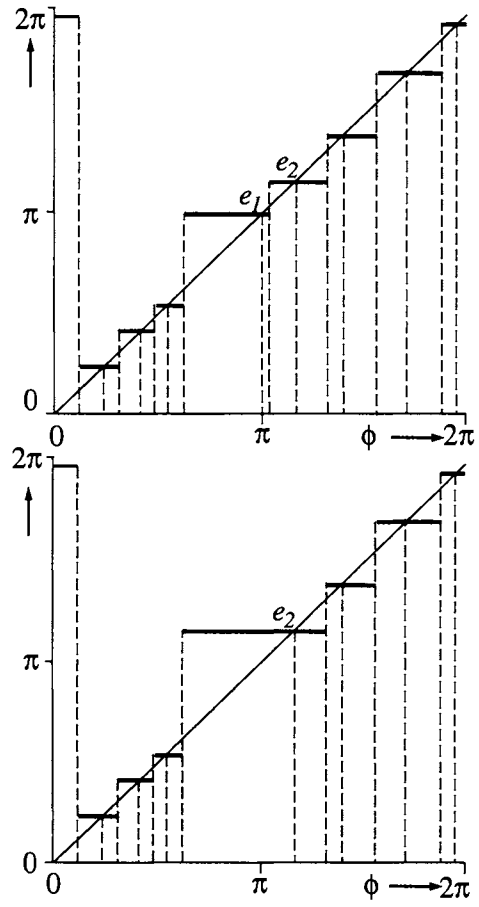


Figure 6: *Topological change of the push function for the part in Figure 5when v_1 is moved.*

- if each member $P' \in \Delta(P, \varepsilon)$ is characterized by a pushing angle set Ψ', and the intersection of all Ψ' is nonempty.

We can state this differently. If the lower bound of $|l(e_1')|$ for any part $P' \in \Delta(P, \varepsilon)$ is larger than the upper bound of all other left half intervals, then there will be a push plan which will work for all the parts in the tolerance class.

3.5 Algorithm to compute ε

We now compute the radius of a tolerance class for which a common push angle exists. The algorithm takes as input the geometry of the nominal part and computes the largest ε such that there exists a plan to feed all the parts in $\Delta(P, \varepsilon)$, and a push angle α that defines the plan.

Initially, we assume that there is no topological change on the push function within the tolerance class $\Delta(P, \varepsilon)$, and study the relationship between $|l(e_i')|$ and ε where e_i is a stable edge. Afterward, we address the issue of topological changes in the push function.

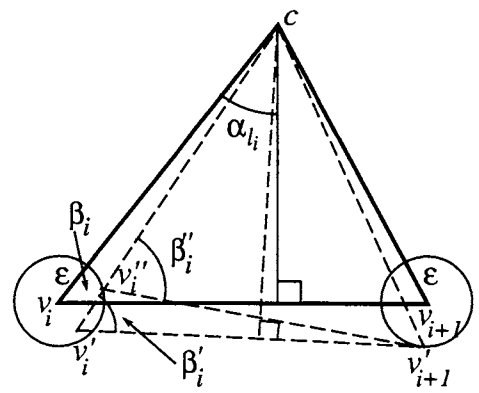


Figure 7: *Case 1. v_i is an equilibrium vertex thus $l(e_i) = \alpha_{l_i}$.*

There are two cases we need to consider, according to whether v_i is an equilibrium vertex or not.

Case 1. As shown in Figure 7, v_i is an equilibrium vertex, thus $l(e_i) = \alpha_{l_i}$. Let $\beta_i = \pi/2 - \alpha_{l_i}$. First we derive a lower bound on $l(e_i)$. We start by making the following observations:

1. Minimizing α_{l_i} is equivalent to maximizing β_i. As shown in Figure 7 and by definition $\alpha_{l_i} + \beta_i = \pi/2$.

2. For every v_i' within the ε disk around v_i, the maximum β_i is achieved when e_i' is tangent to the ε disk around v_{i+1}. So given the vertex v_i', the maximum value of α among all $P' \in \Delta(P, \varepsilon)$ and $v_i' \in P'$ is well defined. The problem now becomes to find the point v_i' that can maximize β_i.

3. For any v_i' *inside* the ε disk around v_i, there is a point v_i'' *on* the ε disk around v_i such that $\beta_i'' > \beta_i$. Thus we need only consider points on the boundary of the circular tolerance zone.

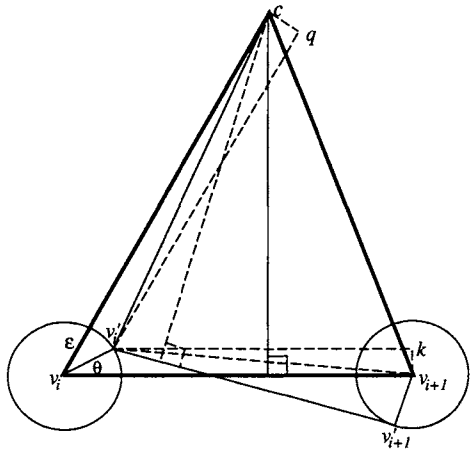


Figure 8: *Relationship between* $\min(|l(e_i')|)$ *and* ε *in case 1.* $\angle cv_i'q = \gamma$, $\angle kv_i'v_{i+1} = \lambda$, $\angle v_{i+1}v_i'v_{i+1}' = \delta$.

A point v_i' on the circle around v_i can be represented by a single variable θ. $v_i'v_{i+1}'$ is tangent to the ε disk around v_{i+1} since this maximizes β_i' when we fix v_i'. To derive the relationship between β_i' and ε, we draw a straight line $v_i'q$ parallel to v_ic with cq perpendicular to $v_i'q$ and $v_i'k$ parallel to v_iv_{i+1} with $v_{i+1}k$ perpendicular to $v_i'k$. Let $\angle cv_i'q = \gamma$, $\angle kv_i'v_{i+1} = \lambda$, $\angle v_{i+1}v_i'v_{i+1}' = \delta$. We have:

$$\beta_i' = \gamma + \beta_i + \delta + \lambda \tag{1}$$

Lemma 1 *When we only consider points on the* ε *ball, the relationship between* $\min(|l(e_i')|)$ *and* ε *can be expressed as*

$$\begin{aligned}\min(|l(e_i')|) &= |l(e_i)| \\ &- \arcsin \frac{\varepsilon \sin(\beta_i - \theta)}{\sqrt{(|cv_i| - \varepsilon\cos(\beta_i - \theta))^2 + \varepsilon^2 \sin^2(\beta_i - \theta)}} \\ &- \arcsin \frac{\varepsilon(1 + \sin\theta)}{\sqrt{(|e_i| - \varepsilon\cos\theta)^2 + \varepsilon^2 \sin^2\theta}},\end{aligned} \tag{2}$$

where θ *is the value in* $[0, 2\pi)$ *that minimizes the right-hand side.*

Proof. From Figure 8:

$$\begin{aligned}\gamma &= \arcsin \frac{|cq|}{|cv_i'|} \\ &= \arcsin \frac{\varepsilon \sin(\beta_i - \theta)}{\sqrt{(|cv_i| - \varepsilon\cos(\beta_i - \theta))^2 + \varepsilon^2 \sin^2(\beta_i - \theta)}}\end{aligned}$$

$$\delta = \arcsin \frac{\varepsilon}{|v_i'v_{i+1}'|} = \arcsin \frac{\varepsilon}{\sqrt{(|e_i| - \varepsilon\cos\theta)^2 + \varepsilon^2 \sin^2\theta}}$$

$$\lambda = \arcsin \frac{\varepsilon \sin\theta}{|v_i'v_{i+1}'|} = \arcsin \frac{\varepsilon \sin\theta}{\sqrt{(|e_i| - \varepsilon\cos\theta)^2 + \varepsilon^2 \sin^2\theta}}$$

For any given ε, we can find the value of θ that maximize $\gamma + \delta + \lambda$. Using Equation 1, the relationship between $\min(|l(e_i')|)$ and ε as stated in the lemma follows. □

We can then show

Lemma 2 *When we only consider points on the* ε *ball, the relationship between* $\max(|l(e_i')|)$ *and* ε *can be expressed as*

$$\begin{aligned}\max(|l(e_i')|) &= |l(e_i)| \\ &+ \arcsin \frac{\varepsilon \sin(\beta_i - \theta)}{\sqrt{(|cv_i| - \varepsilon\cos(\beta_i - \theta))^2 + \varepsilon^2 \sin^2(\beta_i - \theta)}} \\ &+ \arcsin \frac{\varepsilon(1 + \sin\theta)}{\sqrt{(|e_i| - \varepsilon\cos\theta)^2 + \varepsilon^2 \sin^2\theta}},\end{aligned} \tag{3}$$

where θ *is the value in* $[0, 2\pi)$ *that maximizes the right-hand side.*

Case 2. As shown in Figure 9, v_i is *not* an equilibrium vertex. Thus $l(e_i)$ is defined by α_{e_i} and α_{v_j}, where v_j is the last equilibrium vertex ($j \neq i$) (namely, when we proceed clockwise from v_i along the boundary of P v_j is the first equilibrium vertex that we encounter).

In this case, we argue the following

Lemma 3 *When we only consider points on the* ε *ball, the relationship between* $|l(e_i')|$ *and* ε *can be expressed as*

$$\begin{cases} \min(|l(e_i')|) = |l(e_i)| - \arctan\frac{\varepsilon}{|cv_j|} - \arctan\frac{2\varepsilon}{|e_i|} \\ \max(|l(e_i')|) = |l(e_i)| + \arctan\frac{\varepsilon}{|cv_j|} + \arctan\frac{2\varepsilon}{|e_i|} \end{cases} \tag{4}$$

Proof. We observe that $|l(e_i')| = |l(e_i)| + \alpha_{v_j} - \alpha_{v_j'} + \alpha_{e_i'} - \alpha_{e_i}$.

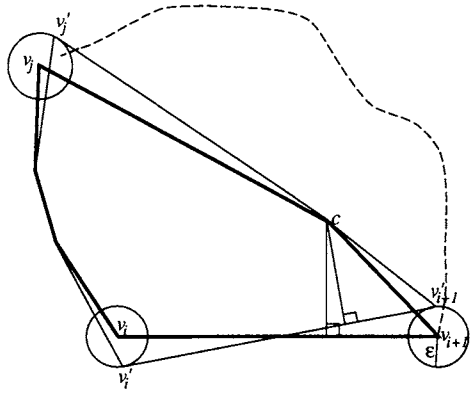


Figure 9: *Case 2. v_2 is not an equilibrium vertex thus $l(e_i) = \alpha_{e_i} - \alpha_{v_j}$ where v_j is the last equilibrium vertex ($j \neq i$).*

Since $-\arctan\frac{\varepsilon}{|cv_j|} \leq \alpha_{v_j} - \alpha_{v'_j} \leq \arctan\frac{\varepsilon}{|cv_j|}$ and $-\arctan\frac{2\varepsilon}{|e_i|} \leq \alpha_{e'_i} - \alpha_{e_i} \leq \arctan\frac{2\varepsilon}{|e_i|}$, the relationship between $|l(e'_i)|$ and ε as stated in the lemma follows. □

Combining both cases, we find that for every stable edge e_i, if there is no topological change on the push function, we can express the relationship between $|l(e'_i)|$ and ε as Equation 2, 3 or 4. They form a set of nonlinear constraints. From these constraints, we can compute the largest ε zone numerically in which $\min(|l(e_1)| \geq \max(|l(e_i)|$ for all $i \neq 1$ and e_i is a stable edge.

Now we consider topological changes on the push function. As we discussed before, it is caused by critical change on the value of α_{l_i} and α_{r_i}. Since in Case 1 $|l(e_i)| = \alpha_{l_i}$, we already know the relationship between α_i and ε. Each possible critical change on the value of α_{l_i} and α_{r_i} can be expressed as a linear constraint on ε. Given ε, we can determine which topological changes of the push function can occur. Thus we can first ignore the topological constraints, and compute ε. Then we check whether ε is valid for all the topologically different situations that are possible within $\Delta(P, \varepsilon)$. If not, we decrease ε to exclude the first topological class which violates the tolerance condition. We repeat until no more violations occur.

Before considering topological constraints, the algorithm runs in time $O(n)$. Since there are $O(n)$ topological constraints and each takes $O(n)$ time to handle, the overall complexity for computing ε is $O(n^2)$.

3.6 Convexity

Intuition suggests that if the condition above holds for all extrema in the tolerance zone, then the condition will hold for any part in the tolerance class. Unfortunately, this is not the case for feeding. We construct a counterexample, illustrated in Figure 10, where the size of the left half interval shrinks, then grows as we move a vertex continuously from v_{i+1} to v''_{i+1}.

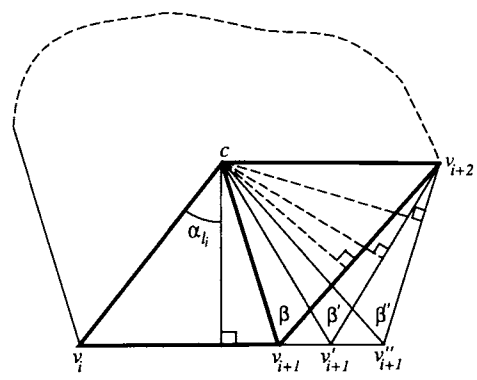


Figure 10: *Counterexample where convexity does not hold.*

Let c be the COM of a nominal part with adjacent edges e_i and e_{i+1} defined by v_i, v_{i+1}, v_{i+2} respectively. We construct cv_{i+2} such that it is parallel to v_iv_{i+1} and $|v_{i+1}v_{i+2}| > |cv_{i+1}|$. Next, extend v_iv_{i+1} to v''_{i+1} such that $|cv''_{i+1}| = |v_{i+1}v_{i+2}|$ and $|cv_{i+1}| = |v''_{i+1}v_{i+2}|$. Let v'_{i+1} be the midpoint of $v_{i+1}v''_{i+1}$. Let $l(e_{i+1})$, $l(e'_{i+1})$ and $l(e''_{i+1})$ be the left environment of the corresponding edge. Since $|l(e_{i+1})|+\beta = \pi/2$, $|l(e'_{i+1})|+\beta' = \pi/2$, $|l(e''_{i+1})| + \beta'' = \pi/2$ and $\beta' > \beta = \beta''$, then $|l(e'_{i+1})| < |l(e_{i+1})| = |l(e''_{i+1})|$. Thus, when we move vertex v_{i+1} to v'_{i+1}, $|l(e_{i+1})|$ initially shrinks, then grows back to its original size.

For a polygon where $|l(e_{i+1})|$ is the critical half interval, the same push plan will work for v_{i+1} and v''_{i+1} but may not work for v'_{i+1}. This demonstrates that, in the context of feeding, we cannot naively assume convexity and test only extrema in the tolerance zone.

3.7 Implementation and experimental results

We implemented the algorithm and ran it on 9 nominal parts, six of which are shown in Figure 11.

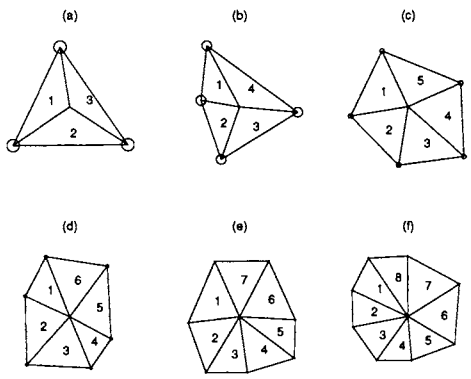


Figure 11: *Nominal polygons and their tolerance zones. The numbers indicate part edges.*

To test that the resulting tolerance zones were correct, we generated 1000 random variations for each part by allowing vertices to vary uniformly within each tolerance zone. As expected, the nominal plan worked for all parts in the tolerance class. Zero failures are indicated in the second column in Table 1.

We then increased the tolerance zone by a factor of 1.1, 1.2, 1.3, 1.4 respectively, and in each case generated 1000 random variations as above. Here, as expected, the nominal plan sometimes fails since some polygons lie outside the tolerance zone. Although the percentage is small, this numerical experiment suggests that the tolerance zone we compute provides a tight bound.

Small changes in part geometry can cause large changes in the tolerance class. For example, the two 8-sided parts in Figure 12 are identical except for the y coordinate of vertex v_1, which in (a) is -0.1 and in (b) is -2.3. Note that the radius of the tolerance class in (b) is significantly larger than in (a)! This is because the part in (a) has a tight topological constraint. This suggests that if we change the design of part from (a) to (b), it will be much more robust to variations in part shape.

(# of edges)	1.0ε	1.1ε	1.2ε	1.3ε	1.4ε
3	0	0.7	1.2	3.9	5.8
4	0	0.2	0.9	2.1	3.0
5	0	0.7	1.8	4.3	5.6
6	0	0.3	0.9	3.6	5.2
7	0	0.0	0.5	2.1	3.0
8	0	0.2	1.1	2.5	6.0
9	0	0.3	0.9	2.1	5.0
10	0	0.4	1.4	2.4	5.3
17	0	0.1	0.5	1.7	3.0

Table 1: *Results for 9 nominal parts, 5000 trials each. For each part and tolerance radius, the table lists the percentage of polygons that cannot be fed with the nominal plan. The first column corresponds to polygons in the computed tolerance class (no failures). The other columns are polygons outside the tolerance class.*

4 Shape Tolerance in Fixturing

Wentink, van der Stappen and Overmars [15] show that any convex polygon without parallel edges can be immobilized by a right angle fixel and a single point contact provided with a clamp. Let P be a convex polygon. When placed in the fixture, it has a horizontal edge contact and a vertical point contact on the right angle fixel. We assume that the contacts are frictionless. This is a conservative assumption since any fixture computed assuming zero friction also holds in presence of friction. We label the two endpoints of the edge contact as v_1 and v_2, the point contact as v_p, and the vertices of the edge to which we apply the clamp as v_{c1} and v_{c2}. Edge $v_{c1}v_{c2}$ is called the *clamp edge*; see Figure 13 for an illustration. If we draw vertical lines from v_1 and v_2, they intersect the horizontal line from v_p at points f_1 and f_2 respectively. We refer to the line segment f_1f_2 as the *fixturing segment* since the condition for the right angle fixel and clamp to immobilize the part is that the normal of the clamp edge at the clamp contact point intersects the fixturing segment. For details see [15].

4.1 A tolerance class for fixturing

As before, let P be a convex polygonal part. Its n vertices are counterclockwise labeled $v_1 \ldots v_n$. Edge

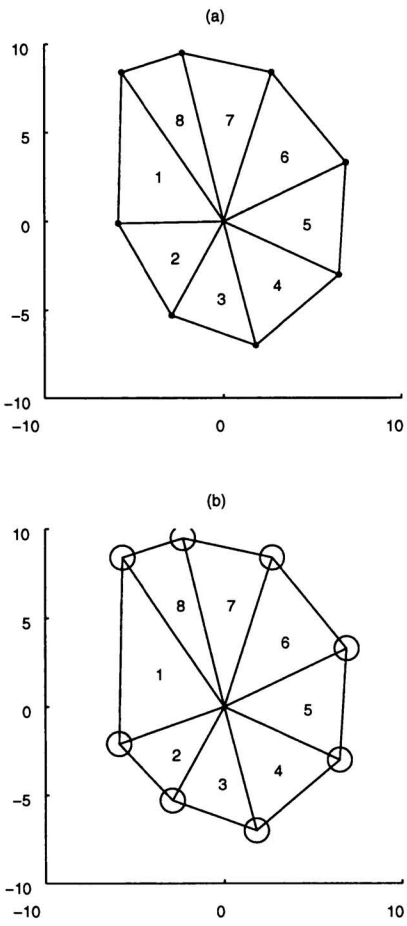


Figure 12: *Computed tolerance zones for two similar polygons.*

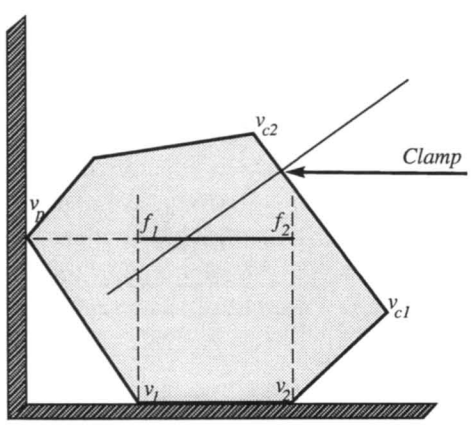


Figure 13: *Fixturing with a right angle fixel and a point clamp.*

v_1v_2 is the edge that we place on the horizontal arm of the right-angle fixel. We assume that a fixed coordinate frame is attached to P. We choose the midpoint of the edge v_1v_2 as the origin which we denote by c, and align the x-axis with the edge v_1v_2. All the coordinates are expressed relative to this frame (Figure 14).

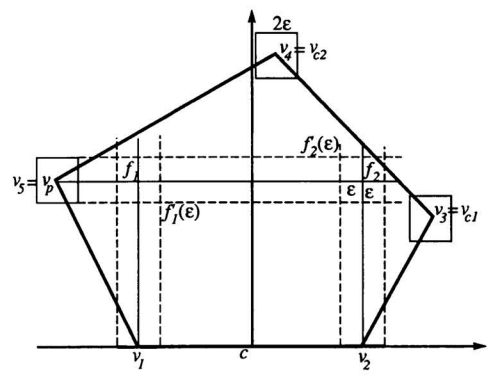


Figure 14: *Description of the part and its shape uncertainty in fixturing.*

Now we define the parametric tolerance class $\Delta(P, \varepsilon)$. Again, we assume that part edges are always straight. For fixturing, we replace the circles around vertices with boxes and line segments. This is more natural for fixturing because horizontal and vertical shifts play a more prominent role than rotation. Let P be the nominal part, and let P' be a candidate. Let c' denote the center of the edge $v_1'v_2'$ of P'. We say that $P' \in \Delta(P, \varepsilon)$ if we can place part P' such that c' is coincident with c, the bottom edges are aligned, and all the vertices of P' are within a square of size ε around the corresponding vertices of the nominal part P. It is easy to check if a candidate $P' \in \Delta(P, \varepsilon)$ in $O(n)$ time. An example is shown in Figure 14.

4.2 Effect of shape variations

A fixture design with a right angle fixel is described by the edge of the part that must touch the bottom arm of the fixel, and the height of the clamp. A part P' is loaded into the fixture design by placing it with edge v_1v_2 on the horizontal arm, shifting it to the left until it hits the vertical arm, and shifting the clamp horizontally until it touches the part. A part P' is said

to be fixtured by a particular fixture design, if, after loading it, it is indeed immobilized. A fixture design is considered valid for the tolerance part set $\Delta(P, \varepsilon)$ if it fixtures each part in the set and the clamp will touch the part at the same edge for each part.

When the part shape changes in the tolerance class, the fixturing segment may change in height and length, the clamp contact point may move horizontally and its normal may rotate. Also, the vertex v_p that makes the contact at the vertical arm of the fixel may be changed. In this paper, we restrict our tolerance zone such that this last type of change does not happen and we only consider the case where v_1, v_2, v_p, v_{c1}, and v_{c2} are distinct vertices.

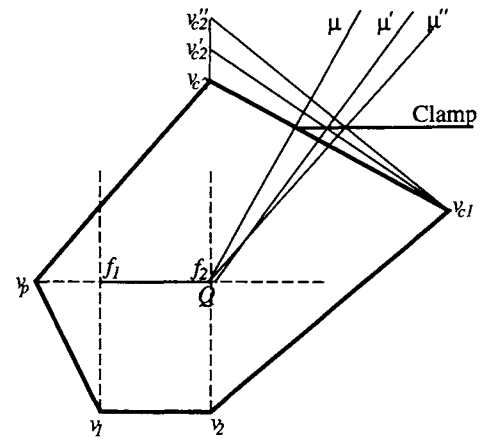


Figure 15: *A counterexample: convexity is violated.*

4.3 Convexity

For fixturing as well we show that convexity does not necessarily help in analyzing the tolerance zone. Namely, we show that it is insufficient to analyze a fixture at the extrema of a tolerance zone, as its behavior can change between extrema.

Assume that we have two polygons P and P'' that are the same except for the location of one vertex which is at position A in P and at position A'' in P''. Now assume both P and P'' can be held with the same fixture. Convexity would suggest that as we move the vertex from A to A'' along the line segment AA'', the resulting polygons will also be held with the same fixture. If convexity holds, then we only need to consider combinations of all the vertices of the tolerance boxes to check if for a certain ε a valid fixture design exists. Unfortunately, this is not always true, as the following counterexample shows (see Figure 15).

As illustrated in the figure, we move the vertex v_{c2} to v''_{c2}. We denote by μ and μ'' the normals at the clamp contact point of edge $v_{c1}v_{c2}$ and $v_{c1}v''_{c2}$ respectively. The two normals intersect at point Q. It is easy to construct a polygon such that Q is on the fixturing segment. Since μ and μ'' both intersect the fixturing segment at Q, the monotonicity should imply that for every point v'_{c2} on the line segment $v_{c2}v''_{c2}$, the normal μ' must also intersect the fixturing segment at Q. (Otherwise it is easy to construct a polygon such that the fixturing segment is so small that it does contain Q but not the other intersection point.) But this is obviously false. So, we have shown that the x-coordinate of the intersection of the normal at the clamp contact point and the fixture line segment does not always change monotonically. Thus a valid fixture for polygons with vertex at v_{c2} and v''_{c2} may not be valid for every polygon with its vertex on $v_{c2}v''_{c2}$.

4.4 An algorithm to compute max ε

Since we assume that v_1,v_2,v_p,v_{c1} and v_{c2} are distinct vertices, a fixture is valid for $\Delta(P, \varepsilon)$ if and only if for every $P' \in \Delta(P, \varepsilon)$, the normal at the clamp lies between $f_1(\varepsilon)$ and $f_2(\varepsilon)$ (see Figure 14). Now consider the edge at which we apply the clamp. The normal at the clamp point will intersect the fixture segment at some position x. We next show that the toleranced edge that achieves the maximum (or minimum) x must be such that at least one endpoint is coincident with one of four extreme points that define the two tolerance boxes at its endpoints. Consider Figure 16. This demonstrates graphically that for any edge l_2 in the tolerance zone which does not have an endpoint at one of the four extreme points ($ABCD$). We can always move the edge parallel to itself until one of the endpoint reaches the boundary, and get a larger (smaller) x. So we have shown that the maximum or minimum of x will be achieved only when at least one endpoint of the edge is coincident with one of the four extreme points

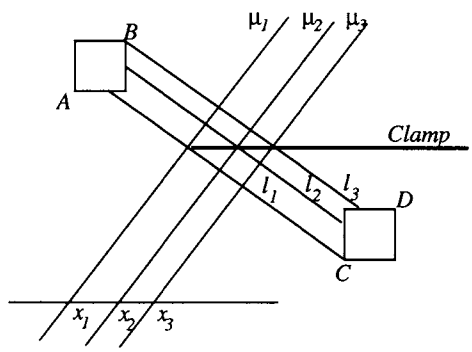


Figure 16: *Boundary condition.*

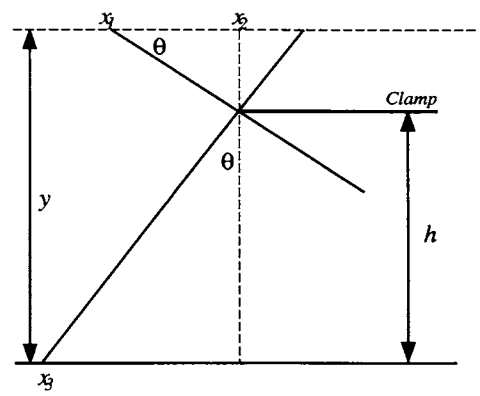


Figure 17: *Relationship between θ and x_3 when we fix one endpoint of the clamp edge.*

$(ABCD)$. Note that, as we have shown before, it is not necessary that both endpoints are extreme points.

Now let us fix one of the endpoints of the edge and look at the relationship between the orientation of the clamp edge and the x-coordinate of intersection of the normal and the fixture segment. As indicated in Figure 17, let y be the vertical distance from the fixed endpoint to the fixture segment, and let h be the vertical distance from the clamp to the fixture segment. Let x_1 be the x-coordinate of the fixed endpoint, x_2 the x-coordinate of the clamp contact point, and x_3 the x-coordinate of intersection between the normal and the fixture segment. Finally, let θ be the orientation of the clamp edge.

Because the normal is perpendicular to the clamp edge, we have

$$\frac{y-h}{x_2-x_2} = \frac{x_2-x_3}{h} = \tan\theta \tag{5}$$

From Equation 5 , we eliminate x_2 and get

$$x_3 = x_1 + (y-h)\cot\theta - h\tan\theta \tag{6}$$

Take the derivative of x_3 on θ and set it to be 0,

$$-\frac{y-h}{\sin^2\theta} - \frac{h}{\cos^2\theta} = 0 \tag{7}$$

Solving Equation 7, we get

$$\theta = \arctan\sqrt{1-\frac{y}{h}} \tag{8}$$

Equation 8 has at most one solution in $[0, \pi/2]$. So, given ε, for each fixed endpoint we only need to consider at most three directions: the maximal and minimal θ within the tolerance zone and the solution of Equation 8. Also we only need to consider four fixed endpoints (the corners of the tolerance box) at each side of the clamp edge. Looking at this carefully, we get a total of 8 clamp edges and 2 points ($f_1(\varepsilon)$ and $f_2(\varepsilon)$) within the tolerance zones that might define the maximum or minimum position of the intersection of the normal and the fixture edge. (The 8 edges are the four tangent lines of the two tolerance boxes plus, for each of the 4 extreme vertices (2 per box), the angle that comes out of Equation 8.) This leads to a system with a constant number of equations from which we can compute the largest ε in which the fixture design will work for all parts in $\Delta(P, \varepsilon)$ (assuming no topological change). Since only five vertices are relevant in the calculation of ε, the complexity of this algorithm is $O(1)$.

4.5 Implementation and experimental results

We implemented the above algorithm and ran similar test as we did in feeding on 4 nominal parts(as shown in Figure 18, the results are listed in Table 2. The numerical experiment suggests that the tolerance zone we compute provides a tight bound.

The same part may have very different tolerance zones for different fixture plans. For example, the two

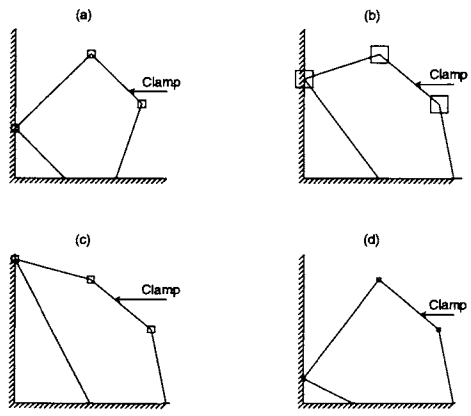


Figure 18: *Nominal polygons and their tolerance zones.*

Polygon	1.0ε	1.1ε	1.2ε	1.3ε	1.4ε
a	0	0.1	0.1	0.6	1.3
b	0	0.1	0.3	1.3	2.9
c	0	0.2	0.2	0.8	1.9
d	0	0.0	0.1	0.4	1.4

Table 2: *Results for 4 nominal parts, 5000 trials each. For each part and tolerance radius, the table lists the percentage of polygons that cannot be immobilized by the nominal plan. The first column corresponds to polygons in the computed tolerance class (no failures). The other columns are polygons outside the tolerance class.*

parts in Figure 19 are identical, but the clamp height is different. Note that the radius of the tolerance class in (b) is significantly larger than in (a)! This suggests that if we change the fixture plan from (a) to (b), it will be much more robust to variations in part shape.

5 Conclusion and Future Work

In general, it is very difficult to characterize the range of mechanical behavior for the uncountable set of parts in a tolerance class. This is especially true for geometric tolerancing, where there is no limit on the part edges within the zone. For example, it is easy to see that a part that meets the geometric zone specification can wind up in a broad range of final orientations when pushed with a fence. It is not at all clear how to bound this range.

Recently there has been renewed interest in parametric tolerance classes and associated algorithms. Fast checking algorithms are very useful during the design cycle. Ideally, part geometry can be monitored in real time as is currently done with VLSI CAD design rule checking.

This paper contributes by rigorously characterizing two new parametric tolerance classes. For each, we give an $O(n)$ time algorithm for testing if an n-sided part is in the class. For feeding we give an $O(n^2)$ algorithm to compute the radius of the largest allowable tolerance zone around each vertex. For fixturing we give an $O(1)$ algorithm to compute the dimensions of rectangular tolerance zones. We implemented both algorithms and illustrate with experimental results.

In both cases the tolerance class is defined with respect to the context. For feeding, since the center of mass plays an important role in fence mechanics, it was very helpful to define variations in shape with respect to a coordinate frame attached to the center of mass. Similarly for fixturing, where variations in shape are relative to the part edge that is in contact with the edge fixel. This suggests an alternative to defining shape tolerance in the abstract.

Efficient algorithms are necessary to provide rapid feedback to designers. As the examples in Figures 12 and 19 indicate, small changes in part or fixture geometry can imply large changes in tolerance sensitivity. In a Computer Aided Design system, low complexity algorithms such as those described in this paper can run in the background to rapidly indicate tolerance effects. Future research is required to anticipate and suggest changes in part geometry that can improve tolerance sensitivity.

Last, the issue of convexity arose repeatedly in this study. We found that intuition often failed us: we guessed that if a solution exists at all extrema, it would hold everywhere in between. To our surprise, we discovered counterexamples. Our results and methodology for constructing counterexamples may be applicable in other contexts such as assembly and motion planning.

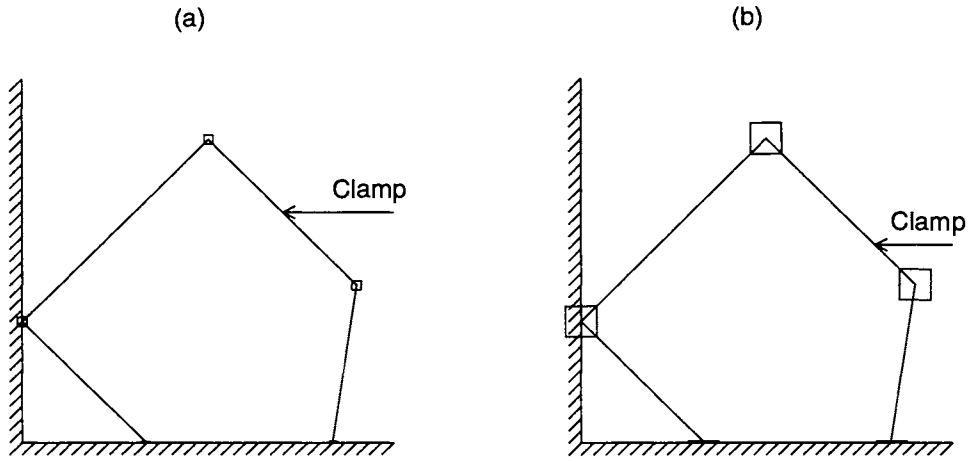


Figure 19: *Shape tolerance computation for fixturing. Intuition suggests that the clamp position in (a) would be more robust to shape variation than the clamp position (slightly lower) in (b). However, the algorithm computes a smaller allowable epsilon box in (a) than in (b). This illustrates that small changes in fixture design can produce counterintuitive effects on shape tolerance.*

Acknowledgements

This work was supported in part by NSF Awards IRI-9612491 and Presidential Faculty Fellow Award IRI-9553197 to Prof. Goldberg and by NATO Travel Grant CRG 951224 (joint with M. Overmars).

Work on this paper by D. Halperin has been supported in part by an Alon Fellowship, by ESPRIT IV LTR Project No. 21957 (CGAL), by the USA-Israel Binational Science Foundation, by The Israel Science Foundation founded by the Israel Academy of Sciences and Humanities, and by the Hermann Minkowski – Minerva Center for Geometry at Tel Aviv University.

References

[1] S. Akella. *Robotic Manipulation for Parts Transfer and Orienting: Mechanics, Planning, and Shape Uncertainty.* PhD thesis, The Robotics Institute, Carnegie Mellon University, Dec. 1996. Robotics Institute Technical Report CMU-RI-TR-96-38.

[2] S. Akella, W. Huang, K. Lynch, and M. Mason. Planar manipulation on a conveyor by a one joint robot with and without sensing. In *Second Workshop on Algorithmic Foundations of Robotics*, Toulouse, France, July 1996.

[3] S. Akella and M. T. Mason. Parts orienting with shape uncertainty. In *Submitted to ICRA 98*, Oct. 1997.

[4] R. P. Berrety, M. Overmars, F. V. der Stappen, and K. Goldberg. On fence design and the complexity of push plans for orienting parts. In *13th Symposium on Computational Geometry*, Nice, France, June 1997. ACM.

[5] G. Boothroyd, C. Poli, and L. E. Murch. *Automatic Assembly.* Marcel Dekker, Inc., 1982.

[6] R. C. Brost and R. R. Peters. Automatic design of 3d fixtures and assembly pallets. In *IEEE International Conference on Robotics and Automation*, 1996.

[7] Y.-B. Chen and D. Ierardi. The complexity of oblivious plans for orienting and distinguishing polygonal parts. *Algorithmica*, 14, 1995.

[8] B. R. Donald. *Error Detection and Recovery in Robotics.* Springer-Verlag, 1987.

[9] K. Goldberg. Orienting polygonal parts without sensors. *Algorithmica*, 10(2):201–225, August 1993. Special Issue on Computational Robotics.

[10] L. Joskowicz, E. Sacks, and V. Srinivasan. Kinematic tolerance analysis. In *Second Workshop on Algorithmic Foundations of Robotics*, Toulouse, France, July 1996.

[11] L. Kavraki. Part orientation with programmable vector fields: Two stable equilibria for most parts. In *IEEE International Conference on Robotics and Automation*, Albuquerque, New Mexico, Apr. 1997.

[12] J. C. Latombe, R. H. Wilson, and F. Cazals. Assembly sequencing with toleranced parts. In *Third Symposium on Solid Modelling and Applications*, 1995.

[13] M. T. Mason. Mechanics and planning of manipulator pushing operations. *International Journal of Robotics Research*, 5(3), Fall 1986.

[14] J. L. Nevins and D. E. Whitney. Computer-controlled assembly. *Scientific American*, 1978.

[15] M. Overmars, C. Wentink, and F. van der Stappen. Fixture planning. In *Second Workshop on Algorithmic Foundations of Robotics*, Toulouse, France, July 1996.

[16] M. A. Peshkin and A. C. Sanderson. Planning robotic manipulation strategies for workpieces that slide. *IEEE Journal of Robotics and Automation*, 4(5), October 1988.

[17] A. Rao and K. Goldberg. Manipulating algebraic parts in the plane. *IEEE Transactions on Robotics and Automation*, 11(4), August 1995.

[18] A. A. G. Requicha. Toward a theory of geometric tolerancing. *IJRR*, 2(4), 1983.

[19] U. Roy, C. Liu, and T. Woo. Review of dimensioning and tolerancing: Representation and processing. *Computer-Aided Design*, 23(7), 1991.

[20] H. Voelker. A current perspective on tolerancing and metrology. *Manufacturing Review*, 6(4), 1993.

[21] J. Wiegley, K. Goldberg, M. Peshkin, and M. Brokowski. A complete algorithm for designing passive fences to orient parts. *Assembly Automation*, 17(2), August 1997.

[22] C. Yap and E.-C. Chang. Geometric tolerancing: Theory, issues, and computation. In *Second Workshop on Algorithmic Foundations of Robotics*, Toulouse, France, July 1996.

Taylor & Francis
Taylor & Francis Group
http://taylorandfrancis.com

Towards Planning for Elastic Objects

Lydia E. Kavraki *Department of Computer Science, Rice University, Houston, TX, USA*
Florent Lamiraux *Department of Computer Science, Rice University, Houston, TX, USA*
Christopher Holleman *Department of Computer Science, Rice University, Houston, TX, USA*

This paper investigates the problem of path planning for a thin elastic plate. The underlying geometric model for the plate is provided by a Bézier representation. The geometric model is augmented by a realistic mechanical model. The latter permits the computation of the shape of the plate with respect to a set of grasping constraints by minimizing the elastic energy of the deformation. We use a probabilistic roadmap planner to compute paths for the plate and we present a number of experimental results to illustrate our approach. Our work is a first step towards considering the physical properties of objects when planning.

1 Introduction

The problem of planning a path for a robot consisting of rigid parts has been studied extensively over the last decade [17, 21]. This paper describes a first step in the direction of solving a variant of the above problem, namely the problem of planning a path for a flexible robot/part. We focus on the case of an elastic metal plate to illustrate our approach and explore some of the issues arising when considering flexible parts.

Several important applications motivate our research: in industrial settings there is a need to manipulate sheets of metal [26], pipes that can bend [31], and cables [25]. In assembly maintainability studies done with virtual prototyping, planning is used to remove a part from an assembly given only the CAD model of the assembly [10]. The flexibility of the part needs to be considered as engineers use deformable parts to produce compact assemblies [9]. Last but not least, our work may have applications in domains like computer generated animation, where physical properties can help generate realistic sequences, and virtual environments for surgical training, where flexible medical tools are guided inside the human body.

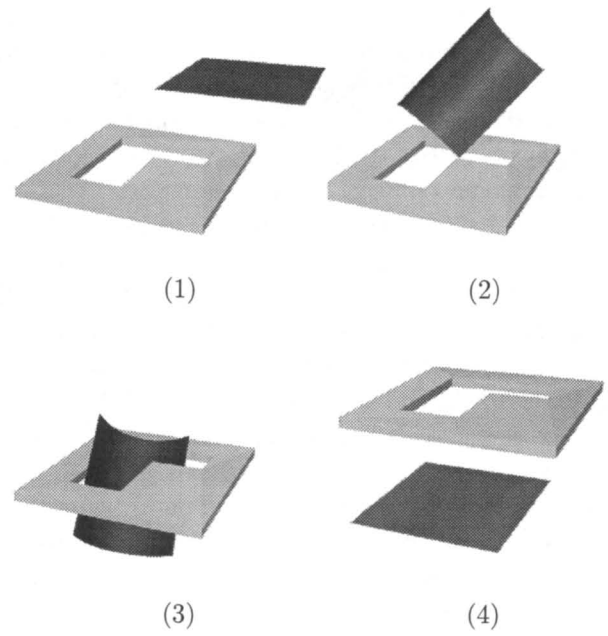


Figure 1: *Snapshots along a path computed by f-PRM for an elastic metal plate that can only bend.*

This paper addresses the problem of planning paths among obstacles for a thin elastic metal plate. We assume that the plate is manipulated by two wide grippers that can bend the plate. Although our work is not restricted to manipulation that induces only bending, we focus on this case to discuss our approach in detail. The geometry of the plate is modeled using the Bézier representation, while deformations corresponding to grasping conditions are computed by minimizing the elastic energy of the plate. The elasticity limit of the material of the plate is taken into account and

we reject deformations that may cause a permanent change in its shape. The actual robots performing the manipulation are not considered. Work in a similar spirit is done in assembly sequencing, where removal paths for parts are computed without taking into account the tools needed to perform the removal [10, 39].

We plan paths for the elastic plate using an extension of the Probabilistic RoadMap planner (PRM) in [19, 20, 28, 33]. PRM was chosen because it is one of the few planners that can deal efficiently with high dimensional planning problems [20]. The geometric representation of deformations requires in general a large number of parameters and these specify degrees of freedom (dof) for the surface. We call our new planner flexible-PRM (f-PRM). f-PRM builds a roadmap in the configuration space of the plate. The nodes of the roadmap are computed by first selecting random values for the parameters that specify the manipulation constraints and then obtaining an equilibrium configuration for these values. The edges of the roadmap are low-energy paths computed by a local planner. Once a sizable roadmap has been produced, f-PRM attempts to answer a user query by connecting the initial and the final configurations to the roadmap and then searching the roadmap for a sequence of paths between these nodes. A few snapshots along a path produced by f-PRM for a surface patch that can only bend (a 7-dof problem) are given in Figure 1.

Although in this paper we concentrate on the case of an elastic plate which is represented by a Bézier surface and can only bend, our approach is not restricted to the specific energy model, geometric representation, and grasping constraints used. As a proof of concept, we use our framework to plan paths for a surface patch that can assume a wide range of deformations due to less restrictive energy and manipulation constraints. This case, which is separately investigated in [18], may be of interest in computer generated animation. We report experiments solving a 27-dof planning problem with few modifications of our original planner.

Our work blends ideas from planning for high-dimensional problems (PRM) with work in geometric modeling (representations of curves and surfaces) and mechanics/physics (energy models for elasticity or other physical properties). The randomized framework of f-PRM provides an excellent testbed for investigating several novel issues that arise in the context of planning with flexible objects. These include (a) understanding the implications of using different geometric representations for the objects, (b) acquiring meaningful energy models, (c) studying manipulation of flexible objects and its effect on the attainable deformations of these objects, (d) devising algorithms for planning low-energy paths between configurations with different deformations, (e) investigating approaches to efficient collision checking when the shape of the object changes, and (f) developing methods for improving the overall quality of the paths.

This paper is organized as follows. Section 2 briefly surveys related work. Section 3 addresses geometric and mechanical issues related to the representation of an elastic plate and its manipulation. Section 4 outlines the general principles of f-PRM. We present in Section 5 experimental results for an elastic surface that can only bend. We discuss in Section 6 how to use f-PRM for planning with a less restrictive energy model.

2 Related Work

Path planning for a flexible surface defines a high-dimensional problem and fortunately much progress has been made in robotics on this front. However little has been done to accommodate the physical properties of robots such as their elasticity. Issues related to object deformation have been studied mostly in the areas of mechanics, geometric modeling, and graphics. We briefly survey separately each of the above areas.

Robotics Recent work on the path planning problem has produced several practical planners for robots that consist of rigid parts (see for example [1, 2, 4, 11, 16, 20]). These methods routinely take into account geometric constraints such as joint limits and obstacles, but also constraints arising from kinematics such as nonholonomic velocity constraints due to the rolling without slipping of wheeled mobile robots [22], or constraints over the radius of curvature of a car-like system [23, 33].

With the exception of the areas of dynamics and control that have guided the design of modern robots [30], there are few cases where physical constraints and planning have been tightly coupled. Kino-dynamic motion planning is one such case as it takes into account dynamic constraints by planning in the tangent space of the configuration space [12, 13].

As far as deformable robots and parts are concerned, a lot of work has been done in the context of manipulation. Robots with flexible links are now being built since they facilitate certain tasks (like hammering a peg into a hole) and their modeling and control is under development (for pointers to current work see [26]). Recent papers consider the dynamic analysis of robots with flexible payloads such as two robots manipulating a flexible metal sheet [26] or a vibrating object [31], or solve the task of inserting one end of a flexible wire into a hole while holding the other end [25]. Furthermore, research in snake-like robots has explored issues related to "geometric mechanics" that are relevant to our discussion [7, 27]. For example, the work [27] describes the net motion of a snake robot as a function of variations in the mechanism's shape variables.

Mechanics In mechanics there exists a large amount of research on modeling physical properties such as elasticity [6]. The work in [38] discusses the case of thin plates and develops an energy model for the deformation of a thin elastic plate that depends only on the planar deformation and the curvature of the plate.

Geometric modeling In geometric modeling several representations for curves and surfaces have been developed to enable accurate manipulation of shape while considering a relatively small number of parameters [5, 14]. In this paper we use a simple model from computer-aided geometric design for our flexible surfaces, namely the tensor product Bézier surface. Other models may also be appropriate and this issue is currently under investigation.

Graphics In graphics physically based models have been proposed for deformable parts [34, 35]. The use of physical simulation and related optimization techniques as a means of geometric interaction has been applied to animation [36], drawing [37], free-form surface and volume modeling [8], mechanical design [40], and interactive molecular simulation [32]. For a discussion on the dynamic simulation of non-penetrating flexible bodies see [3].

3 Deformation of an Elastic Plate

Before considering planning, we present the geometric and mechanical model used for the elastic plate. We assume that the plate is manipulated by two robot grippers. In this paper we do not consider the full version of the manipulation problem: we only specify conditions, which we call *grasping constraints* or *limit conditions*, and which reflect the effect of manipulation on our object.

Let $F_A = (\vec{w}_1, \vec{w}_2, \vec{w}_3)$ be an orthonormal reference frame for the *undeformed configuration* of the plate, which we also refer to as the *load-free configuration.* In that configuration the plate occupies the volume

$$\begin{cases} 0 & \leq & x_1 & \leq & L \\ -\frac{W}{2} & \leq & x_2 & \leq & \frac{W}{2} \\ -\frac{h}{2} & \leq & x_3 & \leq & \frac{h}{2}. \end{cases}$$

We assume that h is small compared to L and W and we represent the geometry of the plate by a surface. However, we take into account the thickness h of the plate in all energy calculations.

3.1 Geometric model for the plate

Tensor product Bézier surfaces are used to represent our plates. The main advantage in choosing the Bézier representation is that several grasping constraints can easily be specified with the Bézier representation as discussed in Section 3.2.

A Bézier surface is defined by a $n \times m$ control net, made up of a grid of points in 3-space. Let $P_{ij}, i = 0, \ldots, n, j = 0, \ldots, m$ be these points, also called the control points or Bézier points. The tensor product approach obtains a point on the surface as

$$S(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} B_n^i(u) B_m^j(v) P_{ij}, \qquad (1)$$

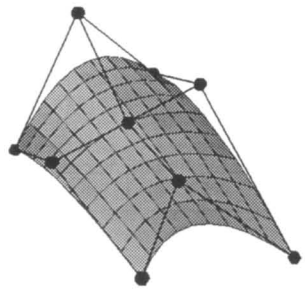


Figure 2: *A biquadratic Bézier surface patch with its 9 control points and several isolines.*

where $u \in [0,1], v \in [0,1]$, and $B_n^i(u), B_m^j(v)$ are the Bernstein polynomials defined by

$$B_n^i(u) = \binom{n}{i} u^i (1-u)^{n-i}.$$

In the following, we will denote by $t_u = \partial S/\partial u$ and $t_v = \partial S/\partial v$ the tangent vectors to the surface. In the load free configuration, the control points have the following coordinates $P_{ij} = (\frac{i}{n}L, \frac{j}{m}W, 0)$.

The surface contains the four corner control points. The rest of the control points are not necessarily on the surface but affect its shape. A simple illustration of a biquadratic surface is given in Figure 2.

3.2 Grasping constraints for manipulation

We consider now a plate made of an elastic material and we act on it by grasping two opposite edges using for instance two wide grippers. Let us assume that we can control the distance $d < L$ between these parallel grippers as shown on Figure 3. The elasticity hypothesis states that there exists a mapping from the space of deformations of the plate into $\mathbf{R}_+$ called the *elastic energy* of the plate. An equilibrium deformation is a local minimum of the elastic energy in the space of deformations fitting the limit conditions imposed by a given grasping. We call *simple bending* the set of deformations reachable by these grasping conditions. Simple bending induces deformations for which the tangent vector t_v is constant over the surface and equal to its value in the undeformed configuration. Any deformation can then be obtained from the curve $S(u,0)$ by

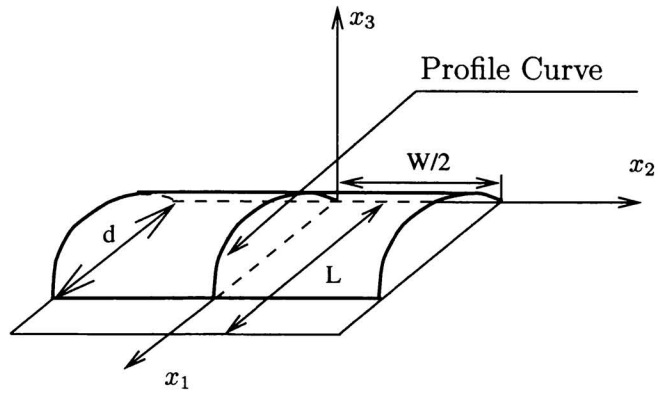


Figure 3: *Grasping by two opposite parallel edges.*

the following translation

$$S(u,v) = S(u,0) + (v - \frac{1}{2})W\vec{w}_2.$$

We can thus represent this deformation by a Bézier curve in the $(\vec{w}_1, \vec{w}_3)$-plane. That is

$$S(u) = \sum_{i=0}^{n} B_n^i(u) P_i,$$

with $u \in [0,1]$. We call this curve the *profile curve* of the surface (see Figure 3). The grasping constraint implies $S(0) = P_0$, $S(1) = P_n$. The position of the other control points is computed by minimizing the *elastic energy* of the plate as defined in Section 3.4.

3.3 Geometric analysis of deformations

We now define the deformation coefficients appearing in the expression of elastic energy. Our discussion draws from [38] which shows that the deformation of a thin elastic plate depends only on the planar deformation and the curvature of the plate. For a general deformation, 6 coefficients are required: 3 for the planar strain and 3 for the curvature. Under simple bending, however, we need only consider the deformations of a curve. In this case, all but one of the strain coefficients and one of the curvature coefficients are zero. We analyze below the simple bending case and provide some intuition for the meaning of the coefficients involved in the expression for elastic energy. We treat the general 2D case in the Appendix.

Let us consider the profile curve $S(u)$ as shown in Figure 3. In the load free configuration, the position of the control point P_i is $(iL/n, 0, 0)$ and the corresponding Bézier curve is given by $S(u) = (Lu, 0, 0)$. The tangent vector $t_u(u) = S'(u)$ to this curve is thus constant and equal to $(L, 0, 0)$. As shown in Figure 4, a deformation can be defined by an extension coefficient ε and a curvature χ. The expressions of these coefficients are

$$\varepsilon(u) = \frac{1}{L}(\|t_u(u)\| - L), \quad (2)$$

$$\chi(u) = \frac{\|t_u(u) \times t'_u(u)\|}{\|t_u(u)\|^3}, \quad (3)$$

where $t'_u(u) = S''(u)$ is the derivative of $t_u(u)$ in the deformed configuration and $\times$ is the cross product. It can easily be seen that ε represents the extension rate of a small piece of matter of length dl_0 in the $\vec{w}_1$ direction. That is $\varepsilon = (dl - dl_0)/dl_0$, where dl is the length of the small piece of matter after deformation (see Figure 4(b)). Equation (3) is the standard formula for curvature, while Figure 4(c) points out that bending the plate induces extension and compression above and below the medium surface. From now on, we will denote the *strain vector* by

$$\mathbf{e}(u) = (\varepsilon(u), \chi(u)).$$

3.4 Mechanical analysis of deformations

Elasticity theory asserts that for a general surface the *elastic energy* of a deformation is obtained by integrating the *elastic energy per unit of surface* $\psi(\mathbf{e}(x))$ over the surface, where $\mathbf{e}(x)$ is the previously defined strain vector at some point x of the surface.

Under our hypothesis of one-dimensional deformation, the elastic energy per unit of surface w.r.t. the local deformation of the plate is given by [38]

$$\psi(\mathbf{e}) = \psi^{\chi}(\chi) + \psi^{\varepsilon}(\varepsilon), \quad (4)$$

where

$$\psi^{\chi}(\chi) = \frac{Eh^3}{24}\frac{\chi^2}{1-\nu^2}, \quad (5)$$

$$\psi^{\varepsilon}(\varepsilon) = \frac{Eh}{2}\frac{\varepsilon^2}{1-\nu^2}. \quad (6)$$

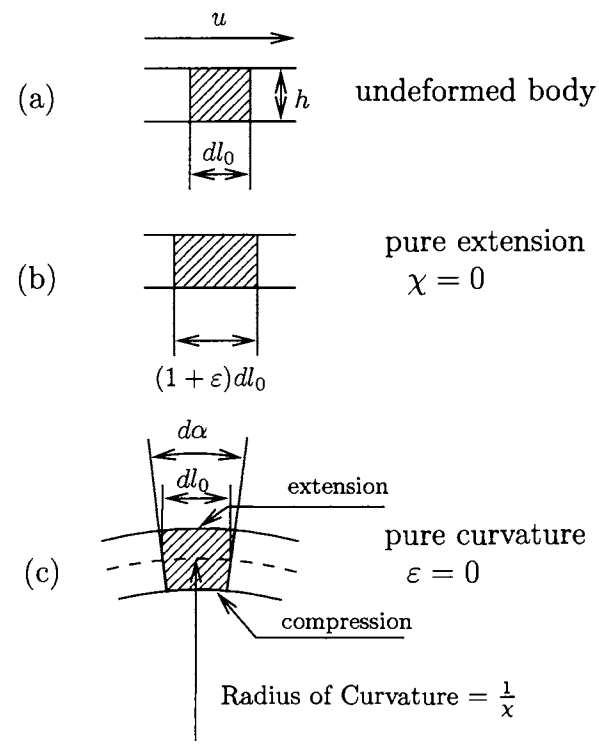


Figure 4: *Extension and curvature coefficients.*

E and ν are coefficients characterizing the elasticity of the material[1]. They are respectively called the Young Modulus and Poisson ratio.

3.5 Equilibrium deformations

An equilibrium deformation is a local minimum of the elastic energy among the deformations fitting the limit conditions imposed by grasping and thus can be computed by minimizing the elastic energy over the free parameters of the geometric representation of the part.

In the simple bending case, the grasping condition determines the position of two control points of the profile curve: P_0 and P_n. Hence, we are required to minimize the elastic energy over the $n-1$ free control points. To perform this minimization, we use the conjugate gradient method [29]. The gradient of the elastic energy is obtained by integrating over the plate the partial derivatives of (4) with respect to the free

[1]For instance, for aluminum, $E = 74$ MPa and $\nu = 0.34$.

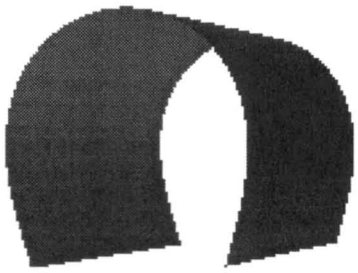


Figure 5: *An equilibrium deformation of a surface with a profile curve of 10 control points.*

control points. Integrals are computed using the Simpson formula [29].

Figure 5 shows the result of the minimization of the elastic energy for a user-specified d, under the simple bending condition and with a profile curve of 10 control points. Notice that the behavior of the surface is the expected one and that the symmetrical surface with respect to the plane defined by the grasping edges is also an equilibrium configuration.

3.6 Elasticity limit

The elastic model we use is valid only for small deformations. Out of the elasticity domain, the material has an elasto-plastic behavior. That is, the unconstrained shape of the object is different before and after the deformation and the material is irreversibly affected. We need to avoid this behavior when planning and ensure that all deformations lie in the elasticity domain. We explain now how to express the latter condition in terms of the coefficients ε and χ of the strain vector.

Figure 4(c) shows how the curvature of the medium surface implies compression and extension within the thickness of the plate. Thus, for a curvature χ and an extension $\varepsilon = 0$, the maximal strain occurs on the upper and lower surfaces. The straight line of undeformed length $dl_0 = Rd\alpha$ is extended on the upper surface to $(R+h/2)d\alpha$, where $R = 1/\chi$ is the radius of curvature. Thus, the extension rate is equal to $\chi h/2$ on the upper surface, and to $-\chi h/2$ on the lower surface. If ε is not zero, the extension rate on the upper and lower surface are equal to the sum of ε and these latter values. (This is a first order approximation [38]).

Therefore, the condition under which all particles of the plate remain in the elasticity domain is

$$|\varepsilon| + \frac{h}{2}|\chi| < \varepsilon_{max}, \tag{7}$$

where ε_{max} is a constant depending on the material.

In our application, this constraint corresponds to bounding from below the value of the bending parameter d

$$d_{min} < d < L. \tag{8}$$

For most metals, ε_{max} is very small, i.e. $< 10^{-3}$. Note however that if the plate is very thin, a large curvature can occur, as can be seen from equation (7) and illustrated in Figure 5. As a final remark, let us point out that to determine precisely the forces and strain within a material, we would need high precision models (possibly finite elements with hundreds of points). However, for the purposes of our work it is enough to reasonably approximate the shape of the surface and our model achieves this approximation.

4 Planning: Description of f-PRM

Our planning method, f-PRM, extends many of the ideas of the probabilistic roadmap approach to planning [19, 20, 33]. The planner repeats a basic step until a query is answered or until an allowed amount of time has elapsed. The method can be seen as a single shot method (answering a single user query), or as a method that consists of a preprocessing and a query processing phase. In both cases, the approach is incremental: the roadmap constructed in the free part F of the configuration space can always be augmented by adding more nodes and more paths, hopefully capturing in greater detail the connectivity of F and allowing for a larger set of user queries to be answered.

Preprocessing: f-PRM constructs a roadmap $R = (V, E)$ in F by repeating the basic step shown in Figure 6. A few comments are in order.

Preprocessing (basic step):

1. Generate a low-energy deformation uniformly at random in F. Then generate N random configurations having this deformation.
2. Update the graph $R = (V, E)$ in which E consists of connections found by a local planner. Each node is tried for connection with its K closest neighbors among all nodes generated in V.
3. Identify "difficult" areas and refine sampling in these areas by generating M more nodes. Connect the new nodes to R and update R.

Figure 6: *Roadmap construction.*

• At each basic step many configurations with the same deformation are generated. This allows us to create (and later search) for paths that do not unnecessarily deform the plate.

• During step (2) each node c is tried for connection with its K closest neighbors among the nodes in V. These neighbors may be of the same or different deformation as c. To find the neighbors, f-PRM uses a distance metric $D(\cdot,\cdot)$ that takes into account the rigid transformations and deformations of the configurations given as its arguments. Using K makes the running time of the algorithm linear in the number of generated nodes. A simple and deterministic local planner is used for the interconnections (see discussion in [20]).

• During step (3), which is also called the node enhancement step, f-PRM identifies *difficult areas* using the following heuristic scheme: a configuration c is chosen from V with probability

$$Pr(c \text{ is selected}) = w(c),$$

and a random configuration is generated in the neighborhood of c. The weights $w(\cdot)$ used above are chosen as in [19]

$$w(c) = \frac{1}{d_c + 1} \Big/ \sum_{t=1}^{N} \frac{1}{d_t + 1},$$

where d_c is the degree of node c, that is the number of connections that c has with other nodes. To generate a random node in the neighborhood of c, we proceed as follows: we start a random walk from node c and perform a predefined number of reflections. For each reflection, we generate a random direction and move in that direction until the robot collides with an obstacle, or a predefined number of steps have been executed. The end configuration of the random walk is added to R and is tried for connection with its neighboring configurations of any deformation in R.

Query processing: given R, a starting configuration s, and a goal configuration g, f-PRM attempts to connect s and g to two close-by nodes s' and g' that belong to the same connected component of R. If successful, the component is searched for a sequence of edges from s' to g'.

5 Planning in the Presence of Manipulation Constraints

Using the elasticity model for a thin plate specified in Section 3, we proceed to define and solve planning problems under the *simple bending* hypothesis. We first outline our implementation choices for f-PRM and then describe our experiments.

5.1 Implementation choices

Generation of random configurations: the representation of a configuration is broken down into two parts (a) the deformation and (b) the rigid body transformation. The deformation is specified by the coordinates of the control points in the reference frame F_A of the load-free configuration. The rigid body transformation is given by a translation and a rotation vector. Decoupling deformations and rigid body transformations allows using the same shape at different positions in the workspace and facilitates collision checking as discussed below.

In the case of simple bending it is enough to specify $d \in [d_{min}, L]$ to obtain a deformation. In our current implementation, we discretize the interval of possible values of d into 200 different values. For each value, we

precompute the corresponding deformation of the profile curve by minimizing the elastic energy. The surface is obtained from the profile curve as specified in Section 3.2. We choose a deformation randomly among the 200 generated off-line. A random rigid body transformation is applied to the deformation. The discretization of d was done to speed up computations and is by no means restrictive.

Collision checking: collision checking is implemented with the RAPID library [24]. This library takes as input collections of triangles describing the environment and the moving object. In our implementation, the plate is approximated by a grid of points evenly sampled across the surface. These points define triangles that are used by RAPID. The obstacles are also decomposed into triangle soups. Once an internal model of the surface and a model of the obstacles have been created by RAPID, a configuration can be queried for collision by specifying a rigid transformation for both models. The creation of an internal model of the surface is expensive compared to the actual collision checks. By keeping the shape of the surface separate from its location in the workspace, the internal model for any deformation can be built once and reused, speeding up collision checking.

Distance measure: let c_1, c_2 be two configurations of the surface. Let $P_i^{c_1}$, $P_i^{c_2}$ denote the positions of their control points, s_1, s_2, the translation vectors of their rigid transformations and θ the rotation angle to obtain c_2 from c_1. The distance D of c_1 and c_2 is given by

$$D(c_1, c_2) = ||s_1 - s_2|| + |\theta| l_{max} + max_i\{||P_i^{c_1} - P_i^{c_2}||\}.$$

In the above, l_{max} denotes half of the diagonal length of the undeformed plate. This term approximates the greatest extent of the surface (see [4] for handling rotation).

Local planner: to connect c_1 and c_2 the local planner first finds the axis of rotation and angle of rotation needed to get from one configuration to the other. Then a path is constructed by linear interpolation of the translation and rotation vector between c_1 and c_2.

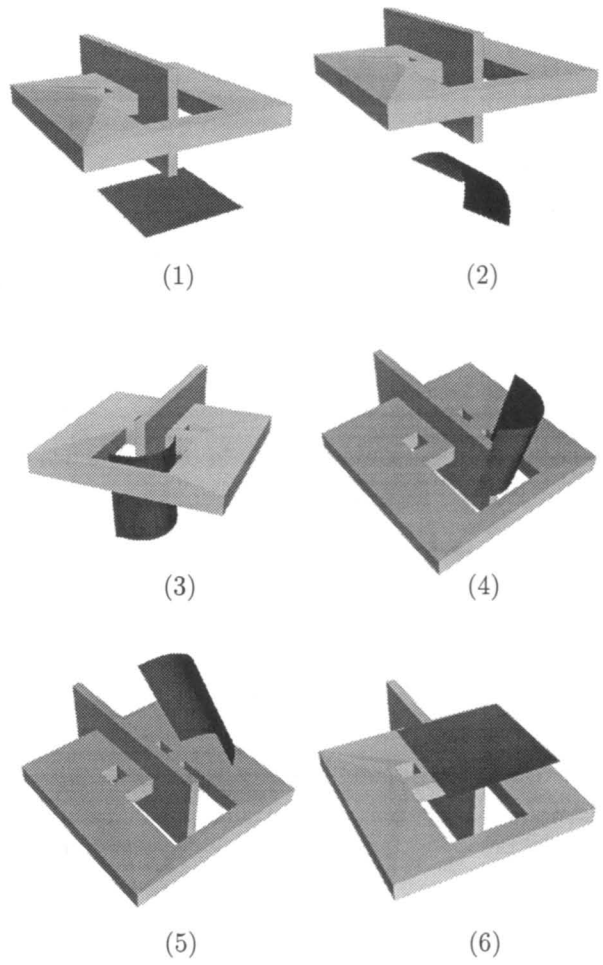


Figure 7: *Experiment with a surface that can only bend.*

Subsequently, if the deformations of c_1 and c_2 are different, a linear interpolation on the parameter d is used to obtain deformation c_2 from that of c_1. Intermediate deformations are read from our precomputed array.

5.2 Experimental results

Our code is written in C++ and all the timing results reported here were obtained on an SGI R10000. Our implementation follows closely the description in Section 4 with the exception that enhancement is started after a few basic steps; the number of nodes generated in each basic step is small and our heuristic scheme for enhancement is not very efficient with small roadmaps.

Figure 7 shows snapshots along a path computed by f-PRM for a 10 control point Bézier curve. The parameters for the basic step of f-PRM are $N = 100$, $M = 100$, and $K = 18$. We assume that we are treating a metal sheet plate and we bound d_{min} to reflect its elasticity limits. During enhancement the random walk consists of a maximum of 10 reflections, each of which can be 100 steps long. We run f-PRM 10 different times changing the value of the random seed generator. The planner reliably solved the problem all 10 times with an average running time of 3.3 min. Note that we solve a considerably difficult problem. The polyhedral obstacle which obstructs the rectangular hole of the workspace in Figure 7 extends beyond the hole (from below and above). This fact restricts the deformations with which the surface can go through the hole. Another example is shown in Figure 1. Without changing any of the parameters of our planner we can solve that problem with an average running time of 1.2 min. The precomputation of the array of deformations took 22 min.

6 Planning in the Absence of Manipulation Constraints

We now discuss the application of f-PRM to a case with less restrictive manipulation constraints and energy model. In particular we assume that each control point of the Bézier surface patch is free to move by itself and the energy model simply disallows extensive shear and bending of the surface. We use this example to illustrate the capabilities of our framework by solving high dimensional planning problems. A detailed discussion can be found in [18]. Note that this line of work may be relevant in computer generated animation. The energy model can be used to make the motion look "natural" and the specified constraints would typically be different from manipulation constraints.

6.1 Implementation choices

We use a 3×3 Bézier surface patch for our experiments. To specify a deformation, we need to specify the coordinates of the control points of the surface. Thus this problem has 27 dof. We describe below those implementation choices that are different from the ones of the elastic surface that can only bend.

Energy model: the energy model excludes deformations that correspond to unreasonable bending, extension, or shear of the surface. We divide the energy into three parts $E_{curv}, E_{area}, E_{shear}$ to reflect energy due to curvature, area variation, and shear. These terms are computed on a 10×10 grid of points sampled evenly across the surface. Let x_{ij}, $i, j = 1, \ldots, 10$ be the grid points, and let x_{ij}^{co}, $i, j = 1, \ldots, 10$ be the corresponding grid points at the load-free shape for referring to properties of that shape. Then

$$\begin{aligned} E_{curv} &= max_{ij}\{\sin^2(\text{angle}((x_{ij} - x_{ij-1}), (x_{ij+1} - x_{ij})))\} \\ E_{area} &= max_i\{\textstyle\sum_j |x_{ij} - x_{ij+1}| - \sum_j |x_{ij}^{co} - x_{ij+1}^{co}|\} \\ E_{shear} &= max_{ij}\{\cos^2(\text{angle}((x_{i+1j} - x_{ij}), (x_{ij+1} - x_{ij})))\}. \end{aligned}$$

Once the energy is computed, each component is compared to a predefined cutoff value.

Generation of random configurations: a deformation is specified by the position of each control point P_{ij}. To avoid generating huge deformations, we displace each of the P_{ij}'s of the load free configuration by a vector whose magnitude is bounded and whose orientation is chosen at random. This idea is explored further in [18]. Again, random rigid transformations can be applied to a deformation to obtain random configurations.

Local planner: the local planner simultaneously interpolates the rigid transformations and deformations of the corresponding configurations. For the deformation, the position of each control point is interpolated in $\mathbf{R}^3$.

6.2 Experimental results

We run f-PRM 10 different times changing the value of the random seed generator for the workspace of Figure 8. The workspace and surface dimensions are the same as the ones used in Section 5.2. The parameters of

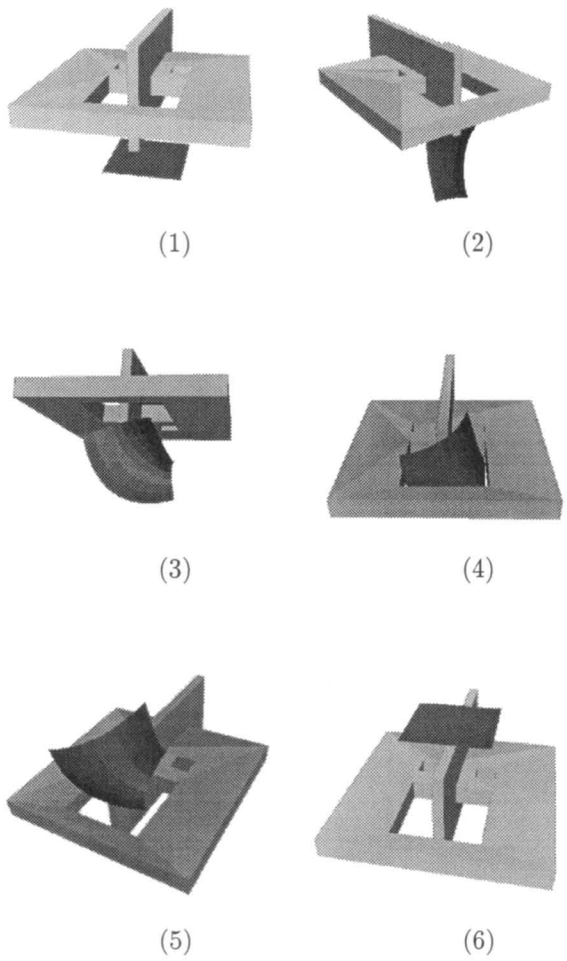


Figure 8: *Snapshots along a path computed by f-PRM.*

f-PRM are kept the same as in Section 5.2. The planner reliably solved the unconstrained problem all 10 times with an average time of 4.7 hours. Note that we now solve a 27-dof problem so we expect to have much higher running times than those of Section 5.2. Keeping the parameters of the planner the same, we find that the average running time for solving the problem of Figure 1 is 9.1 min, further emphasizing the difficulty of the example in Figure 8.

7 Discussion

This paper investigates the problem of path planning for a thin elastic metal plate. We used the Bézier representation for describing the geometry of the plate and provided a model for its elastic energy. We compute the shape of the plate with respect to a set of grasping constraints by minimizing the elastic energy of the deformation. A probabilistic roadmap approach is developed to compute paths for the plate. Our experiments show that our new planner, f-PRM, can successfully solve high-dimensional problems, but can also deal with highly constrained and realistic models when the deformation is controlled by manipulation constraints. We also extended our framework to the case of less restrictive energy models and manipulation constraints that typically require solving higher dimensional planning problems.

Our work is a first step in the direction of considering object flexibility in general, and elasticity in particular, during planning. It raises a number of interesting issues which we expect to address in the future. These include: studying different geometric representations for the flexible objects, allowing for more complex manipulation constraints, developing efficient strategies for planning local paths, and devising methods that smooth computed paths or possibly relax the energy of the object along these paths.

Acknowledgments

Lydia Kavraki and Florent Lamiraux are partially supported by NSF CAREER Award IRI-970228. Christopher Holleman is supported on RUE IRI-970228. The authors would like to thank Joe Warren for many helpful discussions and creative ideas for this work, and Jean-Claude Latombe and Leo Guibas for their comments. The comments of the anonymous referees were also very helpful to improve the presentation of this paper.

References

[1] J. M. Ahuactzin, E.-G. Talbi, P. Bessière, and E. Mazer. Using genetic algorithms for robot motion planning. In *10th Europ. Conf. Artific. Intelligence*, pages 671–675, London, England, 1992.

[2] N. Amato, O. Burchan Bayazit, L. Dale, C. Jones, D. Vallejo, "OBPRM: an obstacle-based PRM for 3D

Workspaces", *Preprints of the 3rd Workshop on the alg. found. of Rob.*, Houston, TX, 1998.

[3] D. Baraff and A. Witkin. Dynamic simulation of non-penetrating rigid bodies. *Computer Graphics (SIGGRAPH'92)*, pages 303–308, 1992.

[4] J. Barraquand and J.-C. Latombe. Robot motion planning: A distributed representation approach. *Int. J. of Robotics Research*, 10:628–649, 1991.

[5] R. Beach. *An introduction to the Curves and Surfaces of Computer-Aided Design.* Van Nostrand-Reinhold, NY, 1991.

[6] S. Borg. *Fundamentals of Engineering Elasticity.* World Scientific Publishing Company, Singapore, 1990.

[7] J. Burdick, J. Radford, and G. Chirikjian. A sidewinding locomotion gait for hyper-redundant robots. In *Proc. IEEE Int. Conf. on Rob. and Autom.*, pages 101–106, Atlanta, GA, 1993.

[8] G. Celniker and D. Gossard. Deformable curve and surface finite-elements for free-form shape design. *Computer Graphics (SIGGRAPH'91)*, pages 257–266, 1991.

[9] H. Chang. Personal Communication.

[10] H. Chang and T. Li. Assembly maintainability study with motion planning. In *Proc. IEEE Int. Conf. on Rob. and Autom.*, pages 1012–1019, 1995.

[11] P. Chen and Y. Hwang. Sandros: A motion planner with performance proportional to task difficulty. In *Proc. of IEEE Int. Conf. Robotics and Automation*, pages 2346–2353, Nice, France, 1992.

[12] B. Donald and P. Xavier. Provably good approximation algorithms for optimal kinodynamic planning: Robots with decoupled dynamics bounds. *Algorithmica*, 14:443–479, 1995.

[13] B. Donald, P. Xavier, J. Canny, and J. Reif. Kinodynamic motion planning. *J. of the ACM*, 40:1048–1066, 1993.

[14] G. Farin. *Curves and Surfaces for Computer-Aided Geometric Design.* Academic Press, San Diego, 1988.

[15] K. Goldberg, D. Halperin, J.-C. Latombe, and R. Wilson, editors. *Algorithmic Foundations of Robotics.* A K Peters, Ltd., 1995.

[16] K. Gupta and X. Zhu. Practical motion planning for many degrees of freedom: A novel approach within sequential framework. In *Proc. of IEEE Int. Conf. Robotics and Automation*, pages 2038–2043, San Diego, CA, 1994.

[17] D. Halperin, L. Kavraki, and J.-C. Latombe. Robotics. In J. Goodman and J. O'Rourke, editors, *Discrete and Computational Geometry*, pages 755–778. CRC Press, NY, 1997.

[18] C. Holleman, L. Kavraki, and J. Warren. Planning paths for a flexible surface patch. In *Proc. IEEE Int. Conf. Robotics and Automation*, 1998.

[19] L. Kavraki and J.-C. Latombe. Randomized preprocessing of configuration space for fast path planning. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 2138–2145, San Diego, CA, 1994.

[20] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars. Probabilistic roadmaps for fast path planning in high dimensional configuration spaces. *IEEE Tr. on Rob. and Autom.*, 12:566–580, 1996.

[21] J.-C. Latombe. *Robot Motion Planning.* Kluwer, Boston, MA, 1991.

[22] J.-P. Laumond. Feasible trajectories for mobile robots with kinematic and environment constraints. In *Proc. Intelligent Autonomous Systems*, pages 346–354, 1987.

[23] J.-P. Laumond, P. Jacobs, M. Taix, and R. Murray. A motion planner for nonholonomic mobile robots. *IEEE Tr. on Rob. and Autom.*, 10:577–593, 1994.

[24] M. Lin, D. Manocha, J. Cohen, and S. Gottschalk. Collision detection: Algorithms and applications. In Goldberg et al. [15], pages 129–141.

[25] H. Nakagaki and K. Kitagaki. Study of deformation tasks of a flexible wire. In *Proc. IEEE Int. Conf. on Rob. and Autom.*, Albuquerque, NM, 1997.

[26] W. Ngugen and J. Mills. Multi-robot control for flexible fixtureless assembly of flexible sheet metal auto body parts. In *Proc. IEEE Int. Conf. on Rob. and Autom.*, pages 2340–2345, Minneapolis, MN, 1996.

[27] J. Ostrowski and J. Burdick. Gait kinematics for a serpentine robot. In *Proc. IEEE Int. Conf. on Rob. and Autom.*, pages 1294–1299, Minneapolis, MN, 1996.

[28] M. Overmars and P. Svestka. A probabilistic learning approach to motion planning. In Goldberg et al. [15], pages 19–37.

[29] W. H. Press, S. A. Teutolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C.* Cambridge University Press, 1995.

[30] M. Spong and M. Viyasagar. *Robot Dynamics and Control.* John Wiley and Sons, NY, 1989.

[31] D. Sun, X. Shi, and Y. Liu. Modeling and cooperation of two-arm robotic system manipulating a deformable object. In *Proc. IEEE Int. Conf. on Rob. and Autom.*, pages 2346–2351, Albuquerque, NM, 1996.

[32] M. Surles. An algorithm with linear complexity for interactive physically-based modeling of large proteins. *Computer Graphics*, 26:221–230, 1992.

[33] P. Svestka and M. Overmars. Motion planning for car-like robots using a probabilistic learning approach. *Int. J. of Robotics Research*, 16(2):119–143, 1997.

[34] D. Terzopoulos, J. Platt, A. Barr, and K. Fleischer. Elastically deformable models. *Computer Graphics (SIGGRAPH'87)*, 21(4):205–214, 1987.

[35] D. Terzopoulos and A. Witkin. Physically based models with rigid and deformable components. *IEEE Computer Graphics and Applications*, pages 41–51, November 1988.

[36] D. Terzopoulos, A. Witkin, and M. Kass. Energy constraints on deformable models: recovering shape and non-rigid motion. In *Proc. of AAAI 87*, Seattle, 1987.

[37] J. Thingvold and E. Cohen. Physical modeling with b-spline surfaces for interactive design and animation. *Computer Graphics (SIGGRAPH'90)*, 24(2):129–139, 1990.

[38] G. Wempner. *Mechanics of Solids with applications to thin bodies.* McGraw-Hill, NY, 1991.

[39] R. H. Wilson. *On Geometric Assembly Planning.* PhD thesis, Stanford Univ., March 1992. Stanford Technical Report STAN-CS-92-1416.

[40] A. Witkin and W. Welch. Fast animation and control of non-rigid structures. *Computer Graphics (SIGGRAPH'90)*, pages 243–252, 1990.

Appendix: Elasticity of a Plate

Elasticity has been studied extensively in mechanics [6, 38]. This theory is based on the existence of a potential energy function per unit of volume which depends on the temperature and deformation fields within the material. An elastic body can be deformed by external causes such as grasping which imposes constraints on the position of a subset of points of the body. The deformation field induced minimizes the elastic potential energy over the whole body.

For materials such as metals, the deformation field has to remain small to ensure the reversibility of the deformation. In this case, the *linear elastic* model is generally used. This model asserts that the elastic energy is quadratic with respect to the *strain field* of a deformation.

In the case of a thin plate, the elastic energy can be approximated by the deformation of the medium surface of the plate, ignoring for a moment the thickness of the plate. Then, the strain field, in analogy with the strain vector in the 1D case (see Section 3.3), has 3 components measuring planar strain and 3 components measuring curvature. Let us parameterize the undeformed surface as follows

$$S_0(x_1, x_2) = x_1\vec{w}_1 + x_2\vec{w}_2,$$

where $(\vec{w}_1, \vec{w}_2, \vec{w}_3)$ is an orthonormal basis. Let us also consider a deformation for which the point of coordinates (x_1, x_2) is transported to $S(x_1, x_2)$. The temperature is assumed to be constant. All the computations needed to establish the results below can be found in [38].

A. Strain field of the plate

The strain field of a deformation of a surface is defined, at each point of the surface, by six coefficients: three of these coefficients account for planar deformation while three account for curvature.

Planar strain coefficients The planar strain coefficients are computed by comparing the tangent vectors to the undeformed surface with those of the deformed one. Let us use the following notation for these tangent vectors

$$\begin{aligned} t_1(x_1, x_2) &= \frac{\partial S(x_1, x_2)}{\partial x_1} \\ t_2(x_1, x_2) &= \frac{\partial S(x_1, x_2)}{\partial x_2} \end{aligned}$$

$$\begin{aligned} t_1^0(x_1,x_2) &= \frac{\partial S_0(x_1,x_2)}{\partial x_1} \\ t_2^0(x_1,x_2) &= \frac{\partial S_0(x_1,x_2)}{\partial x_2}. \end{aligned}$$

The strain coefficients correspond to the difference between the inner products of the tangent vectors before and after deformation, as follows

$$\begin{aligned} e_{11} &= \tfrac{1}{2}(t_1 \cdot t_1 - t_1^0 \cdot t_1^0) = \frac{1}{2}(t_1 \cdot t_1 - 1) \\ e_{22} &= \tfrac{1}{2}(t_2 \cdot t_2 - t_2^0 \cdot t_2^0) = \frac{1}{2}(t_2 \cdot t_2 - 1) \\ e_{12} &= \tfrac{1}{2}(t_1 \cdot t_2 - t_1^0 \cdot t_2^0) = \frac{1}{2}(t_1 \cdot t_2). \end{aligned}$$

e_{11} and e_{22} are called the *extension coefficients* in the x_1 and x_2 directions, whereas e_{12} is the *shear* between these directions.

Notice that the coefficient ε of the one-dimensional deformation of Section 3 is a first order approximation of the real extension coefficient in the u direction.

Curvature coefficients To express the curvature, we need the normalized tangent and normal vectors to the deformed surface

$$\begin{aligned} \bar{t}_1 &= t_1/\|t_1\| \\ \bar{t}_2 &= t_2/\|t_2\| \\ n &= (\bar{t}_1 \times \bar{t}_2)/\|\bar{t}_1 \times \bar{t}_2\|. \end{aligned}$$

The curvature of the deformed surface can be represented by the three following coefficients

$$\begin{aligned} \chi_{11} &= \frac{1}{\|t_1\|}\frac{\partial \bar{t}_1}{\partial dx_1} \cdot n \\ \chi_{22} &= \frac{1}{\|t_2\|}\frac{\partial \bar{t}_2}{\partial dx_2} \cdot n \\ \chi_{12} &= \frac{1}{\|t_1\|}\frac{\partial \bar{t}_2}{\partial dx_1} \cdot n = \frac{1}{\|t_2\|}\frac{\partial \bar{t}_1}{\partial dx_2} \cdot n. \end{aligned}$$

B. Elastic energy of the plate

We can group the coefficients defined above in two matrices to simplify the expression of the elastic energy

$$e_s = \begin{pmatrix} e_{11} & e_{12} \\ e_{12} & e_{22} \end{pmatrix}, \quad \chi = \begin{pmatrix} \chi_{11} & \chi_{12} \\ \chi_{12} & \chi_{22} \end{pmatrix}.$$

These matrices represent in fact the *planar strain tensor* and the *curvature tensor*. With this notation, the elastic energy is the sum of two terms:

$$\psi_s(e_s,\chi) = \psi_s^e(e_s) + \psi_s^\chi(\chi),$$

with

$$\psi_s^e(e_s) = \frac{Eh}{2}\frac{(1-\nu)\mathrm{tr}({e_s}^2) + \nu(\mathrm{tr}e_s)^2}{(1-\nu)(1+\nu)},$$

and

$$\psi_s^\chi(\chi) = \frac{Eh^3}{24}\frac{(1-\nu)\mathrm{tr}(\chi^2) + \nu(\mathrm{tr}\chi)^2}{(1-\nu)(1+\nu)}.$$

The elastic energy is thus quadratic with respect to the strain field. Moreover, we can see from the last two equations that since h is assumed to be small, the energy deformation due to the curvature is far smaller than the energy due to extension and shear within the tangent plane of the surface. This fact explains why it is easier to bend a thin plate than to extend it.

Taylor & Francis
Taylor & Francis Group
http://taylorandfrancis.com

Models and Algorithms for the Collision of Rigid and Deformable Bodies

Anton Deguet, *INRIA Rhne-Alpes, Montbonnot, France*
Ammar Joukhadar, *INRIA Rhne-Alpes, Montbonnot, France*
Christian Laugier, *INRIA Rhne-Alpes, Montbonnot, France*

In this paper, we describe models and algorithms designed to produce efficient and physically consistent dynamic simulations. These models and algorithms have been implemented in a unique framework, modeling both deformations and contacts through visco-elastic relations. Since the model of interaction "penalty based" is much debated, we present a compared study of both the "penalty" and "impulse" contact models.

Indeed, the "penalty" based model is said to have two major drawbacks : - the determination of the visco-elastic parameters and - the choice of the computation time step. We present a solution for both problems based on physical concepts. Finally, we will present results comparing real data, "impulse" based simulation and "penalty" based simulation.

1 Introduction

At present, almost all planners used in robotics are based on geometric models. Such models cannot represent all aspects of reality but, when combined with basic dynamic control strategies and used in well structured and known environments, they allow one to solve many problems. The phenomena that geometric models are unable to represent - deformations, complex interactions with dynamic and static friction, inertia, collisions, etc. - are nonetheless necessary for the solution of complex problems such as manipulating a rigid or a deformable object using a dextrous hand, guiding an off-road vehicle on a rugged terrain [3] and estimating pose statistics of parts in an assembly line [16].

Since all of the above mentioned phenomena are obviously dynamic, i.e. related to forces encountered in the system being modeled, the most natural approach is to adapt and integrate the Dynamic laws in dynamic simulators. However, models and algorithms for dynamic simulation cannot always be deduced from Dynamic laws. In fact, Dynamic laws are only observations which describe real phenomena with physical values but do not explain what causes them. Thus, the general approach consists in considering a Dynamics law (i.e. an observation) and trying to conceive a model and an algorithm able to produce consistent simulations. The challenge is therefore to integrate different models in a unified framework and find algorithms with acceptable complexity so as to deal with diverse situations in a reasonable computation time.

For robotics applications, one may distinguish five dynamic problems useful for dynamic simulation :

1. Ballistic movements which can be treated using the Newton-Euler law (Isaac Newton, 1642-1727 and Leonhard Euler, 1707-1783) or Lagrangian formalism (Louis de Lagrange, 1736-1813).

2. Constraints due to mechanical articulations which can be represented at a macroscopic level by geometrical constraints.

3. Normal responses for collisions which are difficult to model and for which there are different physical laws based on geometrical constraints, impulsive forces or penalty forces.

4. Friction forces that are efficiently computed by the means of the Coulomb law (Charles de Coulomb, 1736-1806).

5. Deformations, modeled by finite elements or mass-connector networks, which must be combined with the four above phenomena and add difficulties.

In this paper, we present a study of different models for collisions and a system designed to simulate the behavior of deformable and rigid colliding bodies in a unified framework consistent with the previously cited Dynamic laws.

2 Related Work

From its inception, Mechanics has been the main contribution to dynamic simulation [18, 6, 17]. More recently, we can distinguish three principal fields concerned with dynamic simulation in real time. The first one is the dynamic control of kinematic chains and deals mainly with the problem of constraints introduced by articulations [4, 5]. The second one is computer graphics. In this field, complex geometric models introduce so many different control parameters that physically correct behavior is hard to obtain even for an experienced operator. Thus, in order to reduce the number of degrees of freedom, simplified dynamic models are used. The most frequent approach consists in considering collisions and contacts as geometrical constraints, and thereby obtains a uniform treatment of both interactions and articulations [1].

Finally, in robotics, dynamic simulations appear to be useful for planning and teleoperation. For this, the different models and algorithms developed for kinematic chains may be applied, but the representation of contacts by geometrical constraints leads to physically incorrect approximations [14]. Furthermore, the constraints-based model induces a global representation of all constraints by a system of non-linear equations which is difficult to maintain and solve (NP-hard).

In opposition to this global and geometrical representation of contact, we present two models based on physical approaches : "impulse" and "penalty".

Notations In the following sections, we assume that it is possible to establish an orthonormal frame associated with the collision. The z axis is aligned with the direction of contact (determined by the relative velocity of the objects) and called the normal direction. The x and y axis define the tangential plan of the collision. Let x, v and a be the position, velocity and acceleration vectors.

2.1 Impulse based models

Introduction For a neophyte, it is easy to imagine that two rigid objects colliding at high speed will rebound immediately. Thus, the collision itself can be considered as instantaneous, and contact forces as impulsive.

This assumption of "instantaneousity" is the base of the theory of impact, also called stereomechanics in [6]. Since the collision is considered instantaneous, it is not necessary to detail what really happens during it, and the phenomenon must be described by a relation between the state variables before ($t_i = t_c^-$) and after the collision ($t_f = t_c^+$). In addition, positions remain constant and thus the only change concerns velocities.

The simplest relation is given by the Newton impulse model which establishes the relation : $v_z(t_f) = -ev_z(t_i)$ where e is the coefficient of restitution. Since $e \in [0, 1]$, the Newton relation shows that the object cannot rebound faster that it arrived. If $e = 1$ the collision is perfectly elastic and plastic if $e = 0$. This model is far too simple for it is impossible to integrate the Coulomb law for friction because there is no expression of normal forces.

Combining Impulse and Coulomb The Poisson law (Simon Denis Poisson, 1781-1840) establishes that $p_z(t_f) = (e-1)(p_z(t_{mc}))$ where p is the impulse, defined by $p = \int F dt$ or $\Delta m\vec{v}$ (variation of the linear momentum). The main difference with the previous law is that even if the phenomenon is instantaneous, it has to be divided into two phases, compression and restitution, separated by a maximum compression point at t_{mc}.

It is important to note that even if this law requires a partial study of what happens during the collision, it is still not a continuous representation of the whole phenomenon relative to time. It can even be shown that time cannot be used to evaluate $p_z(t_{mc}) = \int_{t_i}^{t_{mc}} F(t)dt$. Because the collision is instantaneous, we should evaluate

$$\lim_{t_{mc} \to t_i} \int_{t_i}^{t_{mc}} F(t)dt$$

and in order to obtain a non null value of p_{mc}, $F(t)$ must take an infinite value which is not realistic [2].

Nevertheless, while an expression of forces as a function of time cannot be used, the Coulomb law requires some expression of the evolution of forces.

Practical implementation A possible solution is to describe the evolution of the collision according to a parameter other than time. According to B. Mirtich, this parameter γ must be continuous and increase monotonically. The following is an abridged presentation of his approach, for further details, see [15].

As $F = m\ddot{x} = m\dot{v}$, $p = \int F(t)dt$ can be rewritten to get $p = \int_1^2 m\dot{v} = m(v_2 - v_1) = m\Delta v$. Introducing γ, we obtain $p(\gamma) = m(v(\gamma) - v(0))$. For simplicity, the matrix M is introduced, which yields

$$Mp(\gamma) = v(\gamma) - v(0) \tag{1}$$

and by differentiation with respect to γ,

$$M\frac{d}{d\gamma}p(\gamma) = \frac{d}{d\gamma}v(\gamma) \tag{2}$$

Equation (2) is justified by the fact that M is constant, nonsingular, symmetric and positive definite.

The problem of choosing γ is still not resolved. B. Mirtich proposes integrating the ODE (2) with respect to p_z. With $\gamma = p_z$, he demonstrates that the vector $\frac{d}{d\gamma}p(\gamma)$ is a function of μ_c, v_x and v_y where μ_c is the coefficient of friction when the tangential velocity is nonzero ($v_x \neq 0 \neq v_y$). This yields a differential equation with respect to p_z,

$$\frac{d}{d\gamma}v(\gamma) = M\begin{pmatrix} -\mu_c v_x/\sqrt{v_x^2 + v_y^2} \\ -\mu_c v_y/\sqrt{v_x^2 + v_y^2} \\ 1 \end{pmatrix} \tag{3}$$

As the parameterization is a projection among the z axis, it is logical to have the third element on the right-hand-side equal to 1.

Collision integration The integration of the equation (3) allows the computation of the evolution of velocity during the collision. When the normal velocity $v_z = 0$, the maximum compression is reached (γ_{mc}). It is then possible to evaluate p_{mc} and, by multiplying it by $(e - 1)$, find the value of total impulse $p(\gamma_f)$. To determine $v(\gamma_f)$, equation (3) is integrated until the total impulse is equal to $p(\gamma_f)$.

In a recent publication [19], Stronge shows that in some situations the Poisson law may introduce an energy gain in the system. To avoid this, he proposes a new definition for e, as a ratio between the work before and after the collision:

$$W_z(t_f) = (1 - e^2)W_z(t_m c) \tag{4}$$

B. Mirtich introduced this criterion in the collision integration. The formulas are more complicated and the parameterization is different. As all details and results cannot be treated in this brief presentation, we encourage the reader to consult [14].

Discussion The main problem solved by this approach is the taking into account of friction forces using the Coulomb law in an instantaneous representation of collision. In a complete simulator, such collisions are considered one after the other because two instantaneous phenomena can be considered as non simultaneous.

Since the Poisson law has been used for many years, the restitution coefficient, e, has already been identified for different materials. However, it has recently been shown that the factor e also depends on the speed of collision and this relation may be approximated by a linear function, $e = 1 - \alpha v_i$ [7].

The foremost problem of Impulse-based collision models appears when collisions between two objects are frequent and ballistic phases are short. When you drop a rigid parallelepiped on the ground, stability should be reached after a few rebounds, but this is not the case for a pure Impulse based simulation. Indeed, a resting contact is modeled through a train of collision and the frequency of impact tends to increase. In this specific case of resting contact, both objects can be considered as static in relation to each other. A local evaluation can be performed to detect this static contact and thereby switch to an adequate model.

For non rigid bodies, contact phases are often long in comparison to ballistic phases. However, the contact cannot be viewed as resting because both objects may still move or bend.

2.2 Penalty based models

Introduction The goal of penalty models is to give a continuous representation of the collision phenomenon at a macroscopic level. It is obvious that a microscopic study should lead to a model of molecular structure, which does not seem feasible in real time. From a macroscopic point of view, the only thing that one can measure is the deformation of colliding bodies.

It is important to underline that the consideration of deformations does not imply that this model is restricted to highly deformable objects, almost all materials are deformed during a collision.

Empirically, this deformation is close to a linear function of forces. Vice versa, the reactive force that first reduces the relative speed of the two objects and finally repulses them from each other, is proportional to the deformation. As this deformation is almost impossible to model, a common hypothesis is that the value of the interpenetration of the two volumes is equivalent to the deformation. Thus, $F = -\lambda x$ where x is a geometrical measure of the interpenetration. This representation by the Hooke law is suitable for perfect elastic collision. A viscous damping term is introduced in order to model plastic phenomenon and $F = -\lambda x - \mu v$. This model is quite similar to that of Kelvin-Voigt used for vibrational mechanics.

Three main problems are raised by this approach. The first one is related to sticking forces due to the viscous term of the force, the second to the representation of time and the third to the identification of λ and μ.

Sticking forces In figure (1), the collision force is plotted as a function of the penetration value. During the ballistic phase P_b, $F = 0$. Then, the force increases until it reaches maximum compression and reduces. (1.a) shows that at the end of the collision, the F is negative because of the damping term. This should not occur and tends to keep the two objects stuck together. K. H. Hunt and F. R. E. Crossley [7] have proposed a new formula to compute F that would respect the shape of (1.b); $F = -\lambda x - \mu x v$. This solution also suppresses the discontinuity of the repulsion force at the beginning of the collision.

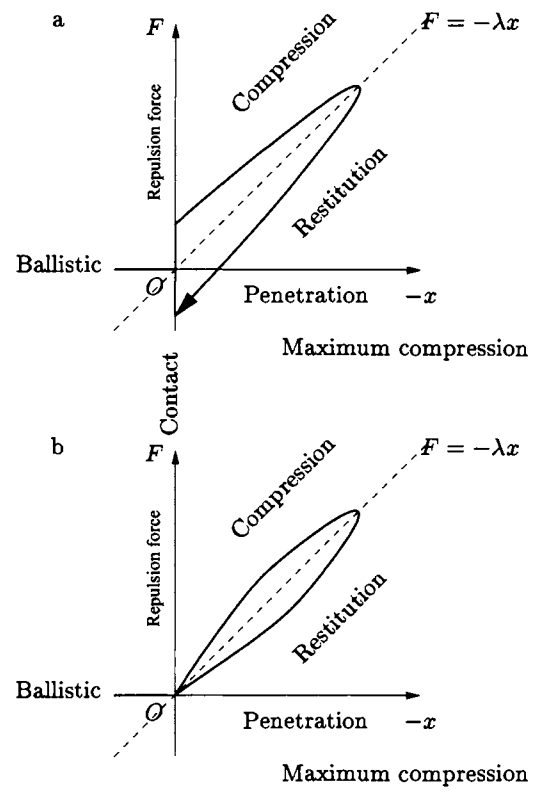


Figure 1: *Regular visco-elastic and non linear visco-elastic model.*

A continuous model In Mechanics, observations are made in a continuous world. Therefore, for implementation, a sampling is almost always introduced for the integration of movements over time. The value of this sampling is hard to determine. If it is too high, the computation cost will be prohibitive and if it is too low, strange phenomena may appear. Let A be a small particle falling on a thin plank (figure 2.a). At the second step (2.b), the algorithm has just detected the interpenetration and is now computing the repulsion force for the next time interval. If the coefficients λ and μ are small, the force will also be small and the speed will not significantly decrease. The particle can then go through the plank (2.c).

Fixing parameters One might suggest that the previous problem could be solved by increasing λ or μ. This would only be a trick and, from a physical point

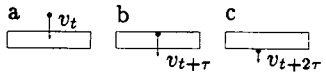


Figure 2: *A soft collision with a sampling that is too low.*

of view, definitively not a correct solution. Indeed, these parameters are supposed to represent the physical properties of the materials. To determine the parameters, a useful result is given by [7] which establishes a relation with the Poisson's coefficient of restitution ($e = 1 - \alpha v_i$, see section 2.1) :

$$\mu = \frac{3}{2}\alpha\lambda \tag{5}$$

Intuitively, the damping term of F causes energy loss and its importance is relative to the value of λ, thus, the relation between e ($\sim \alpha$) and μ/λ is not surprising. Since α can be deduced from e and v_i, which are both known, λ is the only unknown. We present an ad hoc solution for λ in section 5.

Discussion The penalty model, based on a non linear visco-elastic force, is simple to implement and gives an explicit expression of the normal forces of repulsion which leads to an easy integration of friction forces with the Coulomb law. Contrary to what is usually affirmed, there is an automatic and physically logical means of determining the collision parameters λ and μ.

The remaining essential question is that of the time step required to obtain a consistent simulation.

Terminology The "penalty" based approach can also be viewed as an approximation method to insure non penetration between two objects. In this case, the collision is modeled by a geometrical constraint, assuming that rigid objects cannot interpenetrate. The problem is then to solve a dynamic system $D(x) = 0$ under constraints $C(x) \leq 0$ which is NP-hard [1]. The approximate solution consists in minimizing $D(x)+\lambda C(x)$ where $\lambda C(x)$ appears to be a penalty function and λ is the weight of the cost function $C(x)$. Since $C(x)$ is a geometrical constraint, it is a distance or a volume that expresses the interpenetration. Finally, the cost function is $-\lambda x$.

Thus, both approaches lead to the same formalism but what appears as a physical parameter in one case is empirical in the other. The utilization of the model is also different. Indeed, the principle is to find the adequate collision response in one computation step. Therefore, this approach is definitely not continuous. The word "penalty" is employed to describe similar but distinct things, a numeric approximative method and a physical model.

In conclusion, one can assert that the "penalty" numerical method is not optimal for approximating repulsive forces in a dynamic constraint problem but this does not mean that the physical model is not valid.

2.3 Discussion

Both models are based on physical laws and use a local representation of contacts, which is also logical from a physical point of view. The difference lies in the physical variables used for these laws : force versus impulse. For the "penalty based" model, the simulation is continuous through time and provides the normal response as a force, whereas the "impulse based" model instantaneously modifies the speeds of colliding objects. However, both detail what happens during the collision through a collision integration. This is essential since it has been proven that the tangential velocity change during the collision. Thus both models are physically more precise than a contact model based on unilateral constraints (which assumes a constant tangential velocity).

Finally, the impulse approach appears appropriate for simulations in which the duration of contact is brief as compared to the period of the colliding objects' natural frequencies (rigid bodies) whereas the "penalty model" seems better adapted to simulations in which contacts are repeated and relatively long (deformable bodies).

As previously noted, almost all solids can be considered as deformable. Theoretically, when the rigidity of a deformable object tends to the infinite, the behavior of both models should be analogous. We present comparison results in section 5.

3 Modeling Rigid and Deformable Objects

The main goal of our simulator is to integrate both rigid and deformable objects. To simulate deformations, we have retained a visco-elastic model which has been used for many mechanical applications [6, 20], as opposed to the finite element method for simplicity and rapidity. We give a summerized description of a simulator described in [8].

3.1 Discretization

In order to be modeled, an object has to be discretized into particles. The level of this discretization mainly depends on the deformability characteristics of the object. Usually, a very fine and uniform discretization of objects is used for modeling deformable objects. However this straightforward solution generates prohibitive execution times and numerical problems. Non-uniform discretization techniques, like the one used in [12], may also generate inconsistent behavior if the inertia properties of the objects are not preserved.

In order to solve such problems, we have developed a technique that subdivides each object into a set of components with the same inertia properties of the object (inertia matrix and inertia center) and in which each component is not subject to large deformations. The discretization is realized using a wave propagation technique (figure 3) applied from the inertia center of the object to be modeled. The number and size of these components obviously depends on the deformability properties of the object to be modeled (i.e. deformable parts will be more deeply subdivided, and almost rigid parts will be represented by a few components). If the object is rigid, the resulting set of particles will form only one primitive. When it is deformable, each particle is considered as a primitive and appropriate connectors will be added between them.

Three kinds of connectors have been implemented: a linear connector which connects two particles and tends to maintain a resting distance, a torsion connector which connects three particles and tends to maintain an unsigned resting angle, and a joint connector which connects four particles to maintain a signed angle.

For a volumic model, it is necessary to maintain the angles between the edges of the network. A possible solution based on linear connectors consists in connecting all particles to get a set of triangles (like a roof's frame). This classical approach presents a major drawback : torques will not depend on the angle, but on the distances between particles. We have introduced torsion and joint connectors to solve this problem.

The second modeling step consists in giving a volumic dimension to the objects, by adding facets delimited by the external particles of the previous network. These facets are necessary to process collision and contact phenomena.

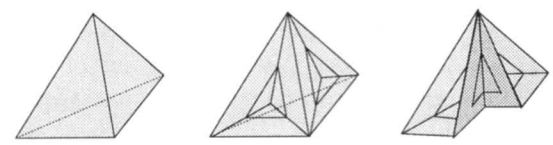


Figure 3: *Discretization of a convex polyhedron.*

3.2 Identification

The parameters λ and μ associated with each connectors and the mass of particles have to be identified in order to obtain physically correct behavior.

For the computation of masses, we have developed an algorithm which ensures that the mass and inertia properties are respected both globally and locally. At a local level, the minimal element of discretization is tetrahedron that will not be significantly deformed during the simulation [9].

For λ and μ, it is possible to find some information about the compression and/or traction properties of certain materials. This can be used to determine the parameters of the linear connectors. Experimental data for torsion and joint connectors is more difficult to obtain since their influences cannot be isolated nor decoupled.

In order to solve this problem, we have introduced an identification process based on a genetic algorithm [9]. This feature has been tested with success on a concrete case for a medical application [11]. In order to detect a

venous thrombosis, the surgeon slides an echographic sensor along the skin of the thigh. If he/she applies too much pressure, the clot can be released. Our simulator is used to model the behavior of a human thigh under pressure (figure 4). The model has also been used to

Figure 4: *Model of a human thigh with an echographic probe.*

simulate the behavior of an animal's Anterior Cruciate Ligament (figure 5). Here as well, results are very satisfying for the margin of error does not exceed 5%. Moreover, surgeons agreed that the simulation was visually realistic [13].

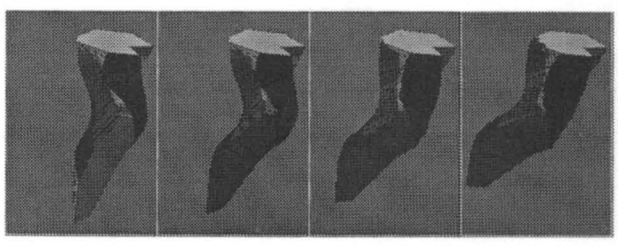


Figure 5: *Simulation of the behavior of the knee ligament.*

3.3 Interactions

For deformable bodies, interactions cannot be considered as instantaneous since they will induce a deformation which has to be analyzed with respect to time. For this reason, a penalty based model seems appropriate. We use the non linear visco-elastic model and the results presented in section 2.2 to determine the parameters of the reaction force.

The interpenetration is defined as the intersecting volume between the two objects. This requires a new efficient algorithm able to deal with negative distances for convex or non convex polyhedra. This algorithm is presented in [10].

Finally, both models of deformation and interaction are based on visco-elastic relations and this approach leads to a logical and coherent framework.

Nevertheless, none of the presented works deals with the problem raised by the utilization of a continuous model in a simulator based on a discrete representation of time. We present a physically based method to solve this problem in section 4.

4 Motion and Deformation Generation

4.1 Introduction

The motions and the deformations of each object are the result of the motion of its primitives. In our case, these primitives are rigid objects or particles which can be considered as a particular case of rigid objects.

Since each primitive is rigid, its motion can be computed using the Newton-Euler differential equations. As the primitives are linked by visco-elastics connectors we obtain following system of differential equations :

$$\vec{F_i} = m_i \vec{a_i} \quad (6)$$

$$\vec{N_i} = I_i \ddot{\vec{\theta}}_i + \dot{\vec{\theta}}_i \wedge (I_i . \dot{\vec{\theta}}_i) \quad (7)$$

$$\vec{F_i} = f(\vec{x_j}, \vec{v_j}, \vec{\theta_j}, \dot{\vec{\theta}}_j) + \vec{F_i^I} + \vec{F_i^C}) | i, j \in [1..n] \quad (8)$$

$$\vec{N_i} = g(\vec{F_i}) \quad (9)$$

where n is the number of primitives, $\vec{x_i}$, $\vec{v_i}$, $\vec{a_i}$, the position, velocity and acceleration of the primitive i, $\vec{\theta_i}$, $\dot{\vec{\theta}}_i$, $\ddot{\vec{\theta}}_i$ the orientation, rotation velocity and rotation acceleration around its inertia center. m_i and I_i are the mass and the inertia matrix of the primitive i. The firsts two equations are those of Newton-Euler. The last equation only converts forces into torques. The third equation is very significant since it represents the forces due to the connectors (relations to primitives j) through the function f. Since we consider linear and torsion/joint connectors, both position and orientation are required to compute f. The third equation also includes all external external phenomena; interaction forces $\vec{F_i^I}$ and control forces $\vec{F_i^C}$.

Unfortunately, this system is *neither continuous nor linear*, because it involves collision forces and control forces that can be applied at any time by the user on the environment. Consequently, it is difficult to solve this system analytically or by an implicit[1] numerical method. A classical means of solving such a system is to use an explicit method based on a second order limited development with a time step τ:

$$\vec{v}_{t+\tau} = \vec{v}_t + \tau\,\vec{a}_t + O(\frac{\tau^2}{2}\dot{\vec{a}}_t) \tag{10}$$

$$\vec{x}_{t+\tau} = \vec{x}_t + \tau\,\vec{v}_t + \frac{\tau^2}{2}\,a_t + O(\frac{\tau^3}{6}\dot{a}_t) \tag{11}$$

$$\vec{\theta_{t+\tau}} = \vec{\theta_t} + I^{-1}(\vec{N_t} - \vec{\theta_t} \wedge (I.\vec{\theta_t}))\tau \tag{12}$$

Some well known drawbacks of such methods for dynamical simulation are as follows:

- The process is very sensitive to the chosen time step. If it is too large, this may lead to numerical divergence generated by the error term (figure 6). In physical terms, this means that the simulation goes too fast and reaches an incoherent position (abnormal compression or traction) because all modifications of forces ($\vec{a}_t$) related to $\vec{F_i^I}$ and $\vec{F_i^C}$) are neglected during τ. On the other hand, if τ is too small, the execution time will be prohibitive. Therefore, it is difficult to choose a time step which ensures numerical stability, because it depends on many factors like collision characteristics or deformability properties. In practice, the time step is usually chosen empirically after several attempts to execute the simulation process and the validity of the simulation is visually determined by an operator.
- The error term cannot be easily estimated, because the system displays a non continuous behavior. This is a major drawback for robotics applications in real time.

4.2 Adaptative time step and error estimation

Problem and approach The equation (10) shows that the numerical error increases when the the time step τ and/or the derivative of the acceleration, $\dot{\vec{a}}$, increase. Since the value of $\dot{\vec{a}}$ cannot be controlled, the only way to decrease the error term is to decrease the time step, and consequently to increase the computation time. It is judicious to remark that when the external forces are null, the error will be negligible and a large sampling is not required. Thus, we used an adaptive time step τ (figure 7), where the sampling rate varies according to the variation of $|\dot{a}|$ in order to keep the error term less than a given threshold while using a few samples as possible.

[1] If $x_{t+\tau} = f(x_\gamma) \mid \gamma \leq t$ the approach is explicit (incremental), else it is implicit.

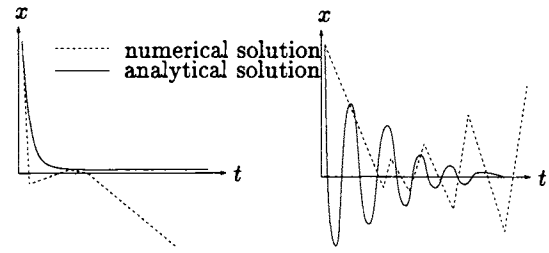


Figure 6: *Without a suitable time step the numerical solution may diverge.*

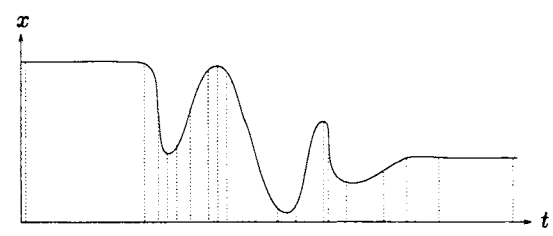


Figure 7: *A good strategy to avoid divergence and reduce the computational time is to use an adaptive time step τ which allows the system to be integrated using a minimum number of samples.*

The main problem comes from the fact that the value of $\dot{a}$ is not easy to determine since we have many simultaneous collisions and we assume that a user can add discontinuous forces through a virtual controller or a force feedback device.

In physical terms, the error can also be considered as a jump in time which induces an improbable state.

Therefore, a Physical law must have been transgressed and this law could be used to determine the amplitude of the error. As we saw before, a too large time step can produce abnormal forces which, after an integration step, will cause a variation of mechanical energy. This variation is a transgression of the main property of the mechanical energy (which remains constant for an isolated system) and seems to be a good criterion to detect the error.

Let E_m be the mechanical energy of the physical system S considered ($E_m = E_p + E_k$, where E_p is the potential energy and E_k is the kinetic energy). E_m depends on $\vec{x}$ and $\vec{v}$:

$$E_m = E_p + E_k = -\int_x^{x+\Delta x} \vec{F}.\delta\vec{x} + \frac{1}{2}m\Delta v^2$$

Three cases must be considered when reasoning about the variation of E_m:

- S in an isolated system without energy dissipation. This situation arises when an object (or a set of objects) is not subjected to an external force, nor friction phenomenon. In this case, E_m is constant.
- S is an isolated system with energy dissipation. This situation arises when a friction force appears between objects. In this case E_m decreases.
- S is a non-isolated system. This situation arises when S is subjected to an external forces (control force or collision). In this case E_m increases.

Time step determination The time step determination algorithm is basically applied during the time intervals in which the object is considered as an isolated system. The extension of the domain of validity of this algorithm is discussed below. As previously explained, the mechanical energy E_m is **constant** when the system is isolated : $\Delta E_m = 0$. If ΔE_m increases during the numerical integration, this means that this "non physical" variation is the result of the error term. At this point, the idea is to monitor the value of $|\Delta E_m|$ at each time step and to choose the greatest time step which satisfies $|\Delta E_m| < \epsilon_e$ for a small ϵ_e (tolerated error). The following algorithm gives a practical implementation of this idea:

Let $\vec{x_t}$ be the position of the object.
Let $\vec{v_t}$ be its velocity.
Let $\vec{F_t}$ be the sum of the forces applied on it.
LOOP
  Calculate $\vec{x_{t+\tau}}$ and $\vec{v_{t+\tau}}$
  Calculate ΔE.
  IF $|\Delta E| > \epsilon_e$ THEN
    LOOP
      $\tau = \frac{\tau}{2}$
      Calculate $\vec{x_{t+\tau}}$ and $\vec{v_{t+\tau}}$
      Calculate ΔE
    UNTIL $|\Delta E| < \epsilon_e$
  ELSE
    $\tau = \frac{3}{2}\tau$
UNTIL ∞

Justification $|\Delta E_m| > \epsilon_e$ means that the variation of energy is too large, and consequently the time step is too large; $|\Delta E_m| < \epsilon_e$ means that the probability that the state of the system is "correct" is very high. An intuitive justification of this assumption relies on the fact that two different states of the system having approximatively the same numerical value for E_m cannot be associated with two consecutive (and close) time steps, because of the continuity of object motion. For instance, when two objects penetrate each other, the mechanical energy increases continuously until they separate. Thus the two objects would have to cover in a single time step a distance which would be greater than the sum of their dimensions in order to reach an incorrect state which verifies energy conservation.

Dealing with external perturbations When an external force F_e (control or collision force) is applied to the system, E_m is no longer constant during a very short interval of time. After this event, we can take into account the potential energy generated by F_e and the system will return to a state where E_m is constant. A safe and blind iteration (using a very small time step ϵ_τ) of the algorithm is necessary to evaluate the value of the new mechanical energy E_m. A good way to choose ϵ_τ, which is consistent with the algorithm described above, is to search for the time step ϵ_τ for which ΔE_m between t and $t + \tau$ is smaller than ϵ. It

is always possible to find such a time step because of the **continuity** of E_m. Consequently, the previous algorithm still applies where an external force is applied to the system.

Discussion The two main difficulties raised by the numerical integration of the dynamic equations were : firstly, how to choose a time step which would lead to a physically correct simulation and, secondly, what kind of compromise between computation time and physical consistency would be acceptable.

The solution presented is not a compromise because the sampling frequency is determined in order to maintain physical accuracy. Nevertheless, since the time step is adaptative, a very small time step adapted to a given phenomenon will not penalize the whole simulation and the global computation time remains acceptable.

More importantly, monitoring the time step and therefore the error through the value of the mechanical energy solves a crucial problem. Indeed, even with a constant time step, as small it might be, it is almost impossible to ensure that the sampling frequency is high enough. This practical approach leads to a high probability of avoiding the numerical divergence for both deformable and contact penalty models.

5 Experiments

5.1 Protocol

In order to test our model of contact and the algorithms developed to compute the movements, we did some experiments similar to those realized in [16]. This work has the advantage of giving numerical results.

The experiment consists in estimating the pose statistics of different small rigid parts falling on part feeders. This information is then used to optimize the motions of a robot arm which picks up the parts and places them correctly (further details in [14]).

We tested our approach with two of the four presented parts; the orange insulator cap and the large stereo button (figure 8). Exact CAD models as well as the physical parameters have been furnished by B. Mirtich. Both restitution and friction coefficients were estimated equal to 0.3. At the moment of impact,

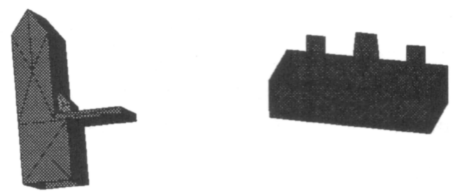


Figure 8: *The insulator cap and the stereo button.*

the simulator computes $\alpha = \frac{1-e}{v_i}$. Since $\alpha = \frac{2\mu}{3\lambda}$ the remaining unknown is λ. We believe that λ can be experimentally deduced, but since the parts were not available, we fixed λ by another means. Indeed, this parameter represents the rigidity of the collision and tends to enforce a non penetration constraint. Thus, we experimentally determined λ so that the interpenetration is very small compared to the size of the object (less than 0.5% of the length). This leads to a computation at $320Khz$ at the worse moment of the collision but the average time step is close to $0.00072s$, i.e. $1.4Khz$.

In most cases, we determine λ through the deformation of the material under static load. This solution is very sensitive to the shape of the cells and can only be applied with face to face contact. However, for deformable bodies, surfaces are discretized and a high percentage of contact surfaces is represented by cells in face to face contact (all central faces).

The two parts are non convex and therefore modeled as an assembly of rigid convex polyhedra (figure 8). For interactions, we consider the sum of the forces due to the volumes of each sub-polyhedron. This sum is then applied to the whole object (figure 9).

5.2 Results

The results are presented in tables 1 and 2. Let $p_1, \ldots, p_n$ represent the probability of each of n states, as measured in the physical test. Let $a_i, \ldots, a_n$ represent the corresponding probabilities computed by a

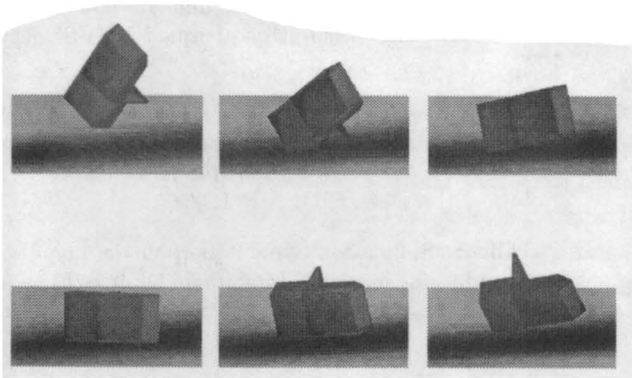


Figure 9: *The insulator cap falls and rebounds.*

given simulator. The error percentage for this simulator is equal to :

$$e = 100 \sum_{i=1}^{n} p_i |a_i - p_i|$$

Pose	Impulse	Penalty	Physical tests
1	41.9	43.0	46.0
2	26.2	29.5	27.1
3	28.3	20.1	19.7
4	3.0	4.0	5.0
5	0.8	2.7	2.2
6	0.0	0.6	0.0
error	4.0	2.2	-

Table 1: *Orange insulator cap statistics.*

Pose	Impulse	Penalty	Physical tests
1	71.7	70.4	75.8
2	20.9	20.5	13.8
3	7.4	8.9	10.5
4	0.1	0.25	0.0
error	4.4	5.2	-

Table 2: *Stereo button statistics.*

5.3 Discussion

These results have been obtained without any particular tuning. They are quite close to the impulse based simulations and to reality verifies the accuracy of our approach. This is very encouraging since the penalty based model is not particularly well suited to stiff collisions.

Our conclusion is not that the penalty based model is better than the impulse based one for every problem. Indeed, to simulate 995 drops for the first part and 776 for the second part, we used 3 workstations (*SiliconGraphics Indy R*4600 100*Mhz*) over a couple of hours whereas the impulse based simulation required 45 minutes for 2000 drops (*SiliconGraphics Indigo*$_2$ *R*4400 200*Mhz*). As previously mentionned, we did not modify the simulator for those experiments and we believe that the execution time could be seriously reduced with some simple optimizations (it was originaly developped for deformable objects).

6 Conclusions

We have presented a simulator which is able to model the movements of both rigid and deformable objects. This simulator is based on a mass-spring model for the deformation and the penalty model for the interactions. This representation is unified because both use visco-elastic relations. Nevertheless, it is usually said that this approach is physically incorrect. In order to correct this, we introduced :

- A relation which allows the determination of the collision parameters as function of the coefficient of restitution e and the elasticity of the materials.
- A practical criterion based on the mechanical energy of the system which allows the detection of numerical divergence due to wrong time step.

We implemented and tested these methods in order to validate the simulator. We reproduced previously released experiments and obtained simular results. This tends to prove the physical accuracy of our system.

In the future, we plan to study in detail (velocity, acceleration, momentum, energy, work) the two different models during the collision with analytical and numerical integration.

The ultimate goal is to combine different algorithms corresponding to different phenomena. In [14], two models based on geometric constraints and impulses have been integrated in order to simulate articulations and impacts. We believe that it would be interesting to integrate impulse and penalty in a unique simulator to model rigid and deformable interactions.

Acknowledgments

This work has been partially supported by the France Berkeley Fund.

We would like to thank John Canny for having invited Anton Deguet to Berkeley during the summer of 1996, and Brian Mirtich for his help and the data he sent us.

References

[1] D. Baraff. *Dynamic Simulation of Non-Penetrating Rigid bodies.* PhD thesis, Cornell University, March 1992.

[2] B. Brogliato. *Nonsmouth Impact Mechanics.* Springer-Verlag, London, 1996.

[3] M. Cherif and C. Laugier. Motion planning of autonomous off-road vehicles under physical interaction constraints. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, Nagoya (JP), May 1995.

[4] J. J. Craig. *Introduction to Robotics, Mechanics and Control.* Addison-Wesley, second edition, 1989.

[5] R. Featherstone. *Robot Dynamics Algorithms.* Kluwer Academic Publishers, 1987.

[6] W. Goldsmith. *Impact: The Theory and Physical Behaviour of Colliding Solids.* Edward Alrnold, London, 1960.

[7] K. H. Hunt and F. R. E. Crossley. Coefficient of restitution interpreted as damping in vibroimpact. *Journal of Applied Mechanics*, pages 440–445, June 1975.

[8] A. Joukhadar. *Simulation Dynamique et Applications Robotiques.* Thse de doctorat, Inst. Nat. Polytechnique de Grenoble, Grenoble (FR), June 1997.

[9] A. Joukhadar, F. Garat, and C. Laugier. Parameter identification for dynamic simulation. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, volume 3, pages 1928–1933, Albuquerque NM (US), April 1997.

[10] A. Joukhadar and C. Laugier. Dynamic simulation: Model, basic algorithms, and optimization. In *Proc. of the Workshop on the Algorithmic Foundations of Robotics*, Toulouse (FR), July 1996.

[11] P. Laffont. Simulation dynamique pour le diagnostic de thromboses veineuses. Mmoire de Diplme d'Etudes Approfondies, Universit de Savoie, Chambry (FR), 1997.

[12] A. Luciani and al. An unified view of multiple behaviour, flexibility, plasticity and fractures: balls, bubbles and agglomerates. In *IFIP WG 5.10 on Modeling in Computer Graphics*, pages 55–74. Springer-Verlag, 1991.

[13] S. Martelli, A. Joukhadar, S. Zaffagnini, M. Marcacci, S. Lavallee, and G. Champleboux. A fiber-based ACL model for geometrical and mechanical simulations. Research Report 2924, Inst. Nat. de Recherche en Informatique et en Automatique, Montbonnot Saint Martin (FR), July 1996.

[14] B. V. Mirtich. *Impulsed-based Dynamic Simulation of Rigid Body Systems.* PhD thesis, University of California at Berkeley, 1996.

[15] B. V. Mirtich and J. F. Canny. Impulse-based dynamic simulation. In *Proc. of the Workshop on the Algorithmic Foundations of Robotics*, San Francisco CA (US), February 1994.

[16] B. V. Mirtich, Y. Zhuang, K. Goldberg, J. Craig, R. Zanutta, B. Carlisle, and J. F. Canny. Estimating pose statistics for robotics part feeders. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, volume 2, pages 1140–1146, Minneapolis MN (US), April 1996.

[17] L. A. Pars. *A Treatise on Analytical Dynamics.* Heinemann, London, 1965.

[18] E. J. Routh. *Dynamics of a System of Rigid Bodies.* Dover, sixth edition, 1905.

[19] W. J. Stronge. Unraveling paradoxical theories for rigid body collisions. *Journal of Applied Mechanics*, 58:1049–1055, December 1991.

[20] S. Timoshenko and J. N. Goodier. *Theory of Elasticity.* McGraw-Hill, second edition, 1951.

Numerical Computation of Optimal Navigation Functions on a Simplicial Complex

Steven M. LaValle, *Iowa State University, Ames, IA, USA*

This paper presents a general approach to computing optimal feedback motion strategies for a holonomic or nonholonomic robot in a static workspace. The proposed algorithm synthesizes a numerical navigation function defined by interpolation in a simplicial complex. The computation progresses much in the same way as Dijkstra's algorithm for graphs; however, the proposed approach applies to continuous spaces with geometric and nonholonomic constraints. By choosing a simplicial complex representation instead of a straightforward grid, the number of interpolation operations (which are required for continuous-state, numerical dynamic programming) is reduced from 2^n to $n+1$, in which n is the dimension of the configuration space. Some preliminary findings are discussed for an implementation of the algorithm for the case of a two-dimensional configuration space.

1 Introduction

Many applications in robotics require automatically computing a motion strategy that brings a complex geometric body from an initial configuration to a goal configuration while satisfying geometric constraints. Basic motion planning (or path planning) approaches typically do not apply in cases that are further complicated by imprecise localization and nonholonomic constraints. For example, in mobile robot or vehicle navigation, both the current and future configurations can only be estimated during execution using sensing and predictive models. A pre-planned path might have to be repeatedly adjusted due to such uncertainties; however, a preferable alternative would be to immediately know the optimal motion command given the current configuration (assuming the geometry and goal configuration are fixed). This corresponds to a feedback motion strategy, which is computed using the method presented in this paper. Arguments for directly incorporating feedback in the form of a navigation function were also made in [26]. Furthermore, applications continue to appear that involve the challenge of incorporating nonholonomic constraints on the system velocities (e.g., wheeled-robot systems [6, 17, 29], push planning [1, 23]).

The focus of this paper is on motion planning problems that require either feedback solutions, nonholonomic analysis, or both. Emphasis is placed on developing a technique that can be applied to a broad set of problems, as opposed to requiring particular simplifications that can be made for a particular problem. Although useful and interesting progress is often made by exploiting characteristics of a particular problem, a more general approach is often more likely to find immediate application, even if the approach is inferior for particular problems that are better solved by a specialized approach. In addition, one is often willing to exchange strong guarantees on performance (such as completeness) if the result is to obtain a method that is more broadly applicable. This philosophy shift was once seen in path planning research. Approaches such as the randomized potential field planner [2] or the randomized roadmap planner [11], offer weaker statements regarding the performance (i.e., probabilistic completeness as opposed to completeness); however, their general applicability and practical efficiency has stimulated their use in a variety of applications.

In this paper, a feedback strategy is computed numerically, which implies that the solution quality generally depends on resolution and quantization errors. This implies resolution completeness, and in the limit-

ing case, the numerical solution converges to an optimal solution. This computational approach is also intended as a step toward a more general approach that can additionally handle complications such as stochastic or nondeterministic uncertainty in sensing and predictability, multiple robots, and dynamics. Such generalization can be made using the unified mathematical framework presented in [19].

Relationship to Dijkstra's algorithm The computational approach offered in this paper is built from the principle of optimality as it appears in optimal control theory [13], and is closely related to the level set method in [28]. It also shares similarities with Dijkstra's algorithm for computing single-source shortest paths in a graph; however, in our case a continuous configuration space must be considered (as opposed to a graph). It might seem possible to construct a graph by quantizing the configuration space to obtain vertices and using standard four-neighbors to obtain edges. However, the application of Dijkstra's algorithm (or wavefront propagation) to the resulting graph would not necessarily yield solutions that obey nonholonomic constraints. Furthermore, even results for holonomic problems will generally not be optimal because lengths are measured by forcing motions to occur along the grid (i.e., Manhattan distance is obtained instead of Euclidean distance). The approach proposed in this paper can be viewed as a way to make Dijkstra's algorithm yield the right results for nonholonomic motion planning problems in a continuous configuration space with geometric constraints. Instead of a graph or a grid, a simplicial complex is constructed; this is used to represent a continuous *cost-to-go function* (or navigation function) which serves the same purpose as the costs that are placed on vertices in Dijkstra's algorithm. Another related approach is the "continuous Dijkstra paradigm" [9, 25], which is used to efficiently compute holonomic shortest paths in a 2D polygonal environment by identifying critical events that correspond to a propagating wavefront of level-set curves that measure distance from a goal position.

Relationship to Barraquand-Latombe approach The proposed approach shares similarities with the nonholonomic planning approach used in [3, 24], which involves a branch-and-bound exploration from an initial configuration, using a discretized approximation of time and the set of possible control inputs. Each exploration step is achieved by applying a control input and integrating the system equation over a small time to obtain a new configuration. Within a given neighborhood, only a single configuration is saved. Eventually, the search expands until a configuration near the goal is reached. In our approach, a numerical navigation function is constructed by incremental expansion from a goal region as opposed to the initial configuration. Instead of keeping track of reachable configurations, our samples of the configuration space represent interpolation control points from which approximate values of the navigation function can be computed. Instead of determining a path, the navigation function is used as a representation of a *configuration-feedback* motion strategy that provides the optimal control input from anywhere in the configuration space. This becomes particularly valuable when there is a high degree of unpredictability in future configurations.

2 Problem Formulation

Basic concepts A workspace is defined that completely and precisely characterizes the environment of a robot. An n-dimensional *configuration space* $\mathcal{C}$ captures position, orientation, and/or other information for robot. Let $x \in \mathcal{C}$ denote a configuration. Let $\mathcal{C}_{free}$ denote the collision-free subset of the configuration space. Let $G \subset \mathcal{C}_{free}$ represent the *goal region*. Let u be an m-dimensional vector of control inputs. A control input could, for example, specify a particular steering angle for a car-like robot. Let $U \subseteq \Re^m$ represent the space of possible control inputs. The task is to determine control inputs that depend on a current configuration, "drive" the robot into the goal region, and optimize a specified criterion.

It will be helpful to define time in the upcoming concepts. Suppose that there is some initial time, $t = 0$, at which the robot is at x_{init}. Suppose also that there is some final time, T_f (one would like to at least have the robot at the goal before $t = T_f$). It will be assumed

that the system is time-invariant (i.e., no decisions will depend on the particular time).

Incorporating nonholonomic constraints Let $\dot{x}$ represent the velocity of the robot in the configuration space. The nonholonomic constraints can be represented using control-theoretic notation as $\dot{x} = f(x(t), u(t))$ [10]. The case of $\dot{x}(t) = u(t)$ corresponds to a holonomic planning problem. Suppose that any control input is allowed if it satisfies $\|u\| = 1$ or $u = 0$. This implies that one can move the robot in any allowable direction at a local tangent space. A solution to a holonomic planning problem can be represented as a smooth path $h : [0, T_f] \rightarrow \mathcal{C}_{free}$ such that $x(0) = x_{init}$ and $x(T_f) \in G$. By setting the control input as $u(t) = \frac{dh}{dt}$, the original path can be obtained.

For nonholonomic problems, one will only be allowed to move the robot through a function of the form $\dot{x} = f(x(t), u(t))$. For example, as described in [15], p. 432, the equations for the nonholonomic car robot can be expressed as $\dot{x} = v\cos(\theta)$, $\dot{y} = v\sin(\theta)$, and $\dot{\theta} = \frac{v}{L}\tan(\phi)$, in which v is the speed of the rear axle midpoint, L is the distance between the front and rear axles, and ϕ is the steering angle. Using the notation in this paper, $(\dot{x}, \dot{y}, \dot{\theta})$ becomes $\dot{x}$, and (v, ϕ) becomes u. Other nonholonomic models, such as those used for push planning in [23] can be encoded in this way. The function f can be considered as a kind of interface between the user and the robot. Commands are specified through $u(t)$, but the resulting velocities in the configuration space get transformed using f (which can generally prevent the user from directly controlling velocities).

Incorporating feedback As stated in Section 1 the exact path executed by the robot often cannot be predicted in practice. Instead of using a predefined control input, $u(t)$, for all $t \in [0, T_f]$, it becomes necessary to allow the control input to depend on the particular configuration of the robot during execution. In an application, such as mobile robot navigation, a sensing module might be used to provide repeated estimates of the current configuration. The control input should be chosen in light of this estimated configuration. This motivates the definition of a *feedback motion strategy* as a function, $\gamma : \mathcal{C}_{free} \rightarrow U$. Thus, at a configuration $x \in \mathcal{C}_{free}$, $u = \gamma(x)$ specifies a particular control input. Note that the computational task is to compute a representation of a *function of configuration*, as opposed to a *path*, which is standard in motion planning.

Because analytical solutions are avoided, one is obligated at some point to approximately represent γ. This task requires special attention. Suppose that we would like to approximate γ using a discretized set of points and linear interpolation (which are used, for example, in the trapezoid rule for numerical integration). This becomes problematic because interpolations in U can be meaningless. For example, suppose $u \in U$ denotes "go forward" for a car-like robot, and $u' \in U$ denotes "go backwards." If $\gamma(x) = u$ and $\gamma(x') = u'$ for two close configurations (i.e., $\|x - x'\|$ is less than some small $\epsilon > 0$), then what would be the appropriate input for a configuration that lies between x and x'? It would certainly be senseless to compute a control input that denotes "stop" by interpolation. This would cause the car-like robot to be permanently fixed at the same configuration.

It will be seen in Section 3 that the approximation difficulty can be overcome by the use of a real-valued navigation function. The navigation function will be ultimately approximated by determining its values at vertices in a simplicial complex and using linear interpolation. The appropriate control input can be recovered during execution by performing some fast, local computations using the navigation function.

Incorporating optimality A *loss functional* is defined that evaluates any configuration trajectory and control function:

$$L = \int_0^{T_f} l(x(t), u(t))dt + Q(x(T_f)). \qquad (1)$$

The term $u(t)$ yields the control input applied at time t. With a given feedback motion strategy, γ, this would be chosen as $u(t) = \gamma(x(t))$. The integrand $l(x(t), u(t))$ represents an instantaneous cost, which when integrated can be imagined as the total amount of energy that is expended. The term $Q(x(T_f))$ is a final cost that can be used to induce a preference over trajecto-

ries that terminate in a goal region of the configuration space.

As an example, the following measures the path length for a trajectory that leads to the goal:

$$L = \begin{cases} \int_0^{T_f} \|\dot{x}(t)\| dt & \text{if } x(T_f) \in G \\ \infty & \text{otherwise} \end{cases} . \quad (2)$$

The term $\int_0^{T_f} \|\dot{x}(t)\| dt$ measures path length.

Discrete-time approximation Since numerical computations will be performed, the continuous-time formulation will be approximated in discrete time. With the discretization of time, $[0, T_f]$ is partitioned into *stages*, denoted by $k \in \{1, \ldots, K+1\}$. Stage k refers to time $(k-1)\Delta t$. The final stage is given by $K = \lfloor T_f / \Delta t \rfloor$. Let x_k represent the configuration at stage k. At each stage k, an *action* u_k can be chosen from an *action space* U. Because

$$\frac{dx}{dt} = \lim_{\Delta t \to 0} \frac{x(t+\Delta t) - x(t)}{\Delta t}, \quad (3)$$

the equation $\dot{x} = u$ can be approximated as

$$x_{k+1} = x_k + \Delta t \, u_k, \quad (4)$$

in which $x_k = x(t)$, $x_{k+1} = x(t+\Delta t)$, and $u_k = u(t)$. The equation $\dot{x} = f(x(t), u(t))$ can be approximated by a transition equation of the form $x_{k+1} = f(x_k, u_k)$.

A discrete-time representation of the loss functional can also be defined:

$$L(x_1, \ldots, x_{K+1}, u_1, \ldots, u_K) = \sum_{k=1}^{K} l_k(x_k, u_k) + l_{K+1}(x_{K+1}), \quad (5)$$

in which l_k and l_{K+1} serve the same purpose as l and Q in the continuous-time loss functional.

A path-planning problem that does not consider optimality can be represented in discrete time by letting $l_k = 0$ for all $k \in \{1, \ldots, K\}$, and defining the final term as $l_{K+1}(x_{K+1}) = 0$ if $x_k \in G$, and $l_{K+1}(x_{K+1}) = 1$ otherwise. This gives equal preference to all trajectories that reach the goal. To approximate the problem of planning an optimal-length path, $l_k = 1$ for each $k \in \{1, \ldots, K\}$ such that $x_k \notin G$. The final term is then defined as $l_{K+1}(x_{K+1}) = 0$ if $x_{K+1} \in G$, and $l_{K+1}(x_{K+1}) = \infty$ otherwise.

3 Optimal Navigation Functions

Usually in motion planning the solution takes the form of a path; however, in our case the solution involves configuration feedback. One possibility is that the algorithm can return a function that maps configurations into actions. Thus from any location in the configuration space, the robot will have the appropriate action during the execution of the optimal strategy.

This section presents the concept of *cost-to-go* functions, which are used as a navigation function in the sense proposed in [26]. Instead of building a direct representation of the feedback strategy, the proposed approach uses a cost-to-go function as an intermediate representation from which the appropriate action can quickly be obtained. It will be seen that this approach is advantageous for the numerical computations because optimal actions can be selected by performing interpolations on the cost-to-go function. It is generally inappropriate to "interpolate" on the space of possible actions.

The principle of optimality Initially, a cost-to-go function will be defined for each stage k. A cost-to-go function will eventually be obtained that is stage independent. The cost-to-go function $L_k^* : \mathcal{C}_{free} \to \Re \cup \{\infty\}$ represents the loss that will accumulate if the optimal trajectory is executed from stage k until stage $K+1$, starting at configuration $x_k \in \mathcal{C}_{free}$. Note that $L_{K+1}^* = l_{K+1}$ from (5). If K is sufficiently large, then for reasonably-behaved planning problems there exists an $i < K$ such that $L_k^* = L_i^*$ for all $i < k$. This will hold for problems in which: 1) all optimal trajectories that reach the goal arrive in a finite amount of time; 2) infinite loss is obtained for a trajectory that fails to reach the goal; 3) no loss accumulates while the robot "waits" in the goal region; 4) the environment and motion models are stationary.

Bellman's principle of optimality provides a powerful constraint on the solution structure [4], which directly

leads to a numerical computation approach. With a discrete-time model, a difference equation is obtained that relates successive cost-to-go functions. This difference equation will now be derived. The *cost-to-go* function at stage k is formally defined as

$$L_k^*(x_k) = \min_{u_k,\ldots,u_K} \left\{ \sum_{i=k}^{K} l_i(x_i, u_i) + l_{K+1}(x_{K+1}) \right\}. \tag{6}$$

The term $\gamma_i^*(x_i)$ represents u_i, the action chosen at stage i from configuration x_i. Equation (6) represents the loss that will be received under the execution of the optimal strategy, γ^*, from stage k to stage $K+1$.

The cost-to-go can be separated:

$$L_k^*(x_k) = \min_{u_k} \min_{u_{k+1},\ldots,u_K} \left\{ l_k(x_k, u_k) + \sum_{i=k+1}^{K} l_i(x_i, u_i) + l_{K+1}(x_{K+1}) \right\}. \tag{7}$$

The second *min* does not affect the l_k term; thus, it can be removed to obtain

$$L_k^*(x_k) = \min_{u_k} \left[l_k(x_k, u_k) + \min_{u_{k+1},\ldots,u_K} \left\{ \sum_{i=k+1}^{K} l_i(x_i, u_i) + l_{K+1}(x_{K+1}) \right\} \right] \tag{8}$$

The second portion of the *min* represents the cost-to-go function for stage $k+1$, yielding:

$$L_k^*(x_k) = \min_{u_k} \left\{ l_k(x_k, u_k) + L_{k+1}^*(x_{k+1}) \right\}. \tag{9}$$

This final form represents a powerful constraint on the optimal strategy. The optimal action at stage k and configuration x depends only on the cost-to-go values at stage $k+1$. Furthermore, only the particular cost-to-go values that are reachable from the transition equation $x_{k+1} = f(x_k, u_k)$ need to be considered. The dependencies are local; yet, the globally-optimal strategy is characterized. This property will be exploited by the methods presented in Section 4.

Using the cost-to-go as a navigation function As stated previously, if K is sufficiently large, there exists an i such that for all $k \leq i$ and all $x \in \mathcal{C}_{free}$, $L_k^*(x_k) = L_i^*(x_i)$. In this case, L^* will be used instead of L_i^* because the cost-to-go is stage-invariant.

Suppose that L^* has been numerically computed, and that values of $L^*(x)$ for any $x \in \mathcal{C}_{free}$ can be obtained through a quantization and interpolation scheme (which will be discussed in Section 4). To execute the optimal strategy, an appropriate action must be chosen using the cost-to-go representation from any given configuration. One approach would be to simply store the action that produced the optimal cost-to-go value for each quantized configuration. The appropriate action could then be selected by recalling the stored action at the nearest quantized configuration. This method could cause errors, particularly since it does not utilize any benefits of interpolation. A preferred alternative is to select actions by locally evaluating (9) at the current configuration. Interpolation schemes can also be used in this step. Note that although the approach to select the action is local (and efficient), the global information is still taken into account (it is encoded in the cost-to-go function). Once the optimal action is determined, the next configuration is obtained (i.e., not a quantized configuration) using $x_{k+1} = f(u_k, x_k)$. This form of iteration continues until the goal is reached or a termination condition is met. Note that this scheme does not force the robot to traverse quantized configurations.

During the time between stages, the trajectory can be linearly interpolated between the endpoints given by the discrete-time transition equation, or can be integrated using the original continuous-time transition equation. Once the optimal action is determined, an exact next configuration is obtained (i.e., not a quantized configuration). This form of iteration continues until the goal is reached or a termination condition is met.

4 Computational Approach

This section presents an algorithm for computing the optimal navigation function L^* for a given problem. The optimal motions are executed by selecting actions using L^* as a navigation function, as discussed in Section 3. Two basic methods are described. The first method is derived from numerical dynamic programming techniques presented in optimal control literature

[13, 14]. The method described here and several extensions were implemented and tested on many problems in [18, 20, 21]. The purpose of describing it here is to provide a basis of comparison for the new method, for which a description soon follows. The new method uses many of the same assumptions as the existing method, but computes the result with much better computational performance.

4.1 Previously established approach

For the purpose of discussion, it will be assumed that $\mathcal{C}_{free}$ is a subset of Euclidean space. Topological issues that arise on manifolds such as $\Re^2 \times S^1$ (the case of translation and rotation in the plane) will not be treated here; however, only minor notational variations are required to include such cases.

An optimal strategy can be computed by successively building approximate representations of the cost-to-go functions [18]. The most straightforward way to represent a cost-to-go function is to specify its values at each location in a discretized grid. The first step is to construct a representation of L^*_{K+1}. The final term, $l_{K+1}(x_{K+1})$, of the loss functional is directly used to assign values of $L^*_{K+1}(x_{K+1})$ at discretized locations. Typically, $l_{K+1}(x_{K+1}) = 0$ if x_{K+1} lies in the goal region, and $l_{K+1}(x_{K+1}) = \infty$ otherwise. This only permits trajectories that terminate in the goal region. If the goal is a point, it might be necessary to expand the goal into a region that includes some of the quantized configurations.

The dynamic programming equation (9) is used to compute the next cost-to-go function, L^*_K, and subsequent cost-to-go functions. For each quantized configuration, x_k, a quantized set of actions, $u_k \in U$, are evaluated. For a given action u_k, the next configuration obtained by $x_{k+1} = f(x_k, u_k)$ generally might not lie on a quantized configuration. However, linear interpolation between neighboring quantized configurations can be used to obtain the appropriate loss value without restricting the motions to the grid. Suppose for example, that for a one-dimensional configuration space, $L^*_{k+1}[i]$ and $L^*_{k+1}[i+1]$ represent the loss values for some configurations x_i and x_{i+1}. Suppose that the transition equation, f, yields some x that is between x_i and x_{i+1}. Let

$$\alpha = \frac{x_{i+1} - x}{x_{i+1} - x_i}. \tag{10}$$

Note that $\alpha = 1$ when $x = x_i$ and $\alpha = 0$ when $x = x_{i+1}$. The interpolated loss can be expressed as

$$L^*_{k+1}(x_{k+1}) \approx \alpha L^*_{k+1}[i] + (1-\alpha) L^*_{k+1}[i+1]. \tag{11}$$

In an n-dimensional configuration 1space, interpolation can be performed between 2^n neighbors. For example, if $\mathcal{C} = \Re^2$, the interpolation can be computed as

$$L^*_{k+1}(x_{k+1}) \approx \alpha\beta L^*_{k+1}[i,j] + (1-\alpha)\beta L^*_{k+1}[i+1,j] + \alpha(1-\beta) L^*_{k+1}[i,j+1] + (1-\alpha)(1-\beta) L^*_{k+1}[i+1,j+1] \tag{12}$$

in which $\alpha, \beta \in [0,1]$ are coefficients that express the normalized distance to the neighbors. Convergence properties of the quantization and interpolation are discussed in [4, 5]. Interpolation represents an important step that overcomes the problems of measuring Manhattan distance due to quantization.

The obstacle constraints must also be taken into account. The constraints can be directly evaluated each time to determine whether each x_{k+1} lies in the free space, or a bitmap representation of the configuration space can be used for quick evaluations (an efficient algorithm for building a bitmap representation of $\mathcal{C}_{free}$ is given in [12]).

Note that L^*_K represents the cost of the optimal one-stage strategy from each configuration x_K. More generally, L^*_{K-i} represents the cost of the optimal $(i+1)$-stage strategy from each configuration x_{K-i}. For a motion planning problem, one is typically concerned only with strategies that require a finite number of stages before terminating in the goal region. For a small, positive δ the dynamic programming iterations are terminated when $|L^*_k(x_k) - L^*_{k+1}(x_{k+1})| < \delta$ for all values in the configuration space. This assumes that the robot is capable of selecting actions that halt it in the goal region. The resulting stabilized cost-to-go

function can be considered as a representation of the optimal strategy. Note that no choice of K is necessary because termination occurs when the loss values have stabilized. Also, only the representation of L^*_{k+1} is retained while constructing L^*_k; earlier representations can be discarded to save storage space.

4.2 The new approach

The approach outlined in Section 4.1 will numerically determine the optimal motion strategy for a very broad class of problems. It can furthermore be extended in a variety of ways to handle complications such as stochastic predictability uncertainty [18]. In spite of these successes, its computational complexity usually limits its practical applicability to problems that have two or three degrees of freedom.

The following two aspects of the previous method greatly contribute to the computational expense: 1) The number of iterations over the configuration space is equal to the number of stages in the longest (in terms of time) optimal trajectory that reaches the goal; 2) Each evaluation of $L^*(x)$ requires an interpolation between 2^n neighbors if $\mathcal{C}_{free}$ is n-dimensional. The new approach obtains the optimal strategy in a single iteration over the configuration space and uses only $n+1$ neighbors for each interpolation. The first improvement is made by styling the computation in a manner that is similar to the execution of Dijkstra's algorithm for graphs. The second improvement is made by approximating the cost-to-go function by constructing a simplicial complex on the configuration space.

Using a simplicial complex A set of *control points*, $p \in \mathcal{C}_{free}$, will be introduced to replace grid points from the previous approach. For a set of control points, $p_1, p_2, \ldots, p_{n+1}$, let $[p_1, p_2, \ldots, p_{n+1}]$ denote their convex hull, which will be referred to as a *simplex*. In two dimensions, a triangulation is obtained. For an n-dimensional configuration space, a *simplicial complex* is constructed [27]. Every d-dimensional simplex has $d+1$ faces, each of which are $(d-1)$-dimensional simplexes. Furthermore, for every pair, S_1, S_2, of i-dimensional simplexes in the simplicial complex, either $S_1 \cap S_2 = \emptyset$ or S_1 and S_2 share an $(i-1)$-dimensional face.

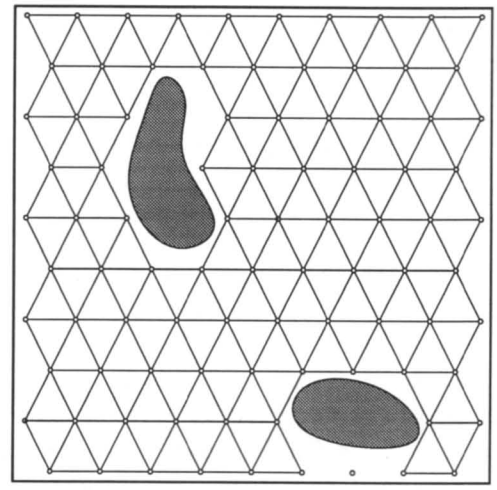


Figure 1: *Simplexes are only constructed in the obstacle-free portion of $\mathcal{C}_{free}$.*

The obstacle regions are avoided in the construction of the simplicial complex, as shown in Figure 1. In practice one might simply generate a simplex if all of its corresponding control points lie in $\mathcal{C}_{free}$. This approach might be efficient; however, to ensure correctness of the results, one must determine whether the entire convex hull of the control points lies in $\mathcal{C}_{free}$. One way this can be accomplished is to use a collision checking algorithm that provides a lower bound on the distance from the point to the obstacle in the configuration space.

Next, consider using the simplicial complex to represent a cost-to-go function. Suppose that the value of L^* is known for each of the control points. For an n-dimensional configuration space, the value of L^* at a point $x \in \mathcal{C}_{free}$ can be estimated through interpolation. Suppose x lies in the interior of an n-dimensional simplex. Note that any point x in a simplex can be uniquely expressed as a linear combination of the control points, $\{p_1, \ldots, p_{n+1}\}$ as

$$x = \sum_{i=0}^{n+1} \beta_i p_i, \tag{13}$$

in which each β_i is real-value such that $\beta_i \geq 0$ and $\sum_{i=0}^{n+1} \beta_i = 1$. The coefficients $\{\beta_1, \beta_2, \ldots, \beta_{n+1}\}$ are actually the *barycentric coordinates* of x. These coor-

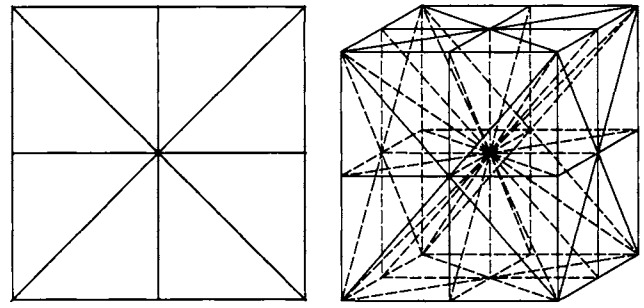
dinates can be used to provide a estimate of L^* through linear interpolation as

$$L^*(x) \approx \sum_{i=0}^{n+1} \beta_i L^*(p_i). \tag{14}$$

The next question that remains is how to choose a simplicial complex. Generally, three factors should guide the selection of a simplicial complex: 1) to yield a nice representation of the navigation function, each simplex should be as close to spherical in shape as possible, 2) for a given configuration, the problem of determining which simplex it belongs to should be efficient, and 3) the computation of the barycentric coordinates should be efficient. For the 2D case it is straightforward to define a simplicial complex by simply tiling the plane with equilateral triangles; however, in higher dimensions, the computational performance will depend greatly on the choice of simplexes that cover the configuration space. Ideally, one would like to construct a simplicial complex such that all d-dimensional simplexes are regular polytopes; however, as shown in [7], this is impossible, even in $\Re^3$ (e.g., $\Re^3$ cannot be tiled with identical tetrahedra, each of which having all faces be equilateral triangles).

We have recently shown that cubes or hypercubes can be triangulated using barycentric subdivision to yield a simplicial complex with very desirable properties: point location (which simplex does a given point belong to?) and barycentric coordinate computation can be performed in time $O(n \lg n)$, in which n is the dimension of the configuration space. The configuration space is tiled with cubes, and each cube is triangulated as shown in Figure 2 for the 2D and 3D cases. This result will be useful for the implementation in higher-dimensional configuration spaces, but at the present time we have only implemented the algorithm for the 2D case, which is described in Section 5.

Algorithm details An overview of the algorithm is given in Figure 3. The computation proceeds in a manner similar to Dijkstra's algorithm, except in our case, the *region* in which L^* is known increases incrementally, as opposed to the costs on *vertices* in a graph.



(a) (b)

Figure 2: *The cube can be triangulated using barycentric subdivision to obtain a simplicial complex on which efficient point location and barycentric coordinate computation are possible.*

```
Q ← {}; P_f ← all p ∈ G
For each p ∈ Pre(G) \ G
    Compute lub(p); INSERT(p,Q)
Until Q = ∅ do
    p_min ← POP(Q)
    For each p in Q ∩ (Pre(R(P_f ∪ p_min) \ R(P_f)))
        Recompute lub(p)
    P_f ← P_f ∪ {p_min}
    For each p ∈ Pre(R(P_f)) \ Q
        Compute lub(p)
        INSERT(p,Q)
```

Figure 3: *This algorithm computes the optimal feedback motion strategy in a single pass over the configuration space.*

For some loss functionals, the computation can be further simplified. For example, in the case of a minimum-time criterion, the computation can be organized as a wavefront expansion. This is analogous to the simplification of Dijkstra's algorithm to wavefront propagation in standard motion planning approaches.

The following notation is used in the algorithm description. For any control point p, let $lub(p)$ denote the lowest upper bound that can be assigned to the optimal cost-to-go at p, given any computation that has been performed. Let P_f denote a set of control points for which $L^*(x)$ is known. In other words, $lub(p) = L^*(p)$ for all $p \in P_f$. For a set of control points, P, let $R(P) \subseteq \mathcal{C}_{free}$ denote the set of all points that lie in

a simplex for which all of its control points are contained in P. In other words, R identifies a region over which a cost could be computed through interpolation of control points in P. Let $Pre(C)$ denote the set of all $x_{k+1} \in \mathcal{C}_{free}$ such that there exists some $u_k \in U$ with $x_{k+1} = f(x_k, u_k)$ and $x_k \in C$. In other words, $Pre(C)$ gives the set of configurations from which the set C is reachable in a single stage. This is a familiar construction in motion planning under uncertainty [22, 16], and the computation of $Pre(C)$ might be straightforward or extremely challenging, depending on the particular motion model.

Initially, it is assumed that $lub(p) = \infty$ if $p \notin G$. If $p \in G$, then $lub(p) = L^*(p) = 0$ (i.e., the optimal cost-to-go from the goal is always zero). The objective is to determine $L^*(p)$, for any control point p, that is reachable from G. Once this occurs, P_f will include all control points ($L^*(p) = \infty$ for unreachable control points), and $R(P_f)$ will be the maximal subset, formed from simplexes, on which the optimal cost-to-go can be determined. If there does not exist enough points initially such that 0 is obtained by interpolation over an n-dimensional simplex, then the algorithm will fail. This can occur, for instance, if the goal region is small and the spacing between control points is too large.

Step 1 initializes Q, which is a priority queue of control points that are sorted in ascending order according to $lub(p)$. Also, P_f is assigned to contain all control points that lie in G. Steps 2 and 3 initialize Q by inserting all control points that are reachable from G in a single stage, excluding points already in G. For points in $Pre(G)$, a finite value for $lub(p)$ can be computed by selecting the action $u_k \in U$ that produces the minimum loss. This computation can generally be performed by discretizing the action space; however, better techniques can be used in some particular cases.

Steps 5 to 11 are iterated until Q is empty. After each iteration, L^* is known for a new control point. In Step 3, p_{min} is removed from Q (the control point for which $lub(p)$ is the smallest). It is known that $lub(p_{min}) = L^*(p_{min})$ because a single-stage trajectory exists that brings p_{min} into $R(P_f)$ with less loss than from any other control point in Q. Once L^* is known for a new control point, P_f must be appropriately expanded, which adds new simplexes over which L^* can be obtained through interpolation. Better upper bounds can now be computed for some of the control points in Q. This update is performed by trying the actions $u_k \in U$ and evaluating the loss obtained for configurations that lie in the new simplexes (i.e., if p_{min} was used in the interpolation). After the control points are updated, the next part is to add new elements to Q, which is performed in Steps 9-11.

The priority queue, Q, is empty when there are no new control points that can reach $R(P_f)$ in a single stage. This is a natural halting point for the algorithm. The algorithm only stores the L^* for all control points in P_f. The cost-to-go at any configuration in $R(P_f)$ can be obtained by linear interpolation, which results in a navigation function that can be used to guide the robot into the goal.

5 Implementation

This section presents some results from an implementation of the algorithm in Figure 3 using GNU C++ and LEDA under Linux on a 200Mhz Pentium Pro PC. The current implementation applies only to case of a planar configuration space with a holonomic motion model. Thus, many practical computational issues remain to be explored as the dimension of the configuration space is increased and different nonholonomic systems are considered. For a small set of problems, it has been observed that the new computation method computes optimal navigation functions between 40 and 80 times faster than the method described in [18].

Three example problems are shown in Figures 4.a, 5.a, and 6.a. Figures 4.b, 5.b, and 6.b show level set contours of the optimal navigation function that was obtained using 80×80 control points. Figure 4.c shows 100 paths that were obtained by using the computed navigation function for Problem 1 and starting from random initial configurations. The table shown in Figure 7 shows computation times. A point robot is assumed. The extremal values of each axis are 0 and 100. When 40×40 control points were used, the robot was allowed to move 3.75 units in each stage. The state

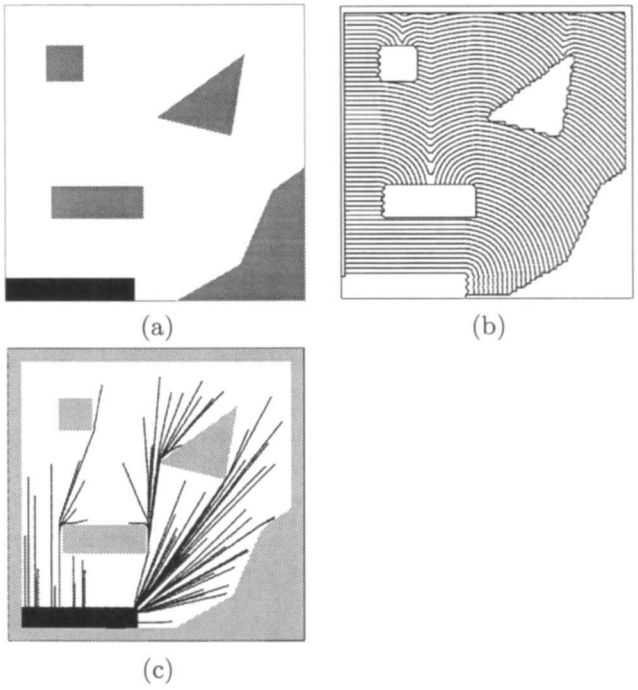


Figure 4: *Problem 1: a) The obstacles are shown in gray, and the goal region is shown in black; b) level set contours of the optimal navigation function for each integer,* $\{1, 2, 3, \ldots\}$*; c)* 100 *paths that were obtained by using the computed navigation function from randomly chosen initial positions.*

transition equation, f, is defined such that the robot is allowed to either move a fixed distance in any direction or remain stationary. When 80×80 control points were used, the robot was allowed to move 1.875 units in each stage. The action space was quantized into 32 and 360 orientations; the cost-to-go for each quantized value is computed, to numerically evaluate (9).

For this particular two-dimensional problem, one could evaluate (9) more efficiently, or even use methods similar to those in [9, 25]. The purpose of these experiments, however, is to estimate how the algorithm will perform when it is given more challenging problems in higher-dimensional configuration spaces. Therefore, no attempt was made to exploit properties particular to the 2D holonomic shortest-path problem.

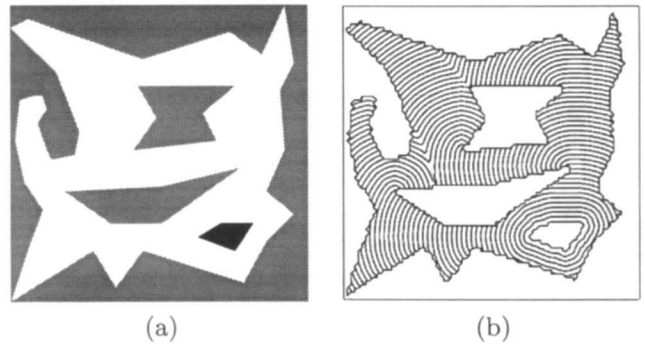


Figure 5: *Problem 2: a) Input; b) level set contours.*

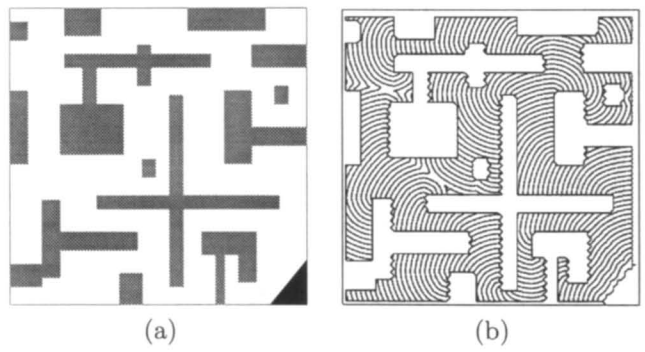


Figure 6: *Problem 3: a) Input; b) level set contours.*

6 Discussion

A new approach has been presented for computing optimal feedback motion strategies for holonomic or nonholonomic planning problems. Instead of precomputing a path, the solution takes the form of a navigation function, as considered in [26]. Two improvements are made over the previous approach to the problem of numerically computing optimal cost-to-go functions: 1) The solution can be obtained in a single iteration over the configuration space; 2) the number of interpolation steps have been reduced from 2^n to $n + 1$. It is important to note that once a navigation function is computed, it can be quickly utilized during execution. Although great performance improvement of the previous technique has been observed for a 2D configuration space, it is important to test these ideas on higher-dimensional spaces. As the dimension is in-

P	Res	Precmp	OldDP	NewDP	Fact
	$\|I\|x\|J\|, \|U\|$	(sec)	(sec)	(sec)	
1	$40 \times 40, 32$	0.45	10.32	0.20	51.6
1	$80 \times 80, 32$	1.77	84.16	0.82	102.6
1	$80 \times 80, 360$	1.77	822.93	8.45	97.4
2	$40 \times 40, 32$	0.61	8.03	0.18	44.6
3	$40 \times 40, 32$	1.37	12.57	0.21	59.8

Figure 7: *Computational performance from a GNU C++ implementation on a 200Mhz Pentium Pro PC running Linux. The columns denote: P = problem number, Res = resolutions, Precmp = precomputation time, OldDP = computation time for the original dynamic programming algorithm, NewDP = computation time for the new algorithm, Fact = factor of improvement in computation time.*

creased, greater benefits of using the simplicial complex as opposed to a grid are expected.

One aspect that deserves careful attention is the sensitivity of the approach to the choices of resolutions. To improve computational performance, it is tempting to increase the spacing between control points. However, this might result in a simplicial complex that does not adequately capture the topological information in $\mathcal{C}_{free}$ (i.e., a path class might be lost). It is also important to appropriately select Δt, which directly affects the distance traveled during each stage. If Δt is too small, the algorithm is likely to terminate prematurely because $R(P_f)$ will not be reachable. If Δt is too large, goal overshoot might occur. Similar issues also exist regarding the quantization of the action space U during the numerical evaluation of (9).

We are currently implementing the algorithm presented in Figure 3 for a three-dimensional configuration space, and intend to perform experiments to determine its performance and robustness. Two applications are worth considering at the outset: 1) optimal planning for car-like robots, and 2) optimal push planning for a robot that pushes rigid bodies on a planar surface. Our approach is expected to be particularly useful for applications that involve unpredictability, such as nonprehensile manipulation [8]. For example, imagine pushing a box with a mobile robot that makes points contact. A vision system can be used to monitor the position of the box, and the motion of the robot should be adjusted during execution to guide the box to the goal. A navigation function would be ideal in this case because a feedback motion strategy is directly obtained, as opposed to forcing the robot to follow a particular trajectory.

In the longer term it will be interesting to attempt generalizations of the algorithm. As argued in [19], one powerful advantage of a unified mathematical framework for motion planning problems is the relative ease of obtaining generalizations. It is expected that this will indeed be the case for the algorithm presented in this paper. One possible generalization is to use heuristic underestimates of the cost-to-go to design an algorithm that is similar to the A^* search generalization of Dijkstra's algorithm. The technique might also apply to problems that involve stochastic uncertainty in predictability, which can dramatically improve computational performance for problems such as computing optimal manipulation strategies under uncertainty [20], or planning in a partially-predictable environment [21].

Acknowledgments

I thank Prashanth Konkimalla for his help with the implementation. I also thank Tammy Verstraete for making corrections to this manuscript.

References

[1] P. K. Agarwal, J.-C. Latombe, R. Motwani, and P. Raghavan. Nonholonomic path planning for pushing a disk among obstacles. In *IEEE Int. Conf. Robot. & Autom.*, 1997.

[2] J. Barraquand, B. Langlois, and J. C. Latombe. Numerical potential field techniques for robot path planning. *IEEE Trans. Syst., Man, Cybern.*, 22(2):224–241, 1992.

[3] J. Barraquand and J.-C. Latombe. Nonholonomic multibody mobile robots: Controllability and motion planning in the presence of obstacles. *Algorithmica*, 10:121–155, 1993.

[4] R. E. Bellman. *Dynamic Programming.* Princeton University Press, Princeton, NJ, 1957.

[5] D. P. Bertsekas. Convergence in discretization procedures in dynamic programming. *IEEE Trans. Autom. Control*, 20(3):415–419, June 1975.

[6] L. G. Bushnell, D. M. Tilbury, and S. S. Sastry. Steering three-input nonholonomic systems: the fire truck example. *Int. J. Robot. Res.*, 14(4):366–381, 1995.

[7] H. S. M. Coxeter. *Regular Polytopes.* Dover Publications, New York, NY, 1973.

[8] M. Erdmann. An exploration of nonprehensile two-palm manipulation using two zebra robots. In J.-P. Laumond and M. Overmars, editors, *Algorithms for Robotic Motion and Manipulation*, pages 239–254. A K Peters, Wellesley, MA, 1997.

[9] J. Hershberger and S. Suri. Efficient computation of Euclidean shortest paths in the plane. In *Proc. 34th Annual IEEE Sympos. Found. Comput. Sci.*, pages 508–517, 1995.

[10] A. Isidori. *Nonlinear Control Systems.* Springer-Verlag, Berlin, 1989.

[11] L. E. Kavraki. *Random Networks in Configuration Space for Fast Path Planning.* PhD thesis, Stanford University, 1994.

[12] L. E. Kavraki. Computation of configuration-space obstacles using the Fast Fourier Transform. *IEEE Trans. Robot. & Autom.*, 11(3):408–413, 1995.

[13] R. E. Larson. A survey of dynamic programming computational procedures. *IEEE Trans. Autom. Control*, 12(6):767–774, December 1967.

[14] R. E. Larson and J. L. Casti. *Principles of Dynamic Programming, Part II.* Dekker, New York, NY, 1982.

[15] J.-C. Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, Boston, MA, 1991.

[16] J.-C. Latombe, A. Lazanas, and S. Shekhar. Robot motion planning with uncertainty in control and sensing. *Artif. Intell.*, 52:1–47, 1991.

[17] J.-P. Laumond, P. E. Jacobs, M. Taix, and R. M. Murray. A motion planner for nonholonomic mobile robots. *IEEE Trans. Robot. & Autom.*, 10(5):577–593, October 1994.

[18] S. M. LaValle. *A Game-Theoretic Framework for Robot Motion Planning.* PhD thesis, University of Illinois, Urbana, IL, July 1995.

[19] S. M. LaValle. Robot motion planning: A game-theoretic foundation. In *Proc. 2nd Int'l Workshop on the Algorithmic Foundations of Robotics*, July 1996.

[20] S. M. LaValle and S. A. Hutchinson. An objective-based framework for motion planning under sensing and control uncertainties. *International Journal of Robotics Research*, 17(1):19–42, January 1998.

[21] S. M. LaValle and R. Sharma. On motion planning in changing, partially-predictable environments. *International Journal of Robotics Research*, 16(6):775–805, December 1997.

[22] T. Lozano-Pérez, M. T. Mason, and R. H. Taylor. Automatic systhesis of fine-motion strategies for robots. *Int. J. Robot. Res.*, 3(1):3–24, 1984.

[23] K. M. Lynch and M. T. Mason. Stable pushing: Mechanics, controllability, and planning. In *Algorithmic Foundations of Robotics.* A. K. Peters, Boston, 1995.

[24] K. M. Lynch and M. T. Mason. Stable pushing: Mechanics, controllability, and planning. *Int. J. Robot. Res.*, 15(6):533–556, 1996.

[25] J. S. B. Mitchell. Shortest paths among obstacles in the plane. *Int. J. Comput. Geom. & Appl.*, 6(3):309–332, 1996.

[26] E. Rimon and D. E. Koditschek. Exact robot navigation using artificial potential fields. *IEEE Trans. Robot. & Autom.*, 8(5):501–518, October 1992.

[27] J. J. Rotman. *Introduction to Algebraic Topology.* Springer-Verlag, Berlin, 1988.

[28] J. A. Sethian. *Level set methods : Evolving interfaces in geometry, fluid mechanics, computer vision, and materials science.* Cambridge University Press, 1996.

[29] P. Svestka and M. H. Overmars. Coordinated motion planning for multiple car-like robots using probabilistic roadmaps. In *IEEE Int. Conf. Robot. & Autom.*, pages 1631–1636, 1995.

Synergistic Robots for Surgery: An Algorithmic View of the Approach

Jocelyne Troccaz, *TIMC/IMAG Laboratory, France*
Yves Delnondedieu, *TIMC/IMAG Laboratory, France*

The purpose of Computer-Assisted Surgery (CAS) is to help physicians or surgeons to plan and to execute optimal strategies from multi-modal data. The execution of such planned strategies may be assisted by guiding systems. Passive systems based on 6D localizers give information to the operator such that he can compare the planned task to the real one he executes. Active systems are robots that realize part of the intervention autonomously. Semi-active systems, more generally called synergistic systems, are based on the cooperation of a robotic device with a human operator. We have developped such a synergistic device : PADyC (Passive Arm with Dynamic Constraints). The basic principle of PADyC is to have a manually actuated arm that constrains the authorized motions of the surgical tool held by the human operator in function of a planned task. For instance, if the task is to stay inside a given 3D cartesian region, the motions of the tool will be free inside the region whilst some of them will be forbidden on the borders of this region. These constraints are realized by a patented mechanical system.

The constraints are computed in the joint space from a cartesian description of the task. The algorithmic problems raised by such a device are very close to those of robot path planning. Since the environment of the robot is not completely structured, off-line computation is difficult and the algorithmic complexity of the algorithms has to be kept low. We have developped an approach for a 3 degrees of freedom prototype taking care of being able to generalize our algorithms to 6D.

In this paper, we will briefly describe the state of the art of surgical robotics and present the principle of PADyC and we will focus on the algorithms allowing to map the surgical task in the joint space of the robot.

1 Introduction

The last decades have seen the development of more and more minimally invasive surgical techniques. Minimally invasive surgery implies the definition of very localized anatomical targets and the realization of the surgical action through natural openings or very small incisions. This raises a certain number of problems for the surgeon:

- surgical planning generally requires the fusion and processing of multi-modal data, a priori information and specific models (biomechanical for instance)
- the planned strategy has to be transferred to the surgical site
- "key-hole" access to the anatomical site results in a lack of perception (vision, tactile feeling) and dexterity.

An interface of some sort is needed in order to connect the "information world" of images, plans and computers, to the physical world of surgeons, patients and tools. This device has to supplement the surgeon's perception and dexterity, increase his accuracy, and allow him to access to distant anatomical targets through complex trajectories. Such an interface device is called a guiding system.

Computer-Assisted Surgery (CAS) systems generally include three main components: (1) multi-modal data acquisition, processing and fusion, (2) surgical planning capabilities and (3) such a guiding system. Guiding systems are passive, semi-active or active. The degree of passivity corresponds to the type of interaction between the human operator and the device:

- Passive systems display information to the surgeon about the position of the surgical tool relative to anatomical data [14],[8] or to a pre-planned strategy [13],[6]. The surgeon is totally in charge of the action.
- Active systems realize a part of the intervention autonomously. For instance, a robot may machine a bone [17], hold a sensor [3] or a radiation tool [1] without the need for interaction with a human operator who generally supervises the action.
- A semi-active system involved a combined action with the human operator for the complete realization of the task. For example, a mechanical guide brought in position by a robot may align a linear drilling trajectory that the surgeon will execute [21],[12].

Semi-active systems allow a safe, ergonomic, direct and accurate transfer of the surgical planning in the operating site but currently they are restricted to rather elementary tasks (linear motions and planar cuts). Recently, different groups [20] have proposed a new generation of semi-active systems called synergistic devices. Synergistic devices are intended for direct physical guidance of a surgical tool, a tool that is also held and controlled by an operator. Contrary to the semi-active systems, the mechanical constraints imposed to the operator's motions are programmable instead of being frozen in a task-dependent hardware. Under computer control, the synergistic device dynamically filters the motions proposed by the surgeon to keep only those which are compatible with the task (cf. figure 1). Synergistic cooperation has the benefit that the robot provides accurate geometric motions whilst the surgeon holding the tool can feel the forces applied and modify them appropriately based on his a priori knowledge, his innate skills and his sensing capabilities. This is a very important element of safety and psychological acceptance of surgical robots.

2 Principles of the PADyC

We are investigating the principle of synergistic devices through the development of the PADyC system [19, 18]. PADyC stands for **P***assive* **A***rm with* **Dy***namic* **C***onstraints*. PADyC is a passive encoded arm with no actuation. It is manually actuated by an operator but constrains the motion of the end-effector according to a pre-planned task. For instance, the surgeon may have planned an optimal planar osteotomy*[1] to position a prosthesis according to biomechanical models. During surgery, PADyC may constrain the surgical tool (a saw) to remain inside the plane while the motion in the plane is free and imposed by the operator.

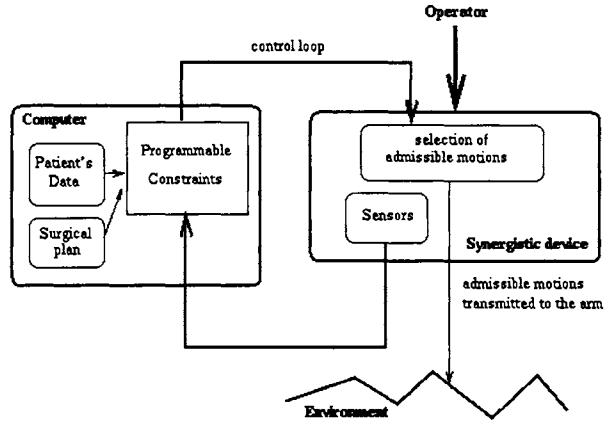


Figure 1: *Principle of a synergistic device.*

2.1 Mechanical architecture

In order to constrain the motion of the end-effector in function of a given task, each joint of the PADyC is able to provide the four following functions:

F1 : motion authorized in the positive direction only

F2 : motion authorized in the negative direction only

F3 : motion authorized in both direction

F4 : no motion authorized

These four functions are obtained using a patented mechanism associated to each encoded joint. It consists of two freewheels mounted in opposite directions.

[1]The medical terms tagged with a star are introduced in a short glossary at the end of the paper

A freewheel naturally provides the basic function of unidirectional motion. Two motors (one for each freewheel) allow to clutch or unclutch the freewheels to obtain the four functions. The velocities ω_i^- and ω_i^+ of the two motors associated to the joint i are computer-controlled. As presented in figure 2, thanks to this mechanism, one can guarantee that at each instant the velocity ω_i of the joint i is bounded by $\omega_i^- \leq \omega_i \leq \omega_i^+$.

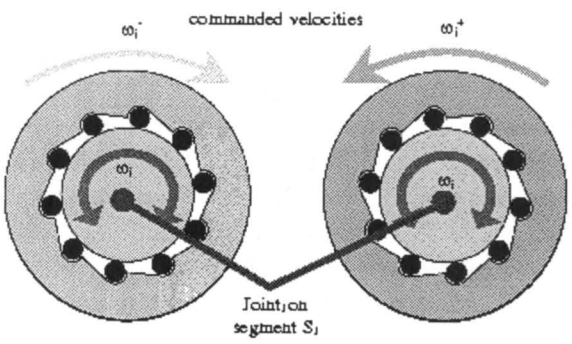


Figure 2: *Mechanical structure of the joint: two freewheels are associated to each joint. One part is moved by a motor at velocity $\omega_i^{-,+}$. The operator moves the other part of the freewheel at velocity ω_i. Depending on the values of the relative velocities $(\omega_i^{-,+} - \omega_i)$, the motion is blocked or authorized.*

The four functions F1 to F4 are obtained by a suitable choice of the $\omega_i^{-,+}$ values.

F1 : $\omega_i^- = 0$ and $\omega_i^+ \neq 0$

F2 : $\omega_i^- \neq 0$ and $\omega_i^+ = 0$

F3 : $\omega_i^{-,+} \neq 0$

F4 : $\omega_i^{-,+} = 0$

Therefore, the motions of the end-effector proposed by the operator are filtered by the mechanism to keep only those which are compatible with the task. The intrinsic safety of such a system is very good since the arm cannot move autonomously (the motors cannot drive the joints and the user cannot backdrive the motors thanks to a suitable design). The joints are naturally locked when unpowered.

2.2 Programming modes

Different types of therapeutic actions may require an assistance. Their geometric complexity may vary a lot. In some cases, the surgeons may need only to know where is the surgical tool with respect to anatomical structures: this is typically the case in endoscopic actions where localization from endoscopic images* may be a very difficult task. In other cases, the surgeon has to precisely execute a pre-planned action. Among others, it may be a linear action (e.g. drilling, puncturing), a planar one (e.g. sawing, chiseling), a conical one (for instance in laparoscopic* surgery or endo-ocular* surgery) or a spatial one (e.g. cavity preparation for prosthesis fit, resection* operations or multiple radiation of tumors). In some other cases, the surgeon may require assistance to position a 3D structure relatively to another structure: for instance a prosthesis has to be positioned relatively to a bone, a bone relatively to another bone for osteosynthesis*. Finally, some delicate tasks may require the definition of anatomical obstacles to be avoided.

Based on these considerations, we have defined four basic programming modes of PADyC.

Free mode: the motion of the surgical tool is free but tracked in 6D and displayed to the operator.

Position mode: a 6D position has to be reached without paying attention to the trajectory.

Trajectory mode: a goal position has to be reached taking care of the executed trajectory.

Region mode: the tool is constrained to remain within a pre-defined region (a plane, a cone or any 3D shape). Motion inside the region is unconstrained. Equivalently, the motion of the tool may be free everywhere except in a forbidden region.

Combinations of these modes are possible but they are out the scope of this paper.

The task is described in terms of these programming modes. It may be generated from the planning tools

included in classical CAS systems. The task is defined in the cartesian space: for instance the shape of a 3D region in which the tool must remain is computed from anatomical data of the patient obtained before the intervention.

2.3 Control issues

As presented in section 2.1, at each instant, each joint is constrained to move in a given direction with a bounded amplitude. The velocity window $[\omega_i^-, \omega_i^+]$ associated to each joint i guarantees that during the next sampling period Δt, the configuration of the robot belongs to a known hyperparallelepiped parallel to the joint axes, called *window of admissible configurations* (cf. figure 3[2]). For a n joint robot, the window of admissible configurations is defined as $WAC(t) = (\Delta q_1^-, \Delta q_1^+, \Delta q_2^-, \Delta q_2^+, ..., \Delta q_n^-, \Delta q_n^+)$.

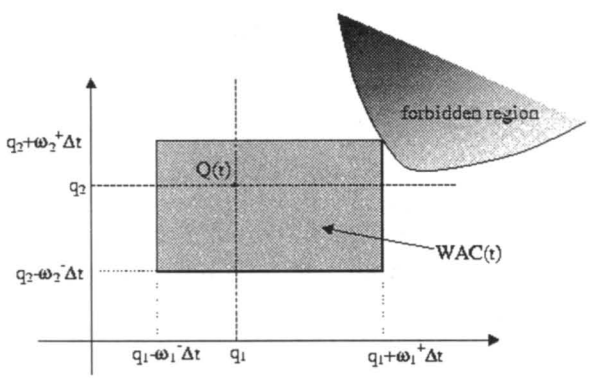


Figure 3: *Window of admissible configurations noted $WAC(t)$.*

The constraint executed at instant t and available until $t + \Delta t$ may be described by the vector $\Omega(t) = (\omega_1^-, \omega_1^+, \omega_2^-, \omega_2^+, ..., \omega_n^-, \omega_n^+)$ where $\omega_i^{-,+}$ are the velocity orders to be sent to the clutching motors of the joint i. *The major control issue of PADyC consists in determining, from the current configuration of the robot and the pre-planned task, the vector of constraint $\Omega(t)$ that has to be applied during the next sampling period to guarantee the correct execution of the task.* This necessitates the computation of the $WAC(t)$ from which $\Omega(t)$ is very easilly deduced: $\Omega(t) = WAC(t)/\Delta t$.

The low-level control of PADyC is very simple since it relies on the execution of the velocity orders sent to the motors. Meanwhile, its high-level control requires to map the geometric description of the task from the cartesian space where it is defined to the joint space where the constraints apply. This is a very well-known problem of robot motion planning [11] for which computational complexity is a critical issue. Moreover, in our application, in some cases the task may vary with time: for instance, a drilling trajectory in a vertebra will move in function of the motions of this vertebra during surgery. Therefore, the computational cost of the algorithms used to map the cartesian space to the joint space must be kept sufficiently low to allow a real-time control.

3 Computing the Constraints

3.1 Free mode

Free mode is a very trivial case since the velocity windows only have to be set wide open. A basic solution would consist in defining $\Omega(t)$ as $\Omega(t) = (-\omega_1^{max}, \omega_1^{max}, -\omega_2^{max}, \omega_2^{max}, ..., -\omega_n^{max}, \omega_n^{max})$. Nevertheless, to make the guiding system more transparent, $\Omega(t)$ is modified to deal with anisotropy in the motion. If Q^c is the current configuration at instant t, $\Omega(t)$ is computed from the Jacobian matrix $J(Q^c)$ such that during Δt, the maximum cartesian velocity has the same amplitude V_m along all the axes of the cartesian space. $\Omega(t) = (\frac{-V_m}{\|\vec{B_1}\|}, \frac{V_m}{\|\vec{B_1}\|}, \frac{-V_m}{\|\vec{B_2}\|}, \frac{V_m}{\|\vec{B_2}\|}, ..., \frac{-V_m}{\|\vec{B_n}\|}, \frac{V_m}{\|\vec{B_n}\|})$ where $\vec{B_i}$ is the ith column vector of $J(Q^c)$.

3.2 Position mode

Being given a position and orientation of the end-effector to be reached, one of the corresponding configurations, Q^f, is computed. $WAC(t)$ is very easilly deduced from the current configuration Q^c at instant t and the goal one (cf. figure 4).

[2] In the following, for sake of readability, illustrations will correspond to the case of a 2 degrees of freedom PADyC robot.

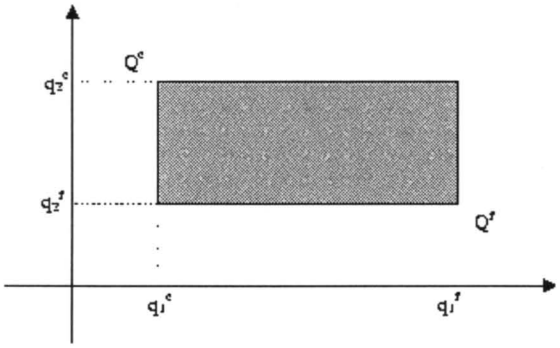


Figure 4: *Position mode: computed $WAC(t)$ (in this example $WAC(t) = (0, (q_1^f - q_1^c), (q_2^f - q_2^c), 0)$.*

If $Q^c = (q_1^c, q_2^c, ..., q_n^c)$, $Q^f = (q_1^f, q_2^f, ..., q_n^f)$ and $\Delta q_i = q_i^f - q_i^c$ for $i \in [1, n]$ then $\Omega(t)$ is computed such that:

$$\text{if } \Delta q_i \geq 0 \text{ then } \omega_i^- = 0 \text{ and } \omega_i^+ = \Delta q_i / \Delta t$$
$$\text{else } \omega_i^- = \Delta q_i / \Delta t \text{ and } \omega_i^+ = 0.$$

This algorithm is iterated every Δt until Q^f is reached. Because of the mechanical structure of the joint, the operator cannot move the robot backward but is constrained to progress towards Q^f. Except when the function F3 (cf. section 2.1) is active, which is not the case in the position mode, $\frac{dq_i}{dt}$ is a monotonous function between t and $t + \Delta t$.

3.3 Region mode

3.3.1 Principle

The initial algorithm we implemented (cf. [18]) was based on the computation of the region of interest in the joint space and the determination of a maximal $WAC(t)$ compatible with this region (cf. figure 5). In the particular case we treated (a two dof PADyC prototype) this raised no problem but obviously the generalization of such an algorithm to a 6 dof system looked unpracticable. It is why we have developped another approach inspired by [2] that avoids the explicit representation of the region in the joint space. The basic idea consists in replacing a 6D problem -computing available positions and orientations of a moving object- by the combination of 3D problems -computing available positions of points located on this object-. More precisely, the algorithm is based on the use of a set of k control points P_j placed on the surgical tool for which the task is defined (cf. figure 6)[3]. For each control point P_j, a range of local admissible motions in the cartesian space is computed from the cartesian description of the task. Then, a $WAC_j(t)$ compatible with the position of P_j at instant t is deduced (see § 3.3.2). Finally, $WAC(t)$ is defined as a function of the k sets $WAC_j(t)$ (see § 3.3.3).

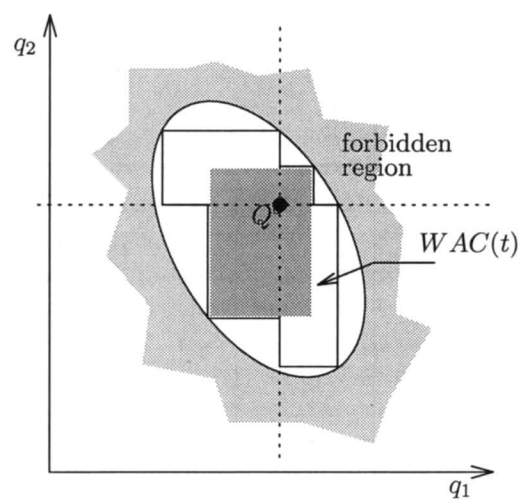


Figure 5: *Region mode: computing $WAC(t)$ in the articular space.*

3.3.2 Computation of $WAC_j(t)$

Let consider that the robot is, at the instant t, at the configuration Q^c. The position of each control point P_j in a fixed reference frame of the cartesian space can be computed using the robot direct kinematic model. For

[3]Other points can be added on the links of the robot in case where potential collisions with the environment have to be avoided

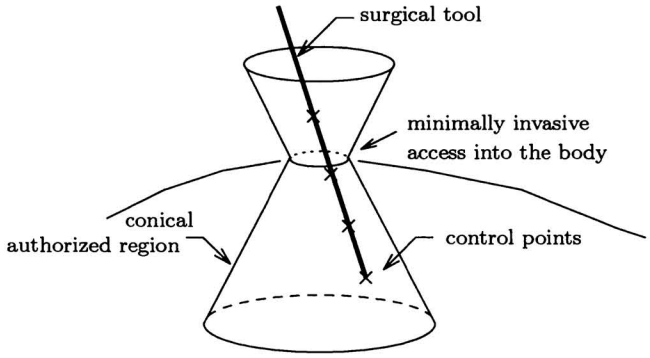


Figure 6: *Region mode: defining a set of control points for instance in a laparoscopic procedure.*

each point P_j, a ball $B_j(t)$ of admissible positions is determined locally. To make this computation very fast, a 3D distance map of the region in a local reference frame is built before the execution of the task.

The second stage of the algorithm consists in determining a $WAC_j(t)$ which image in the cartesian space, $WAC_j^{cart}(t)$ (cf. figure 7a), is included in the ball $B_j(t)$. Because Δt is small, $B_j(t)$ is small. Thus a linear approximation of the cartesian image of $WAC_j(t)$ can be used. In other words, we consider that $WAC_j^{cart}(t)$ will be included in $B_j(t)$ if the cartesian image of all the vertices of $WAC_j(t)$ are inside $B_j(t)$.

In practice an initial $WAC_j(t)$ noted $WAC_j^0(t)$ is selected (cf. free mode). For each configuration $Q^k = Q^c + \Delta Q^k$, vertex of $WAC_j^0(t)$, we compute the corresponding position P_j^k of P_j as $P_j^k = P_j + J(Q^c).\Delta Q^k$. If all the points P_j^k are inside $B_j(t)$ then $WAC_j(t) = WAC_j^0(t)$. Otherwise the points P_j^k are moved inside $B_j(t)$ to give $P_j^{k'}$. The $P_j^{k'}$ are mapped back to the articular space (configurations represented by small crosses on figure 7b). A new value of $WAC_j(t)$ is deduced (see figure 7b).

3.3.3 Computation of $WAC(t)$

Finally $WAC(t)$ is simply defined as $WAC(t) = \bigcap_{j=1}^{k} WAC_j(t)$ (cf. figure 8).

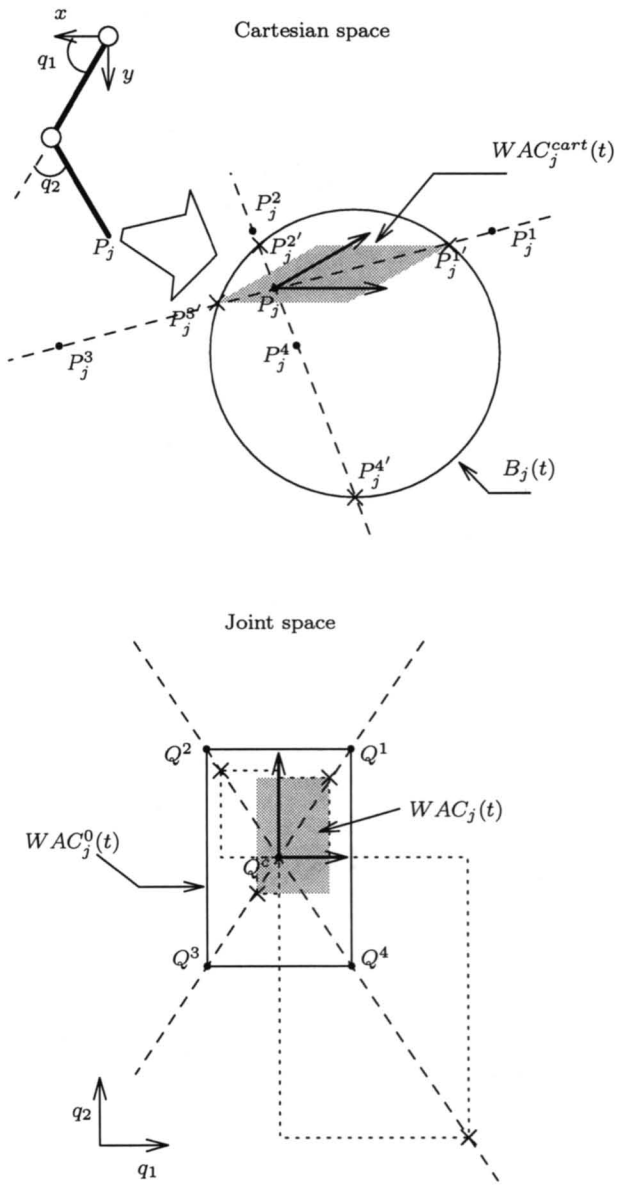


Figure 7: *Region mode: computing $WAC_j(t)$.*

3.3.4 Discussion of this approach

We have presented a method to compute a $WAC(t)$ compatible with the task in the region mode. This method has two major advantages. Firstly, because the distance map attached to a region is a local information, it has not to be updated on-line if the region

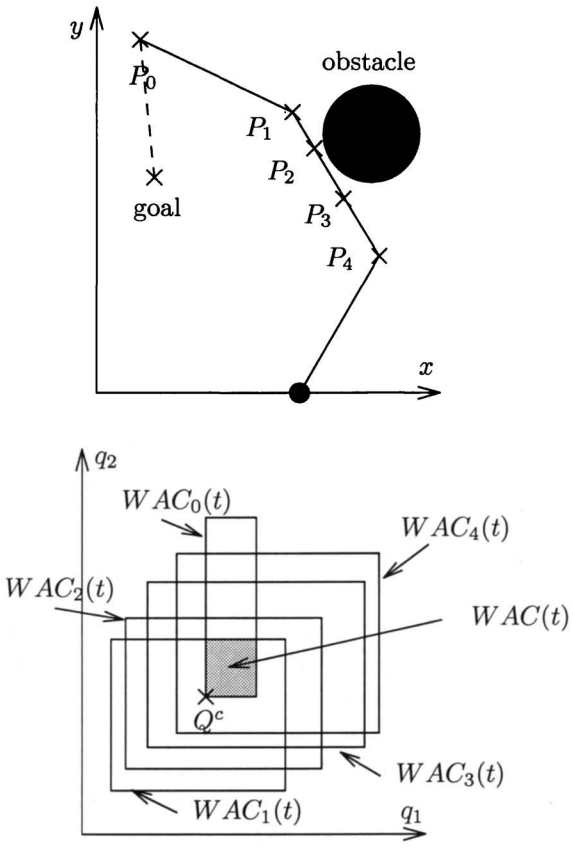


Figure 8: *Region mode: fusion of constraints.*

moves. This improves a lot the performance in case of dynamical tasks. Secondly, as pointed out by [2], the computational cost of such a method can be reduced a lot using parallel processing of the sets $WAC_j(t)$. Finally, let us mention that the solution $WAC(t)$ is not unique since the only requirement of the constraint is to stay inside the region. Some modifications could be introduced in the algorithms to fit a particular criterion (maximal area for instance).

3.4 Trajectory mode

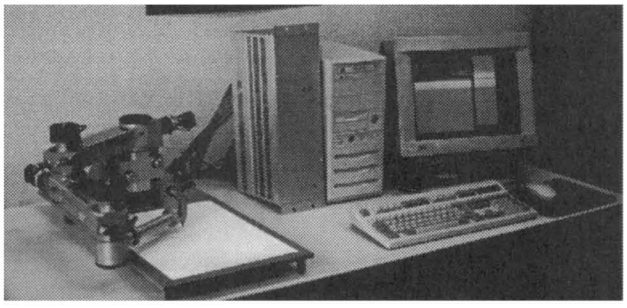
A trajectory is defined in the 3D cartesian space and associated to a required accuracy. For instance a linear trajectory has to be drilled in a vertebra with a maximum deviation of 1 mm perpendicular to the direction of drilling. This results in the definition of a corridor of possible positions for the surgical tool. As described in the previous section, control points may be associated to the tool and $WAC(t)$ can be computed applying to this corridor the algorithms of the region mode.

4 Status of the PADyC Project

PADyC is a four year project. We have designed four prototypes. The more recent one is a three dof prototype in the plane (cf. figure 9a) using cheap off-the-shelf components. Different releases of the algorithms developed for the free, position, trajectory and region modes have been implemented and tested on the prototypes. Despite a medium mechanical performance of the prototype (rigidity in particular), we have demonstrated the ability to produce programmable constraints that allows the operator to easilly execute pre-planned tasks (cf. figure 9b). The human operators who tested the system, and in particular the surgeons, were very enthusiastic about such an approach. On the control point of view the performance of the algorithms presented above allow a real-time control and can be applied to more complex kinematic architectures. The next stage of this project is to select a surgical domain of applications and to design a six degrees of freedom system.

5 Related Work

Passive systems, as introduced in section 1, rely on the combined use of a 6D localizer which allows the tracking of the position of the tool and of a display which deliver this information to the surgeon. Different technologies of localizers (optical, magnetic, ultrasonic, mechanical) are available. When a mechanical passive arm is selected, brakes can be super-imposed to the joints: the surgeon thus has complete and free control of position, until a desired position is reached, at which point the localizer can be entirely frozen and subsequently used as a rigid guide. An extension of this idea, which provides the function of a synergistic device, is to use brakes that can be fractionally activated rather than turned entirely on or off. With appropriate control of the brakes, such a device could constrain the motion of a surgical tool grasped jointly



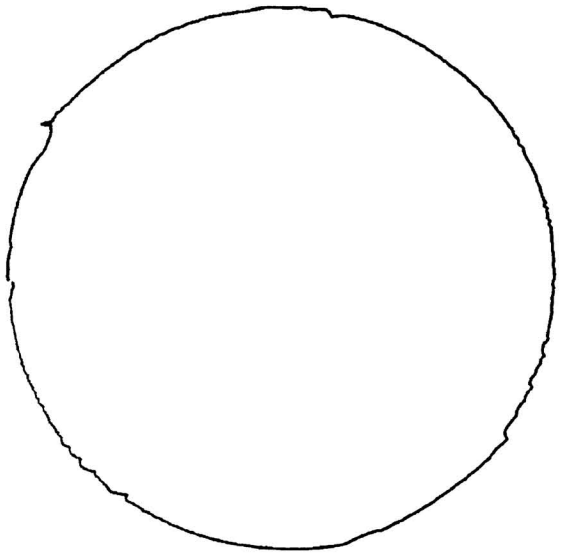


Figure 9: *PADyC prototype: (a/top) three degree of freedom prototype - (b/bottom) a trajectory with an accuracy of +/-1mm.*

by the robot and the surgeon. This idea was introduced in [16] who wanted to improve the ergonomy of passive systems based on mechanical localizers. More recently, this idea of programmable braking has been implemented in two systems named IMCAS [7] and P-TER [5]. These manually actuated arms are respectively based on piezoelectric and particle brakes. In both systems, a force sensor allows for the detection of the direction of desired motion. Depending on the configuration and on the task to be executed, the brakes are computer-controlled in order to filter the proposed motion. P-TER includes a redundant number of brakes in order to be able to couple the joint velocities ω_i and to produce a larger range of constraints (for instance $\omega_1 + \omega_2 \leq C$). In both projects, planar two d.o.f. prototypes have been built.

Like PADyC, IMCAS and P-TER, Cobot [4, 15] is actuated by a human operator. Constraints are produced using Continuously Variable Transmissions (CVT). Let us consider, for instance, a two degree of freedom translational cobot. The CVT consists of a rolling wheel, free to roll on a flat working surface and a handle, attached to it. The user and the cobot interact through the handle, and the workspace of the cobot is the horizontal plane in which the user can move this handle. The wheel is oriented using a computer-controlled motor. The CVT couples the x and y velocities by steering the wheel to reach the orientation α such that : $\tan\alpha = vy/vx$. This allows the execution of a predefined trajectory. CVTs have been also designed for arm-like cobots with revolute joints. By analogy a rotational CVT holds two angular joint velocities in proportion $T = \omega_2/\omega_1$. For a n joints cobot, (n-1) CVTs are necessary to couple the n joints such that a trajectory in space may be executed.

A conventional robot may attempt to cooperate with a human user by measuring the user's applied force and driving joint motors to comply with these forces. However the transparency is usually poor. ACROBOT (or Active Constraint ROBOT) is an actuated arm specially designed for transparent cooperation with a human user, while nevertheless using actuated joints [9, 10]. ACROBOT uses backdrivable motors and transmissions, where conventional robots are usually made strong and stiff at the expense of backdriveability. Mechanically, ACROBOT places the human user and the robot's motor on a more nearly equal basis for controlling position. For tasks involving the region mode, ACROBOT's motors are actively driven to comply with the user's force. Good transparency can result due to mechanical sharing of forces between motor and human, made possible by backdriveability and by servo system action aiding the surgeons de-

sired motion. As the user approaches and then contacts a constraint surface defined in the pre-operative plan, the motors are actuated to gradually increases its resistance until, at the edge of the permitted region, it prevents further motion by the surgeon. The proportional and derivative coefficients of the PID control programmably vary during the execution of the task. A four dof system is being developped for knee surgery applications (TKA*).

6 Discussion

As described previously, different approaches implement the notion of synergistic devices. IMCAS and P-TER are based on programmable brakes. PADyC uses disengageable freewheels. Non holonomic coupling devices are the basic elements of Cobots. Finally, ACROBOT is a force-controlled device which actuation is programmably shared between the robot actuators and the operator. Basically, these different technologies allow to emulate different behaviors. Actuated robots such as ACROBOT behave as spring-damper systems. Devices based on programmable brakes generate viscous friction. PADyC and Cobot produce virtual walls to give rigid constraints. Selecting one of these technologies determines the type of physical constraints - rigid, plastic, elastic- that can be generated by the system and the type of application it is best-suited for.

Concerning the programming modes, Cobot implements the free, trajectory and region modes. Because of its principle, Cobot is naturally adapted to the trajectory mode. The free mode requires the use of a force sensor to detect the user's desired direction of motion and to steer the wheel in this direction. The region mode switches from the free mode, inside the region, to the trajectory mode, on the boundary of the region. ACROBOT implements a 2D region mode sufficient for the knee application. PADyC implements the free, position, trajectory and region modes. Because of its design, the trajectory mode is the less natural one for the system which generates windows of possible displacements. No elements are given for P-TER and IMCAS but it can be foreseen that, because of the technology, the trajectory mode will not be a natural mode for these systems.

These five systems are at different levels of development. ACROBOT took advantage of its more conventional technology and control scheme and is the closest to the application. The four other systems are based on new technologies and new control paradigms which are still in development and evaluation. Nevertheless, whatever the selected technology, with small variations in complexity, the implementation of control algorithms for the synergistic devices requires the ability to compute constraints from a cartesian description of the task. Very few attention has been paid to this problem which is mandatory to be able to develop from 4 to 6 dof devices. In this paper, we have proposed algorithms for the PADyC system that are not limited to a restricted number of dof. Such algorithms could probably be modified to be used for other technologies (programmable brakes in particular). More generally, we think that a common hardware and software platform should be implemented to integrate these different approaches and to take benefit of these emerging technologies. This is clearly a very interesting alternative to conventional robots for surgical applications and for any application that requires man/robot cooperation.

Acknowledgements

This research is supported by an INSERM grant (PSTRC GMCAO PARMPSTRC9607) and by european grants (IGOS and IGOSII Telematics projects).

References

[1] J. Adler, A. Schweikard, R. Tombropoulos, and J.-C. Latombe. Image-guided robotic radiosurgery. In *First Symposium on Medical Robotics and Computer Assisted Surgery (MRCAS'94)*, pages 291–297, Pittsburgh, PA, 1994.

[2] J. Barraquand and J. Latombe. Robot motion planning: a distributed representation approach. technical report STAN-CS-89-1257, Stanford University, Robotics Laboratory, Computer Science Dept, 1989.

[3] S. Boudet, J. Gariepy, and S. Mansour. An integrated robotics and medical control device to quantify atheromatous plaques: experiments on the arteries of a patient. In *IEEE/RSJ IROS'97 conference*, pages 1533–1538, 1997.

[4] J. Colgate, M. Peshkin, and W. Wannasuphoprasit. Nonholonomic haptic displays. In *IEEE International Conference on Robotics and Automation*, 1996.

[5] H. Davis and W. Book. Torque Control of a Redundantly Actuated Passive Manipulator. In *AACC*, 1997.

[6] A. DiGioia, D. Simon, B. Jaramaz, M. Blackwell, F. Morgan, R. O'Toole, B. Colgan, and E. Kischell. HipNav: Pre-operative planning and intra-operative navigational guidance for acetabular implant placement in Total Hip Replacement Surgery. In *Computer Assisted Orthopedic Surgery*. (to appear), 1997.

[7] S. Erbse, K. Radermacher, M. Anton, G. Rau, W. Boeckmann, G. Jakse, and H.-W. Staudte. Development of an automatic surgical holding system based on ergonomic analysis. In J. e. a. Troccaz, editor, *CVRMed-MRCAS'97 Proceedings, LNCS Series 1205*, pages 737–746. Springer-Verlag, March 1997.

[8] W. E. L. Grimson, T. Lozano-Perez, W. M. Wells, G. J. Ettinger, S. J. White, and R. Kikinis. An automatic registration method for frameless stereotaxy, image guided surgery, and enhanced reality visualization. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'94)*, pages 430–436, Seattle, Washington, 1994.

[9] S. Harris, W. Lin, K. Fan, R. Hibberd, J. Cobb, R. Middleton, and B. Davies. Experiences with Robotic Systems for Knee Surgery. In J. Troccaz, E. Grimson, and R. Mosges, editors, *CVRMed-MRCAS'97 Proceedings, LNCS Series 1205*, pages 757–766, Berlin, March 1997. Springer-Verlag.

[10] S. Ho, R. Hibberd, and B. Davies. Robot Assisted Knee Surgery. *IEEE Engineering in Medicine and Biology Magazine*, 14(3):292–300, May/June 1995.

[11] J. Latombe. *Robot Motion Planning*. Kluwer Academic, Norwell,MA, 1991.

[12] J. Lavallee, S. Troccaz, L. Gaborit, P. Cinquin, A. Benabid, and D. Hoffmann. Image guided robot: a clinical application in stereotactic neurosurgery. In *IEEE Int. Conf. on Robotics and Automation*, pages 618–625, Nice , France, 1992.

[13] S. Lavallee, P. Sautot, J. Troccaz, P. Cinquin, and P. Merloz. Computer Assisted Spine Surgery : a technique for accurate transpedicular screw fixation using CT data and a 3D optical localizer. *Journal of Image Guided Surgery*, 1(1):65–73, 1995.

[14] R. Mosges, G. Schlondorff, L. Klimek, D. Meyer-Ebrecht, W. Krybus, and L. Adams. Computer assisted surgery. an innovative surgical technique in clinical routine. In H. Lemke, editor, *Computer Assisted Radiology, CAR 89*, pages 413–415, Berlin, June 1989. Springer-Verlag.

[15] M. Peshkin, J. Colgate, and C. Moore. Passive robots and haptic displays based on nonholonomic elements. In *IEEE International Conference on Robotics and Automation*, 1996.

[16] R. Taylor, C. Cutting, Y. Kim, A. Kalvin, D. Larose, B. Haddad, D. Khoramabadi, M. Noz, R. Olyha, M. Bruun, and D. Grimm. A model-based optimal planning and execution system with active sensing and passive manipulation for augmentation of human precision in computer-integrated surgery. In *2nd International Workshop on Experimental Robotics*, Toulouse(France), June 1991. Springer Verlag.

[17] R. Taylor, B. Mittlestadt, H. Paul, W. Hanson, P. Kazanzides, J. Zuhars, B. Williamson, B. Musits, E. Glassman, and W. Bargar. An image-directed robotic system for precise orthopaedic surgery. *IEEE Trans. Robotics and Automation*, 10(3):261–275, 1994.

[18] J. Troccaz and Y. Delnondedieu. Semi-active guiding systems in surgery. A two-dof prototype of the passive arm with dynamic constraints (PADyC). *Mechatronics*, 6(4):399–421, June 1996.

[19] J. Troccaz, S. Lavallee, and E. Hellion. PADYC : A Passive Arm with Dynamic Constraints. In *ICAR (Int. Conf. on Adanced Robotics)*, pages 361–366, Tokyo, November 1993. JIRA, Tokyo.

[20] J. Troccaz, M. Peshkin, and B. Davies. The use of localizers, robots and synergistic devices in Computer Assisted Surgery. In J. Troccaz and al., editors, *CVRMed-MRCAS'97 Proc., LNCS Series 1205*, pages 727–736. Springer-Verlag, 1997.

[21] K. Y.S., H. J., J. E.A., and H. S. A robot with improved absolute positioning accuracy for CT guided stereotactic brain surgery. *IEEE Trans. on Biomedical Engineering*, 35(2):153–160, 1988.

Glossary

endoscopic images: video anatomical images acquired through natural openings of the body (trachea, rectum for instance) or small incisions. The generally limited field of view of the endoscopic

camera makes the interpretation of these images difficult

endo-ocular surgery: minimally invasive surgery of the eye (retina in particular) for which small incisions are made to introduce the surgical instruments, a light source and a video device. The only possible motions of the instruments are rotations around the entry point and translation along and rotation around the tool axis

laparoscopic surgery: minimally invasive surgery of the abdomen for which small incisions are made to introduce the surgical instruments, a light source and a video device. The only possible motions of the instruments are rotations around the entry point and translation along and rotation around the tool axis

osteotomy: cutting of a bone

osteosynthesis: rigid fixation of two bone fragments

resection: surgical removal of tissues (tumors for instance)

TKA or Total Knee Arthroplasty: placement of a prosthesis in the knee joint which requires the preparation of the tibial and femoral bones. This preparation consists in machining well-defined planar surfaces on the bones which provide contact surfaces with the prosthesis elements.

Taylor & Francis
Taylor & Francis Group
http://taylorandfrancis.com

Fourier Methods on Groups: Applications in Robot Kinematics and Motion Planning

Alexander B. Kyatkin, *Johns Hopkins University, Maryland, USA*
Gregory S. Chirikjian, *Johns Hopkins University, Maryland, USA*

In this paper we develop and apply a fast Fourier transform on the "discrete motion group" to the problem of computing the configuration space obstacles of mobile robots and the problem of finding the workspace density of discretely actuated manipulators with many states. Fast transforms allow us to compute efficiently convolution-like integrals that arise in robot kinematics and motion planning. The results of the implementation are discussed for particular examples.

1 Introduction

In this paper we address two problems in robotics which may be solved using similar mathematical tools: (1) the problem of computing the configuration space obstacles of mobile robots which move in a static environment; (2) the problem of computing the workspaces of discretely actuated m-link revolute manipulators. Both of these problems may be posed as convolutions on the Euclidean motion group [1]. $SE(n)$ can be viewed as the set of all $(n+1)\times(n+1)$ homogeneous transform matrices of the form

$$g = \begin{pmatrix} R & \mathbf{x} \\ \vec{0}^T & 1 \end{pmatrix},$$

where $R \in SO(n)$ (i.e., R is an $n \times n$ proper orthogonal matrix: $RR^T = 1$ and $\det(R) = +1$), $\mathbf{x} \in \mathbb{R}^n$, and the group operation is matrix multiplication of two homogeneous transform matrices. For the case of rigid body motion in the plane the homogeneous transforms take the simple form

$$g(x_1, x_2, \theta) = \begin{pmatrix} \cos\theta & -\sin\theta & x_1 \\ \sin\theta & \cos\theta & x_2 \\ 0 & 0 & 1 \end{pmatrix} \in SE(2)$$

A particularly useful subgroup of $SE(2)$ is the one where $\theta = 2\pi i/N$ for $i = 0, ..., N-1$. We call this group the *discrete motion group* of the plane. While $SE(n)$ has been studied previously in various applications in robotics and physics (e.g. [1, 2, 3, 4, 13]), and discrete subgroups of $SE(N)$ (where both rotation and translation are discretized) have been studied extensively in crystallography (e.g. [5]), the discrete motion group defined above is far less familiar to the robotics community. In fact, the authors are aware of only one previous work using this group, [28].

We develop and apply Fourier analysis on the discrete motion group. Robotics applications of Fourier analysis on $SE(N)$ have been addressed previously by the authors in several works. The advantage of using the discrete motion group is that this allows us to apply Fast Fourier Transform (FFT) algorithms [6, 7]. In practice this reduces the time required from hours to minutes for the examples in this paper.

We develop and implement Fourier methods (using the FFT) on the discrete motion group for the 2D case, and apply them to the calculation of the configuration space of a mobile robot and the problem of computing the workspaces of revolute manipulators. First, we review briefly these problems.

1.1 Computing the configuration space of a mobile robot among static obstacles

The question of computing the configuration space of a robot among static obstacles has been considered

[1] The group of translations and rotations of a rigid body, denoted as $SE(n)$.

in number of papers [8, 9, 10, 11]. Some of these algorithms [8] compute analytically the boundary of the regions of free configuration space for polygonal robots and obstacles. In this paper we develop and implement a method which builds on work of [11, 10], which may be applied to both polygonal and non-polygonal shapes. We suggest faster implementation of this method to compute the configuration space, which also potentially allows one to solve inverse problems, i. e. design of a robot that is capable of reaching specified positions and orientations in a given obstacle field.

The mathematical formulation of the method is given below. We compute a function on the configuration space (the space of translations and rotations of the robot) which has nonzero values only in the regions where the robot hits the obstacle. The magnitude of this "density function" is the ratio of the overlapping volume (area) of the robot to the total volume (area), i. e. it changes from 0 to 1 in the overlapping regions. To calculate this value we compute the integral

$$c(\mathbf{x}, A) = \frac{\int_{\mathbb{R}^n} f_1(\mathbf{y})\, f_2(A^{-1}(\mathbf{y}-\mathbf{x}))\, d^n y}{\int_{\mathbb{R}^n} f_2(\mathbf{y})\, d^n y} \quad (1)$$

where $f_{1,2}(\mathbf{x})$ are equal to 1 if the vector $\mathbf{x}$ is inside or on the boundary of obstacles (robot) and zero otherwise, $n = 2, 3$ for 2D (3D) coordinate space and $d^n y = dy_1 ... dy_n$ is the usual integration measure for $\mathbb{R}^n$. The function is normalized to 1, i. e. it is divided by the volume (area) of the robot, $\mathbf{x} \in \mathbb{R}^n$ and $A \in SO(n)$.

To compute this integral directly, or simply to check pixel by pixel that the obstacle $f_1(\mathbf{y})$ and the rotated and translated robot $f_2(A^{-1}(\mathbf{y}-\mathbf{x}))$ do not overlap, we need to perform $O(N N_r^2)$ computations, where N is the number of sampled orientations and N_r is the number of sampled points in a bounded region of $\mathbb{R}^n$. For a large 3D array of values the computation of this integral by direct summation may be quite slow. The "overlap function" $c(\mathbf{x}, A)$, however, may be computed in $O(N N_r \log N_r)$ computations using Fourier methods on the motion group and FFT methods. This implementation is addressed in sections 2-4. We discuss applications of this method to mobile robot configuration space generation in detail in section (5.1).

1.2 The workspace of discretely actuated manipulators

We also apply the Fourier transform on the discrete motion group to the workspace generation of planar discretely actuated manipulators (see [12, 20] for review of a particular kind of discrete manipulators - binary manipulators).

Discrete manipulators are robot arms with discrete-state actuators. The *workspace density*, defined as the number of reachable frames per unit volume of the Euclidean motion group $SE(n)$ [20], determines how accurately a position and orientation can be reached. The workspace density information is important for the kinematic design of manipulators and for planning the motions of discretely actuated manipulator arms [19].

To compute the workspace density function using brute force and iterating is computationally unrealistic for a large number of actuators m, because it requires K^m evaluations of the kinematic equations relating actuator state to the resulting end frame for a manipulator with m actuated modules each with K states. However, as has been shown in [21, 22], partitioning the manipulator into segments, computing the workspace density function for each segment, and computing the workspace density function of the whole manipulator as a convolution of those for each segment is an efficient way to solve this problem. Using the convolution approach allows us to reduce the exponential growth in m to linear growth in m for inequivalent segments and to $O(\log m)$ for identical segments.

The workspace density of a manipulator which has double the number of identical modules of one with workspace density $f(g)$ may be generated as a convolution on the motion group

$$F(g) = (f * f)(g) = \int_{SE(n)} f(h)\, f(h^{-1} \circ g) d\mu(h) \ , \quad (2)$$

where $d\mu(h)$ is a left and right invariant integration measure on $SE(n)$, i. e. $d\mu(g \circ h) = d\mu(h \circ g) = d\mu(g)$. Throughout this paper we will interchange notations for this convolution. For instance, any function on the Euclidean motion group $SE(n)$ can be written as $f(g) = f(\mathbf{x}, A)$, where $\mathbf{x} \in \mathbb{R}^n$ and $A \in SO(n)$. Using

this notation, the above convolution integral is written as

$$F(\mathbf{x}, A) = \int_{SO(n)} \int_{\mathbb{R}^n} f(\mathbf{y}, R)\, f(R^{-1}(\mathbf{x}-\mathbf{y}), R^{-1}A)\, d^n y\, dR$$

where dR is the normalized left-right invariant integration measure on $SO(n)$. While the idea of such integration measures is relatively new in the robotics literature [3, 4, 1], such measures have been known in the mathematical physics literature for decades [13], and so we do not discuss them here.

In this paper we concentrate on the planar case, where this convolution integral is written in its most concrete form as

$$F(x_1, x_2; \theta) = \frac{1}{(2\pi)^2} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \int_{-\pi}^{\pi} f(y_1, y_2; \alpha)$$
$$f((x_1 - y_1)\cos\alpha + (x_2 - y_2)\sin\alpha,$$
$$-(x_1 - y_1)\sin\alpha + (x_2 - y_2)\cos\alpha; \theta - \alpha)\, dy_1\, dy_2\, d\theta$$

This integral may be computed efficiently using the Fourier transform on the motion group (in analogy with the use of the usual Fourier transform to compute convolutions of functions on the real line). This problem is addressed in sections (2-4,5.2). The efficiency of the Fourier transform is discussed in section 4. Moreover, using Fourier methods allow us to solve inverse problems in manipulator design [29, 31]. But first some mathematical background is required.

2 Convolution on Groups,Unitary Representations and the Fourier Transform

Recall that a *group* [23, 24, 25] is a set G together with an operation $\circ$ which takes two elements of the set and returns an element of the set. All elements $g, g_1, g_2, g_3 \in G$ must satisfy the following properties:

- $g_1 \circ (g_2 \circ g_3) = (g_1 \circ g_2) \circ g_3$.
- There exists an element $e \in G$ such that $e \circ g = g \circ e = g$.
- For every element $g \in G$ there is an element $g^{-1} \in G$ such that $g^{-1} \circ g = g \circ g^{-1} = e$.

In order to distinguish a group from the set, the former is denoted $(G, \circ)$, unless the operation is understood from the context, in which case G refers to both the set and the group.

The group G is called *unimodular* if a measure μ exists such that functions on the group can be integrated,

$$\mu(f) = \int_G f(g) d\mu(g),$$

and $\mu(R(h)f) = \mu(L(h)f) = \mu(f)$ where $R(h)f(g) = f(g \circ h)$ and $L(h)f(g) = f(h^{-1} \circ g)$ are respectively right and left shifted versions of the function $f(g)$. It is well known that all finite groups [2] and all compact Lie groups are unimodular, as are the groups considered in this paper.

Given two real-valued functions with group-valued argument, one defines the convolution of these functions as:

$$(f_1 * f_2)(g) = \int_G f_1(h) f_2(h^{-1} \circ g) d\mu(h), \qquad (3)$$

where the integral is taken over the group with respect to the measure μ.

The useful concept of a Fourier transform also extends to "well-behaved" functions on unimodular groups. In the case of the Fourier transform on the real line, the function $u(x, p) = e^{ipx}$ plays a very important role. An analogous function is needed for a unimodular group G so that the Fourier transform pair can be defined for this case. However, since G is usually not commutative, a scalar function $u(g, p)$ is not sufficient to reflect its noncommutative nature. Instead of a unitary scalar function, we must look at a unitary matrix $U(g, p)$ with elements $u_{mn}(g, p)$. The variable p is called the *dual* variable, and may be a scalar or other quantity. These matrices (called irreducible unitary representations of the group) must have the homomorphism property

$$U(g_1 \circ g_2, p) = U(g_1, p) U(g_2, p)$$

[2] In the special case of finite groups the integral is viewed as a summation.

for all $g_1, g_2 \in G$ and a enumerates distinct irreducible unitary representations (IURs) of G. A number of works including [13] have shown that the elements of the unitary matrices $U(g,p)$ can be defined in terms of the standard functions of mathematical physics.

In particular, for the case of $SE(2)$ the elements of the infinite dimensional matrix $U(g,p)$ are

$$u_{mn}(g(r,\phi,\theta),p) = i^{n-m} e^{-i[n\theta+(m-n)\phi]} J_{n-m}(pr) \quad (4)$$

$p \in R^+$, the group elements are expressed in polar coordinates:

$$g = g(r,\phi,\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & r\cos\phi \\ \sin\theta & \cos\theta & r\sin\phi \\ 0 & 0 & 1 \end{pmatrix}$$

and $J_\nu(x)$ is the νth order Bessel function, ϕ is a polar angle of $\mathbf{r}$, and θ is $SO(2)$ orientation angle.

The Fourier transform of a suitable function $f(g)$, and the inverse transform are defined as:

$$\mathcal{F}(f) = \hat{f}(p) = \int_G f(g) U(g^{-1},p) d\mu(g)$$

and

$$f(g) = \mathcal{F}^{-1}(\hat{f}(p)) = \int_{\hat{G}} \mathrm{trace}(\hat{f}(p) U(g,p)) d\nu(p)$$

where $d\nu(p)$ is an appropriately defined measure on the dual space of G.

One of the most powerful properties of the Fourier transform of functions on $\mathbb{R}^N$ is that the Fourier transform of the convolution of two functions is the product of the Fourier transforms of the functions. This property extends to the concept of a Fourier transform for functions on G. Namely, we have that for functions $f_1(g)$ and $f_2(g)$ which decay rapidly to zero outside of a compact subset of G that

$$\mathcal{F}(f_1 * f_2) = \mathcal{F}(f_2)\mathcal{F}(f_1),$$

where the convolution is defined as in Equation 3. The order of the product of these Fourier transform matrices is important since G is generally not commutative.

3 The Discrete Motion Group

The "discrete motion group" is the group $G_N = \mathbb{R}^2 \triangleright C_N$, where C_N is the N-element finite subgroup of $SO(2)$ (i.e. the group of rotational symmetries of a regular planar N-gon), $\mathbb{R}^2$ is a continuous group of translations (yet we use the "discrete motion group" terminology for G_N) and the notation $\triangleright$ means semi-direct product (which in the present context means nothing more than that group elements are expressed as homogeneous transforms). The continuous motion group $SE(2) = \mathbb{R}^2 \triangleright SO(2)$ is considered in [13, 14, 15, 18, 29]. Fourier analysis and FFTs for finite groups are considered in [17]. The Fourier Transform for G_N resembles both those of the continuous and finite groups.

The matrix elements of the irreducible unitary representations of the discrete motion group $G_N = \mathbb{R}^2 \triangleright C_N$ are given as [28, 30]

$$U_{mn}(A_j, \mathbf{r}; p, \phi) = e^{i p \mathbf{u}_m^\phi \cdot \mathbf{r}} \delta_{A_j^{-1}\mathbf{u}_m, \mathbf{u}_n} \quad (5)$$

where $\mathbf{u}_k^\phi$ denotes the vector to the angle ϕ on the unit circle on the interval $F_k = [2\pi k/N, 2\pi(k+1)/N]$, $k = 0, ..., N-1$ (ϕ measures the angle on this segment), $\mathbf{u}_m = \mathbf{u}_m^0$. The vector $\mathbf{u}_m^\phi$, which is inside the segment F_m may be received by the rotation A_m (which transform F_0 to F_m) from $\mathbf{u}_0^\phi$, $\mathbf{u}_m^\phi = A_m \mathbf{u}_0^\phi$. The parameter ϕ denotes the position inside the segment F_0. The vectors $\mathbf{u}_m^\phi$ are illustrated in Fig. (1). The function $\delta_{A_j^{-1}\mathbf{u}_m, \mathbf{u}_n}$ is the Kronecker delta function for vector-valued indices

$$\delta_{A_j^{-1}\mathbf{u}_m, \mathbf{u}_n} = \begin{cases} 1, & \text{if } A_j^{-1}\mathbf{u}_m = \mathbf{u}_n \\ 0 & \text{otherwise} \end{cases}$$

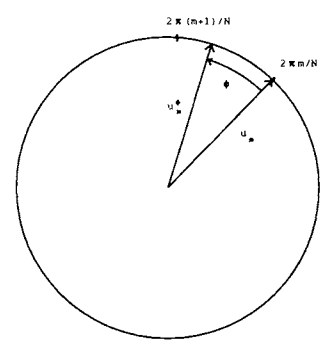


Figure 1

The direct Fourier transform is defined as

$$\hat{f}(p,\phi) = \sum_{i=0}^{N-1} \int_{\mathbb{R}^2} f(A_i,\mathbf{r})\, U^{-1}(A_i,\mathbf{r};p,\phi)\, d^2r. \quad (6)$$

The inverse Fourier transform is [28, 30]

$$\mathcal{F}^{-1}(\hat{f}) = \frac{1}{4\pi^2} \int_0^{\infty} \int_0^{2\pi/N} \mathrm{Tr}\left(\hat{f}(p,\phi)\, U(A_i,\mathbf{r};p,\phi)\right) \times$$

$$p\, dp\, d\phi\ . \quad (7)$$

We define a convolution on the discrete motion group as

$$F(\mathbf{r},A_j) = \frac{1}{2\pi N} \sum_{i=0}^{N-1} \int_{\mathbb{R}^2} f_1(\mathbf{a},A_i) \times$$

$$f_2(A_i^{-1}(\mathbf{r}-\mathbf{a}), A_{j-i}) d^2a\ . \quad (8)$$

The normalization factor corresponds in the limit $N \to \infty$ to the normalization of the convolution on the continuous motion group $SE(2)$ presented earlier.

The Fourier transform of the convolution of functions on the discrete motion group is just the product of the Fourier matrices with the corresponding normalization

$$\hat{F}^{\phi}_{mn}(p) = \frac{A}{2\pi N} \sum_{k=0}^{N-1} (\hat{f}_2)^{\phi}_{mk}(p)\, (\hat{f}_1)^{\phi}_{kn}(p) \quad (9)$$

where A is the area of a compact region of $\mathbb{R}^2 \ni \mathbf{r}$ on which the functions have support. The area factor arises because the discrete Fourier transform may be received as an approximation of the continuous case using

$$r_1 \to \frac{L}{N_x} i; \quad p_1 \to \frac{2\pi}{L} i\ ,$$

(and analogously for the 2nd component). Here L is the length of the compact region in the x (y) direction. While the factor L is cancelled out of equations if we apply direct and inverse discrete Fourier transform to the function, it appears in the convolution, defined in (8).

4 Efficiency of Computation of Convolution Integrals Using the Fourier Transform

In this section we show that using the Fourier transform on the discrete motion group is a fast method to compute convolution integrals on the discrete motion group (we assume below that the finite rotation group has N elements). Particularly, we show that the convolution of a function $f(\mathbf{r},A_i)$ sampled at $N_g = N \cdot N_r$ points ($N_r = N_x \cdot N_y$ is the number of samples in an $\mathbb{R}^2$ region) may be performed in $O(N_g \log N_r) + O(N_g N)$ operations instead of $O(N_g^2)$ computations in the direct coordinate space integration using the "plain" integration in (8). The structure of matrix elements of (5) allows one to apply Fast Fourier Methods and reduce the amount of computations (without the application of FFT the amount of computations using the Fourier transform method is $O(N_g^2/N)$).

First, we estimate the amount of computations to perform the direct and inverse Fourier transforms of $f(g)$. We assume that we restrict the p values to a finite interval (what is analogous to considering the band-limited approximation to function) and sample it at N_p points, and sample the ϕ values at N_ϕ points. We also assume that the total number of harmonics $N_p N_\phi N^2 = N_g = N N_r$ (we assume the relation $N_p N_\phi N = N_r$ to have the number of samples on polar grid equal the number of samples N_r on the Cartesian grid).

Let us consider the direct Fourier transform (6). Each term i (for fixed A_i) gives one nonzero term in each row and column of the Fourier matrix $\hat{f}^{\phi}_{mn}(p)$ (because only one element in each row and column of $U^{-1}_{mn}(g;p,\phi)$ is nonzero). For each fixed i we may compute the usual FFT of $f(\mathbf{r};A_i)$, which may be computed in $O(N_r \log(N_r))$ operations. The Fourier transform elements received by FFT are computed on a square grid of $\mathbf{p}$ values. We, however, may interpolate the Fourier elements computed on the grid to the Fourier elements computed at polar coordinates. The radial part p is determined by the length of $\mathbf{p}$, the angular part determines the indices m, ϕ (other index n is determined uniquely for given A_i). We bi-linearly interpolate values on a square grid to values on a polar

grid (more complicated interpolation technique [26], or two-dimensional spline interpolation [27] may be used in order to reach better accuracy). Such an interpolation may be performed in $O(N_r)$ computations. Each term i in (6) may be computed in $O(N_r \log(N_r))$ computations. The whole Fourier matrix may be received in $O(N\, N_r \log(N_r))$ computations.

Again, one element from each row and column is used in computation of the inverse Fourier transform for each rotation element A_i. After inverse interpolation to Cartesian coordinates (which may be done in $O(N_r)$ computations), the inverse Fourier integration may be performed in $O(N_r \log(N_r))$ for each of the N nonzero matrix elements of U using the FFT. Thus, in $O(N\, N_r \log(N_r))$ computations we reproduce the function for each A_i.

The matrix product of $\hat{f}^{\phi}_{mn}(p)$ may be computed directly in $O(N^3)$ computations [3] for each value of p and parameters ϕ. This means that the convolution (which is a matrix product of Fourier matrices) may be performed in $O(N^3 N_p\, N_\phi) = O(N N_g)$ computations.

Therefore, the convolution of functions on the discrete motion group may be performed in $O(N_g \log N_r) + O(N_g N)$ using Fourier methods on the discrete motion group and the usual FFT, without assuming any special matrix multiplication technique.

It may be shown that without the application of the FFT the convolution may be performed using the Fourier transform in $O(N_g^2/N)$, which is still faster than evaluating the convolution by direct integration.

5 Numerical Examples

In this section three example applications of these techniques in robotics are demonstrated. In Section 5.1 the configuration space of a mobile robot is calculated. In Section 5.2 the workspace density of an m-link revolute manipulator is calculated.

[3]Estimates as fast as $O(N^{2.38})$ have been reported [16, 17], which, depending on how they are implemented, have the potential to increase speed further.

5.1 Configuration space of a mobile robot among static obstacles

Here we apply the numerically implemented Fourier transform on the discrete motion group of the plane to the computation of the configuration space of a rigid mobile robot moving among static 2D obstacles. To find the configuration space we compute the integral (1) using the above described Fourier methods for the discrete motion group. Because the obstacles and the robot are functions only of Cartesian position (i. e. to find (1) we need to compute the Fourier transform of $f_2(\mathbf{x})$ only for the fixed orientation) the direct Fourier transform is performed faster than for an arbitrary function on the motion group.

A function of position may be considered as a function on the discrete motion group which does not depend on the orientation, i. e. $f(\mathbf{x}, A_i) = f(\mathbf{x})$. Then the "overlap" function (1) may be formally written as the integral

$$c(\mathbf{x}, A_j) = \frac{1}{N} \frac{\sum_{i=0}^{N-1} \int_{\mathbb{R}^2} f_1(\mathbf{y}, A_i)\, f_2(A_j^{-1}(\mathbf{y}-\mathbf{x}), A_j^{-1} A_i)\, d^2y}{\int_{\mathbb{R}^2} f_2(\mathbf{y})\, d^2y} \tag{10}$$

Because functions $f_{1,2}(\mathbf{x}, A_i) = f_{1,2}(\mathbf{x})$ do not depend on the orientations A_i, matrix elements in the same column are the same, i. e.

$$(\hat{f}_{1,2})_{mn} = (\hat{f}_{1,2})_{qn}$$

for any m, q. This may be observed from the expression

$$U(g^{-1}; p, \phi)_{mn} = e^{-i\,p\,\mathbf{u}_n^{\phi}\cdot\mathbf{r}}\, \delta_{A_i^{-1}\mathbf{u}_n, \mathbf{u}_m}$$

(the exponent depends only on the n-index), the definition of the direct transform (6), and the fact that the functions do not depend on the orientation. Thus, we compute a row of the Fourier matrix for a particular orientation (for example $A_0 = \mathbf{1}$, the identity element), and so we define

$$(\hat{f}_{1,2})_n = (\hat{f}_{1,2})_{nn}\ .$$

This may be done using the 2D FFT for the functions $f_{1,2}(\mathbf{x})$ and interpolating the Fourier values to points on a polar coordinate grid.

The "overlap" function (1) is written as

$$c(\mathbf{x}, A_j) =$$

$$C \frac{\sum_q \sum_n \int_0^\infty \int_\phi \left(\overline{\hat{f}_{2q}}\, \hat{f}_{1n}\right) U_{nq}(g; p, \phi)\, p\, dp\, d\phi}{\int_{\mathbb{R}^2} f_2(\mathbf{y})\, d^2y} \quad (11)$$

where $C = 1$. The product of the column $\overline{\hat{f}_2}$ and the row $\hat{f}_1$ may be performed in $O(N_g = N_r N)$ computations (instead of $O(N_g N)$ for orientation-dependent functions), and the inverse transform may be performed in $(N_g \log(N_r))$ computations. The normalization of the function $f_2(\mathbf{x})$ may be computed by direct integration in $O(N_r)$. Thus, the inverse transform is the largest time-consuming computation. We note that the direct Fourier transform is performed only for one orientation. This is N times faster (for discrete rotation group C_N) than to perform the FFT for each orientation as is done in [11], but the inverse transform is of the same order of computations. Thus, as N becomes large, there is a built-in factor of two speed increase using our method. We also mention that inverse problems, i. e. problems of finding the allowed shape of the robot for the given desired configuration space and the shape of the obstacles, may be solved using Fourier methods on the motion group, because it is reduced to a functional matrix equation (regularization methods for the solution of singular functional matrix equations are discussed in [31]).

When we discretize the coordinate regions and use the FFT to compute (1), the coefficient C in (11) must be defined as $C = A$, where A is the area of the compact region of $\mathbf{x}$ where the FFT is computed (the functions must have a support inside the area A, the functions are considered periodic outside of this region). The area factor arises because the discrete Fourier transform may be received as an approximation of the continuous case using

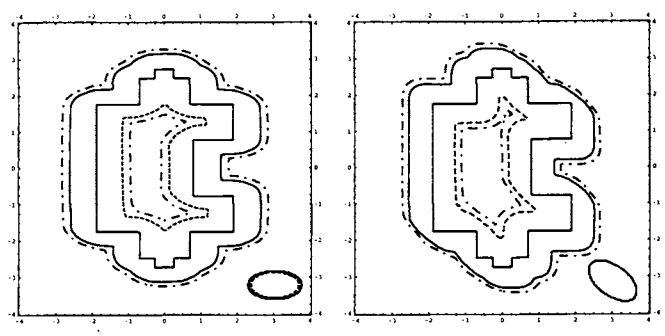
$$x_1 \to \frac{L}{N_x} i; \quad p_1 \to \frac{2\pi}{L} i\ ,$$

(and analogously for the 2nd component). Here L is the length of the compact region in the x (y) direction. While the factor L is cancelled out of equations if we apply direct and inverse discrete Fourier transform to the function, it appears in the convolution-like integral (1).

Thus, we compute the 2D FFT of $f_{1,2}(\mathbf{x})$, interpolate the Fourier elements to the polar grid, arrange them into the Fourier column and row, multiply column and row $\hat{f}_{mn} = \overline{\hat{f}_{2m}}\, \hat{f}_{1n}$, and take the inverse Fourier transform (the corresponding elements $m = n - i$ from the Fourier matrix $\hat{f}_{mn}$ must be taken for each orientation A_i and interpolated back to the Cartesian grid to take the 2D inverse FFT).

We implemented the computation of (11) using the FFT in the C programming language. Time to compute $c(\mathbf{x}, A_j)$ was 30 sec (on a 250 MHz SGI workstation) for a 256×256 square grid in $\mathbb{R}^2$ for the C_{10} group ($N = 10$, and we subdivide each segment into $N_\phi = 20$ subsegments).

Because for small values of $c(\mathbf{x}, A_j)$ the function exhibits oscillations (due to finite discretization of the integration area in the Fourier transform), we depict the boundary of the configuration space, defined by the contour line where $c(\mathbf{x}, A_j) = \epsilon$. The smallest possible choice of ϵ was in our examples in the region $0.005 - 0.035$.



a) b)

Figure 2

We note that this method may not give good results if a very narrow and long object is attached to the robot (i.e. when the intersection area is not sensitive to the overlap). However, the value of $f_1(\mathbf{x})$ may be increased in these regions and this method may be useful for some of these shapes.

As an example we depict slices of configuration space in Fig. (2a, 2b). The obstacle has a polygonal shape

(the Fourier method may be applied both for polygonal and for "smooth" obstacles. The results are better for "smoother" obstacle, i. e. the value of ϵ may be chosen smaller), the robot has an elliptic shape (the size and the orientation of the robot are depicted in the lower right corner on the figures).

The solid line outside the obstacle depicts the boundary of the configuration space ($\epsilon \approx 0.030 - 0.035$). We increase the value of the function f_1 to 10 in the rim depicted in Fig.(2a) (between the original boundary and the boundary scaled down by factor $k = 1.1$). We also depict in the pictures the boundary (dashed line) which describes the configuration space of the robot when it is located completely inside of the obstacle ($c(\mathbf{x}, A) \approx 0.95 - 0.97$).

Figures depict also the corresponding boundaries for the scaled robot (enlarged by the factor 1.27). This is region where the exact position of the configuration space boundary may be found by direct integration.

5.2 The workspace density of discretely actuated manipulators

First, we discuss briefly implementation of the Fourier transform of functions on the discrete motion group.

The matrix elements of the IURs in (5) may be written as

$$U_{mn}(A_j, \mathbf{r}; p, \phi) = e^{i\,p\,\mathbf{u}_m^{\phi}\cdot\mathbf{r}}\,\delta_{m-j,n}$$

where $\delta_{i,j}$ is the usual Kroneker delta function and $\mathbf{u}_m^{\phi}\cdot\mathbf{r} = r\cos(2\pi/N\,m + 2\pi/(NN_\phi)\,\phi - \theta)$, where $m = 0, ..., N-1$, and discrete values of ϕ are $\phi = 0, ..., N_\phi - 1$ (N_ϕ is a number of sample points on the interval $[2\pi/Nm, 2\pi/N\,(m+1)]$), and θ is the polar angle of $\mathbf{r}$.

The direct Fourier transform is given by (6). Each term in the sum $i = 0, ..., N-1$ through the elements of the rotation group C_N gives one element of the Fourier matrix in each row and each column, summing through all $i = 0, ..., N-1$ gives the whole matrix. For each term i in (6) we restrict integration with respect to $\mathbf{r}$ to a compact region, sampled at the square grid $N_r^{1/2} \times N_r^{1/2}$.

Because we want to use the advantages of the Fast Fourier Transform (FFT), we compute the two dimensional FFT (for each i) and interpolate the value of the Fourier transform on the square grid $N_r^{1/2} \times N_r^{1/2}$ of $\mathbf{p}$ to the polar coordinates. The radial coordinate is $p = |\mathbf{p}|$, the polar angles is determined by the values of m and ϕ (the value of n is determined from m and the index of rotation i). Thus, we have to compute N FFTs (for the C_N rotation group) to obtain the whole Fourier matrix.

In the inverse Fourier transform we have to interpolate back the Fourier parameter p from polar coordinate to the values of $\mathbf{p}$ on the square grid (we take one element from each row and column of the Fourier matrix for each rotation element i) and compute the two dimensional inverse FFT for each rotation i. Again, the whole function $f(g)$ is reproduced in N FFTs.

The Fourier transform of convolution (8) is defined by (9).

In this subsection we apply the Fourier method to generate the workspace density of the discretely actuated m-link planar revolute manipulator, shown in Fig. (3).

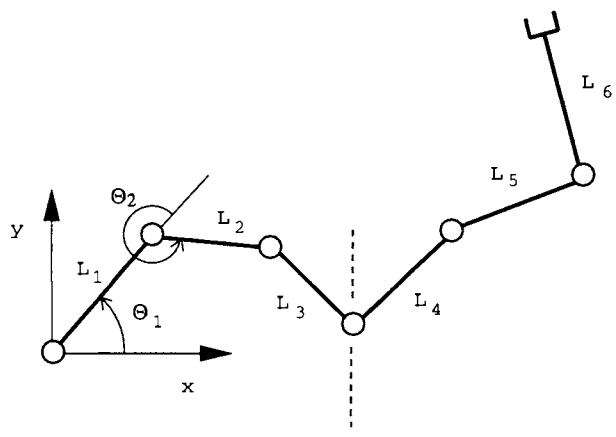


Figure 3

We assume that each link may reach a discrete angle θ_i relative to the other link, and the length of each link L_i has a fixed value. In our example we want to generate the workspace of a 6-link manipulator, assuming that we know the workspace of each 3-link half manipulator.

We note that the position and orientation of the end-effector for the discretely actuated planar revolute manipulator (each link has N possible equally spaced discrete angle values) has values from the discrete motion group $G = \mathbb{R}^2 \triangleright C_N$, where C_N is the N-element finite subgroup of $SO(2)$. For a small number of links, m, the workspace density defined as number of end-effector position per unit volume of the discrete motion group, may be computed by brute force, because it requires N^m computations. We assume below that each link may reach $N = 60$ possible angles relative to the other link. The grey level workspace density values for 3-link manipulator parameters

$$L_1 = 0.2; \quad L_2 = 0.3; \quad L_3 = 0.3 \ . \tag{12}$$

are depicted in Fig. (4a) and (4b) for $\theta = 0$ and $\theta = \pi/2$ end-effector orientation angles. The boundary of the workspace (which coincides with the boundary of workspace for the continuous manipulator, in which case the analytical solution may be easily found) may be also clearly observed.

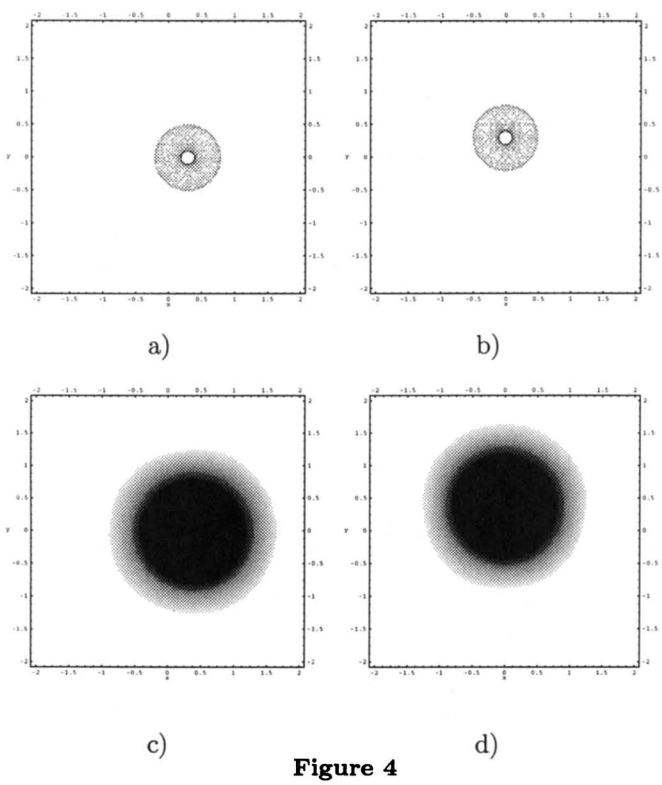


Figure 4

For a large number of links, m, the number of brute force computations increases exponentially. We may, however, partition the manipulator into segments and convolve the workspace densities of these segments. As an example we received the workspace density of the 6-link manipulator convolving the workspace density of the 3-link manipulator with parameters (12) (for lower part) and 3-link upper half with the parameters

$$L_4 = 0.2; \quad L_5 = 0.3; \quad L_6 = 0.4 \ .$$

We computed the workspace on a $256 \times 256 \times 60$ grid using the Fourier method on the discrete motion group (with C_{60} rotation subgroup). The whole computation takes 15 min on a 250 MHz workstation. The convolved workspace density is depicted in Fig. (4c,d) for the orientation angles $\theta = 0$ and $\theta = \pi/2$. The boundary of the workspace also may be observed.

6 Conclusions

This paper develops Fourier analysis on the discrete motion group and applies the resulting theory to problems in robot kinematics and motion planning. These problems include the fast calculation of mobile robot configuration spaces, and the workspace density functions of manipulator arms with finite state actuators. All of the problems presented here can be formulated as Euclidean group convolutions. Using the discrete motion group as an approximation to the Euclidean group allows us to apply FFT techniques to calculate these generalized convolutions in a very efficient way.

References

[1] Basavaraj, U., Duffy, J., "End-Effector Motion Capabilities of Serial Manipulators," *International Journal of Robotics Research*, Vol. 12, No. 2, April 1993, pp. 132-145.

[2] McCarthy, J.M., *An Introduction to Theoretical Kinematics*, MIT Press, Cambridge, Mass., 1991.

[3] Murray, R. M., Li, Z., Sastry, S.S., *A Mathematical Introduction to Robotic Manipulation*, CRC Press, Ann Arbor MI, 1994.

[4] Park, F.C., Brockett, R.W., *Kinematic Dexterity of Robotic Mechanisms*, The International Journal of Robotics Research, Vol. 13, No. 1, February 1994, pp 1-15.

[5] Janssen, T., *Crystallographic Groups*, North-Holland, New York, 1973.

[6] J. W. Cooley and J. Tukey, *An algorithm for the machine calculation of complex Fourier series*, Math. of Comput. **19** (1965) pp. 297-301.

[7] D. F. Elliott, K. R. Rao, *Fast Transforms: Algorithms, Analyses, Applications*, Academic Press, New York, London, 1982.

[8] T. Lozano-Perez, *Spatial Planning: A Configuration Space Approach*, IEEE Trans. on Computers, **32** (1983), pp. 108-120.

[9] W. Newman, M. Branicky, *Real-Time Configuration Space Transforms for Obstacle Avoidance*, Int. J. of Robotics Research, **10** (1991) pp. 650-667.

[10] L. Guibas, L. Ramshaw, J. Stolfi, *A Kinematic Framework for Computational Geometry*, Proc. of IEEE Symp. on Found. of Computer Science, 1983, pp. 100-111.

[11] L. Kavraki, *Computation of Configuration-Space Obstacles Using the Fast Fourier Transform*, IEEE Trans. on Robotics and Automation, **11** (1995), pp. 408-413.

[12] G. S. Chirikjian, *Kinematic Synthesis of Mechanisms and Robotic Manipulators with Binary Actuators*, J. Mech. Design **117** (1995) 573-580.

[13] N. J. Vilenkin, *Bessel Functions and Representations of the Group of Euclidean Motions*, Uspehi Mat. Nauk. **11** (1956), 69-112 (in Russian).

[14] A. Orihara: Bessel Functions and the Euclidean Motion Group, *Tohoku Math. J.* **13** (1961), 66-71.

[15] J. Talman: *Special Functions*, W. A. Benjamin, Inc., Amsterdam, 1968.

[16] D. Coppersmith and S. Winograd: *Matrix multiplication via arithmetic progressions*, Proceedings of the 19th ACM Symposium on the Theory of Computing (1987), New York, N. Y., pp. 1-6.

[17] D. N. Rockmore: *Efficient Computation of Fourier Inversion for Finite Groups*, Journal Assoc. for Comp. Machinery, **41** (1994) pp.31-66.

[18] G. Chirikjian: Fredholm Integral Equations on the Euclidean Motion Group, *Inverse Problems* **12** (1996), 579-599.

[19] I. Ebert-Uphoff, G. S. Chirikjian, *Inverse Kinematics of Discretely Actuated Hyper-Redundant Manipulators Using Workspace Density*, Proceedings 1996 IEEE International Conference on Robotics and Automation, Minneapolis, MN April 1996, pp. 139-145.

[20] I. Ebert-Uphoff, G. S. Chirikjian, *Efficient Workspace Generation for Binary Manipulators with Many Actuators*, J. of Robotic Systems, **12(6)** (1995) 383-400.

[21] I. Ebert-Uphoff, G. S. Chirikjian, *Discretely Actuated Manipulator Workspace Generation by Closed-Form Convolution*, Proceeding 1996 ASME Design Engineering Technical Conference and Computers in Engineering Conference, Irvine, California, August 1996.

[22] G. S. Chirikjian, I. Ebert-Uphoff, *Numerical Convolution on the Euclidean Group with Applications to Workspace Generation*, scheduled to appear in IEEE Trans. Robotics and Automation, 2/1998.

[23] M. Sugiura, *Unitary Representations and Harmonic Analysis*, 2nd edition, Elsevier Science Publisher, The Netherlands, 1990.

[24] D. Gurarie, *Symmetry and Laplacians. Introduction to Harmonic Analysis, Group Representations and Applications*, Elsevier Science Publisher, The Netherlands, 1992.

[25] A. Barut, R. Raczka, *Theory of Group Representations and Applications*, World Scientific, Singapore, 1986.

[26] H. Stark, J. W. Woods, I. Paul, and R. Hingorani, *Direct Fourier reconstruction in computer tomography*, IEEE Trans. Acoust. Speech Signal Processing, **29**, (1981), pp. 237-244.

[27] H. Speath, *Two Dimensional Spline Interpolation Algorithms*, Wellesley, Massachusetts, AK Peters, 1995.

[28] J. P. Gauthier, G. Bornard and M. Sibermann, *Motion and Pattern Analysis: Harmonic Analysis on Motion Groups and Their Homogeneous Spaces*, IEEE Trans. Syst. Man Cybern., **21** (1991) pp. 159-172.

[29] A. Kyatkin, G. Chirikjian, *Numerical Synthesis of Binary Manipulator Workspaces Using the Fourier Transform on the Euclidean Motion Group*, Tech. Report RMS-97-1, Dept. of Mechanical Engineering, JHU (1997).

[30] A. Kyatkin, G. Chirikjian, *Computation of robot configurations and workspaces via the Fourier transform on the discrete motion group*, Tech. Report RMS-97-2, Dept. of Mechanical Engineering, JHU (1997).

[31] A. Kyatkin, G. Chirikjian, *Regularized solutions of a nonlinear convolution equation on the Euclidean group*, to appear in Acta Appl. Math. (1998).

Taylor & Francis
Taylor & Francis Group
http://taylorandfrancis.com

The Self-Reconfiguring Robotic Molecule: Design and Control Algorithms

Keith Kotay, *Dartmouth College, Hanover, NH, USA*
Daniela Rus, *Dartmouth College, Hanover, NH, USA*
Marsette Vona, *Dartmouth College, Hanover, NH, USA*
Craig McGray, *Dartmouth College, Hanover, NH, USA*

We discuss a robotic module called a Molecule. *Molecules can be the basis for building self-reconfiguring robots. They support multiple modalities of locomotion and manipulation. We describe the design, functionality, and control of the Molecule. We show how a set of Molecules can aggregate as active three-dimensional structures that can move and change shape. We show an efficient algorithm for planning the trajectory of Molecules for self-reconfiguration. Finally, we discuss our Molecule experiments.*

1 Introduction

Our vision is to create versatile robots by using self-reconfiguration: hundreds of small modules will autonomously organize and reorganize as geometric structures to best fit the terrain on which the robot has to move, the shape of the object the robot has to manipulate, or the sensing needs for the given task. Large collections of small robots will actively organize as the most optimal geometric structure to perform useful coordinated work.

A self-reconfiguring robot consists of a set of identical modules that can dynamically and autonomously reconfigure in a variety of shapes, to best fit the terrain, environment, and task. Self-reconfiguration leads to versatile robots that can support multiple modalities of locomotion and manipulation. For example, a self-reconfiguring robot can aggregate as a snake to traverse a tunnel and then reconfigure as a six-legged robot to traverse rough terrain, such as a Lunar surface, and change shape and gait again to climb stairs and enter a building.

We build on the ground-breaking work of [14, 17, 12, 5] who introduced the first robot systems capable of self-reconfiguration. We have designed a small and simple robotic module we call a *Molecule* capable of self-reconfiguration in three dimensions. The Molecule (see Figure 1) is capable of independent movement on a substrate of identical Molecules, including straight-line traversal and 90 degree convex and concave transitions to adjacent surfaces. In this paper we describe the motion control for these operations and we show a geometric approach to specifying and planning molecular motion on a substrate of Molecules in $O(n)$ time, where n is the number of Molecules in the substrate.

The Molecule is a 4 degree-of-freedom, small-scale module capable of aggregating with other identical modules to form three-dimensional dynamic structures. The Molecule consists of two *atoms* connected by a right-angle rigid bond (see Figure 2.) Each atom has 5 inter-Molecule connectors and two degrees of freedom. One degree of freedom allows rotation about one connector. The second degree-of-freedom allows rotation of the atom about the bond. Detailed descriptions of the design and control of the Molecule can be found in [8].

A set of such Molecules can self-aggregate as arbitrary three-dimensional structures. We show this result by demonstrating several different tilings of the plane with the Molecule. The tilings can be stacked so that the resulting structures are three-dimensional. A structure made of Molecules can self-reconfigure by using the basic Molecule motions. For example, a structure made of Molecules can climb a set of stairs by self-reconfiguration as follows (see Figure 5). Linear motions are used to place the structure at the bottom of the staircase. Concave transitions and linear motions enable Molecules to climb on top of each other, composing a tower whose height equals the height of the

step. The tower can then be used by other Molecules to climb up and move onto the step. This process is then repeated for each step.

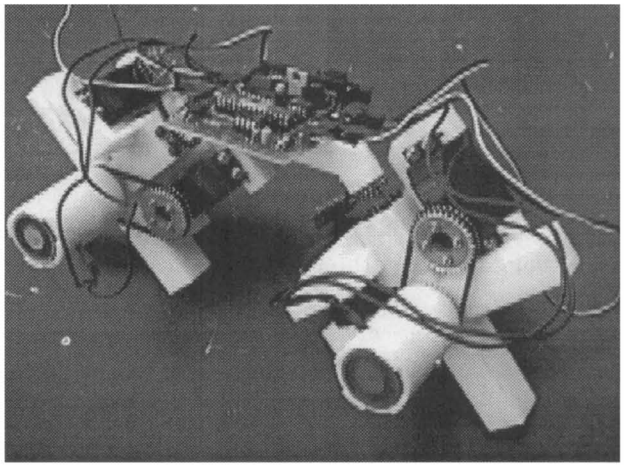


Figure 1: *The robotic Molecule. The Molecule is composed of two atoms, connected by an right-angle rigid bond. The Molecule has 4 degrees of freedom: two rotational degrees of freedom about the bond and one rotational degree of freedom per atom about a single inter-Molecule connector. The connectors have been implemented with electromagnets.*

In our lab we have prorotyped a Molecule and experimented with its basic motion capabilities. The Molecule performed linear motion and transitions. These experiments validated our design. We plan to build 16 modules to test our theory of self-reconfiguration.

This paper is organized as follows. We continue with a summary of related work. We then describe the mechanical design of the Molecule and the fabrication process. We address how this Molecule can form arbitrary three dimensional structures. Finally, we show a geometric approach to computing the motion of individual Molecules on a substrate of n other Molecules in $O(n)$ time, by constructing a reachability graph and searching this graph for a path. This graph can be used to extract a motion plan in the Molecule control space.

2 Related Work

Our work draws on previous experiences with navigation algorithms [10], designing self-organizing robots, and designing minimalist robot systems [4].

Related work in designing modular robots includes [15, 16, 13, 6]. In [15] a method for designing various robotic arms with different reachability properties out of the same set of 7 modules is proposed. The mechanical design algorithm is implemented as simulated annealing that starts with a random mechanical design and converges to the design with desired reachability properties. The modules are assembled by hand as the computed shape. Our work is different in that our modules could self-aggregate (without human intervention) and the planning phase is of a task-directed, geometric nature. Our modular self-reconfiguring robots are aggregated according to task and we can view reachability as a specific kind of task.

Related work in self-organizing robots includes robots in which modules are reconfigurable using external intervention [3]. In [5] a cellular robotic system is proposed to coordinate a set of specialized modules. [1] describe a theoretical framework for counting the number of unique configurations realizable from a set of modules and joints, without considering implementation issues. [17] studies multiple modes of locomotion that are achieved by composing a few basic elements in different ways. [12, 18] consider a system of modules that can achieve planar motion by walking over each other due to changes in the polarity of magnetic fields. [14] describes metamorphic robots that can aggregate as stationary two-dimensional structures with varying geometry and that implement planar locomotion.

Our self-reconfigurable robots [8, 7] are different from metamorphic robots in their design and functionality. The structures built from self-organizing Molecules and Inchworms are three dimensional, can move along any axis in a three dimensional space, and have motion autonomy relative to a three-dimensional world.

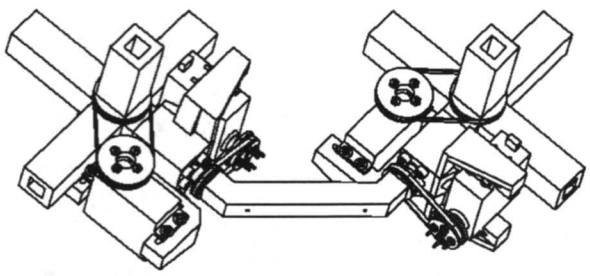


Figure 2: *The CAD model of the Molecule. The Molecule consists of two atoms. A rigid bond that connects the two atoms so that their relative orientation is 90 degrees. The figure shows four actuators, one per degree of freedom. The stalks in the figure represent inter-Molecule connection points.*

3 The Design of the Molecule

3.1 Concept

The initial goal for our Molecule design was to create a regular solid shape which could be closely packed in 3-D space, and which could move about by attaching itself to a substrate of similar units. Since a cube is the simplest regular solid which fully packs in 3-D space, we decided on a cube-shaped design similar to an atom in our current design (Figure 2). This design had inter-unit connectors on each face of the cube, with a rotational degree of freedom about each connector. This design was not capable of independent movement however, two cooperating connected units were necessary for movement. Furthermore, movement required that a unit be able to support the weight of its partner. Because each unit required 6 rotational degrees of freedom plus the weight of the connection mechanisms, we felt it would be very difficult to create a unit which was strong enough to lift another unit.

Consequently, we attempted to reduce the number of degrees of freedom in our unit without severely reducing its capabilities. The result is our robotic Molecule (see Figure 2). It consists of two atoms linked by a rigid 90 degree connection we call the *bond*. Each atom has five inter-Molecule connection points and two degrees of freedom. One degree of freedom allows the atom to rotate 180 degrees relative to its bond connection, and the other degree of freedom allows the atom (and therefore the entire Molecule) to rotate 180 degrees relative to one of the inter-Molecule connectors (the rotational degree of freedom cannot be about the connector opposite the bond connection). This design is capable of independent movement on a substrate of identical Molecules, including straight-line traversal and 90 degree concave and convex transitions to adjacent surfaces. However, the L-shaped design cannot be as closely packed in 3-D space as a cube-shaped design (see section 7).

3.2 Implementation

Our current design uses R/C servomotors for the rotational degrees of freedom and electromagnets for the inter-Molecule connectors. The prototype Molecule uses only a single connector on each atom. This connector is attached to the non-bond rotational degree of freedom as described above. The rotating connection points on each atom are the only connection points required for Molecule motion. The other connection points are used for attachment to other Molecules to create stable 3-D structures. A Molecule to Molecule connection requires the connecting electromagnets to be oppositely polarized so that the electromagnetic fields will attract. The 1" electromagnets used on our prototype are sufficiently powerful to support an entire Molecule. To prevent the rotation of one electromagnet with respect to another, we developed interlocking sheaths which encircle the contacting faces of the electromagnets.

Each Molecule also contains a microprocessor and the circuitry needed to control the servomotors and electromagnets. The microprocessor performs low-level control of the hardware but currently the high-level control of the Molecule takes place off-board in a workstation. The Molecule communicates with the workstation using an RS-485 serial connection.

3.3 Fabrication

We have built a prototype Molecule to demonstrate the feasibility of our approach. The parts cost of the pro-

totype Molecule is about $1000, and it takes approximately three days to fabricate and assemble all parts. We have utilized several technologies to reduce the design iteration time, improve the mechanical precision, and reduce the cost of the prototype[1]. Our design begins with a fully detailed and dimensioned 3-D CAD model of the prototype developed with Pro/Engineer 18.0 from Parametric Technolgy Corp. We then fabricate most of the structure using an FDM1600 Fused Deposition Modeling (FDM) Rapid Prototyping machine manufactured by Stratasys, Inc. This machine and its associated software converts the data from our CAD model directly into high-strength, light weight ABS plastic parts. These ABS parts are assembled together with the electromagnets, the servos, and a few other parts to create a full Molecule.

4 Aggregating Three-dimensional Structures

The Molecule can connect with other identical Molecules to create dynamic three dimensional geometric structures. Figure 3 shows an example of creating a wall by packing Molecules. It is possible to create arbitrary three-dimensional geometric shapes consistent with the Molecular structure. These three-dimensional shapes can only have surfaces that meet at angles of 90 degrees.

Molecules can be placed next to each other without explicit connections. To increase the robustness of the resulting structures, we insist that each Molecule in a structure be connected to at least one other Molecule. One can imagine stringing Molecules together subject to this constraint in an arbitrary fashion. An interesting question is, what the class of three-dimensional structures that can be created using the Molecule? We propose examining this question by defining planar tilings using the Molecule.

Consider a possible tiling of the plane using the Molecule. It is interesting to observe that if we place an xy grid whose size equals the atom size on the plane, each Molecule will occupy three cells in this grid, arranged as an "L" shape. We would like to find ways of covering every cell in such a planar grid with the Molecule. Figure 4(Left) shows an example of a possible tiling that covers every cell in the grid. The generator for the tiling consists of a "string" of two Molecules. Other tilings are also possible. Figures 4(Center) and 4(Right) show two other generators and the resulting tilings. Note that in all three figures the Molecules are simplified by omitting the actuators. In Section 7 we will show how such tilings can be used to generate dynamic structures made out of Molecules that can travel along any direction. In this section we show how tilings help with constructing arbitrary three dimensional objects out of Molecules.

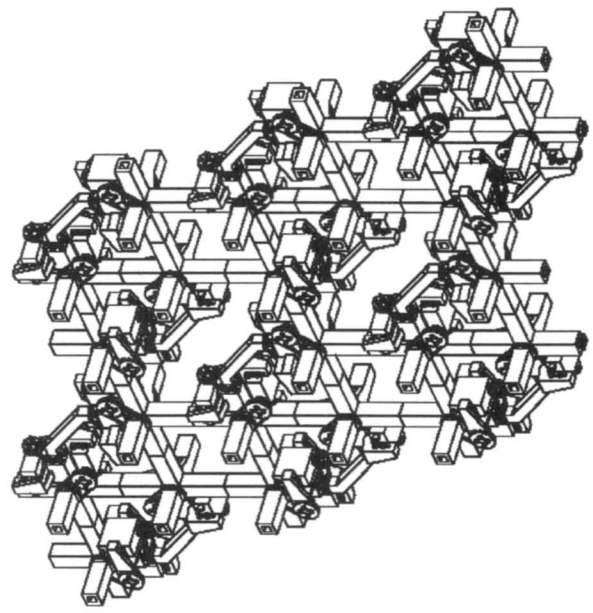


Figure 3: *This figure shows a planar wall created by tiling with the Molecule. This figure details each Molecule to illustrate the location of its connectors and the actuators.*

To show this result, we will use a cell-decomposition method. We define a cell decomposition of a three-dimentional object O to be the division of the object into a union of disjoint cells. An l-grid cell decomposition is a cell decomposition into equal cells where each cell is a square and the side of the square is l. An

[1]We thank Brian Locke and the Thayer School of Engineering for their help in this process.

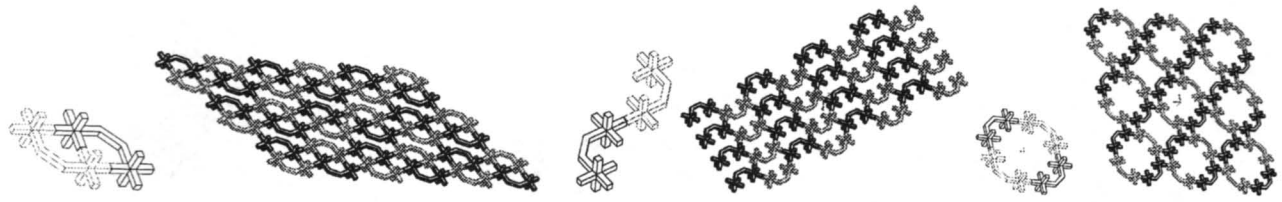


Figure 4: *This figure shows three planar tilings using the Molecule. The "pair" generator on the left is used to create the leftmost tiling. The "string" generator in the middle is used to create the middle tiling. The "ring" generator on the right is used to create the rightmost tiling. The use of color (black or gray) is to visualize better where each individual Molecule is locatet. Lated in Section 7 we show that this particular color encoding of Molecular structures has interesting geometric properties with respect to planning.*

l-grid approximates a three-dimensional object with a Manhattan object[2] that consists of a union of identical cubes.

Theorem 1 *Any connected three-dimensional Manhattan object that allows an l-grid cell decomposition where l, the size of the grid, is the size of a pair tile can be created using Molecule robots.*

Proof: It is possible to place an xyz grid of size equal to the tile size on the object. Since pair tiles pack tightly in the plane, we can construct each xy section of the object by selecting an appropriate planar shape. Since pair tiles are flat (i.e. all atoms are coplanar) we can stack these tiles on top of one another in a tightly packed fashion. So, we can construct each xy section of the object by selecting an appropriate planar shape. These layers can be stacked to obtain the desired object. □

Corollary 2 *Any three-dimensional structure that allows a cell decomposition where the size of the grid is the size of an individual Atom, can be approximated using Molecule robots, with an approximation error of at most two cells for every cell on the xy exterior of the structure.*

Proof: Begin by outlining each xy slice of the desired Manhattan Object. Stack pair tiles so that all cells in each slice of the object are completely covered. Now, remove any Molecules that do not cover at least one cell in the desired object. Among the remaining Molecules, there may be some that cover one or two cells that are not in the desired structure. These Molecules, must, however, cover at least one cell in the desired structure or they would have been removed already. Furthermore, since this cell is adjacent to a cell in the same plane that is not in the desired structure, it must be on the xy exterior. So, for each cell on the xy exterior of the desired structure, there can be at most two cells in the actual structure that are not in the desired structure. □

Note that Theorem 1 and Corollary 2 give a method for constructing objects out of Molecules and a measure for quantifying the approximation of the construction only when the size of the object is large relative to the size of an individual Molecule. One can imagine building these Molecules at the MEMS scale. Such a construction would make possible the construction of objects with high-resolution surfaces.

A caveat of Theorem 1 and its corollary is that the weight of the object can be supported by its molecular layers. Since the body of our Molecule is currently made out of plastic, this places some restrictions over how many can be stacked on top of each other. We are currently investigating this question.

[2] A Manhattan object has all its surfaces either parallel or perpendicular to one another.

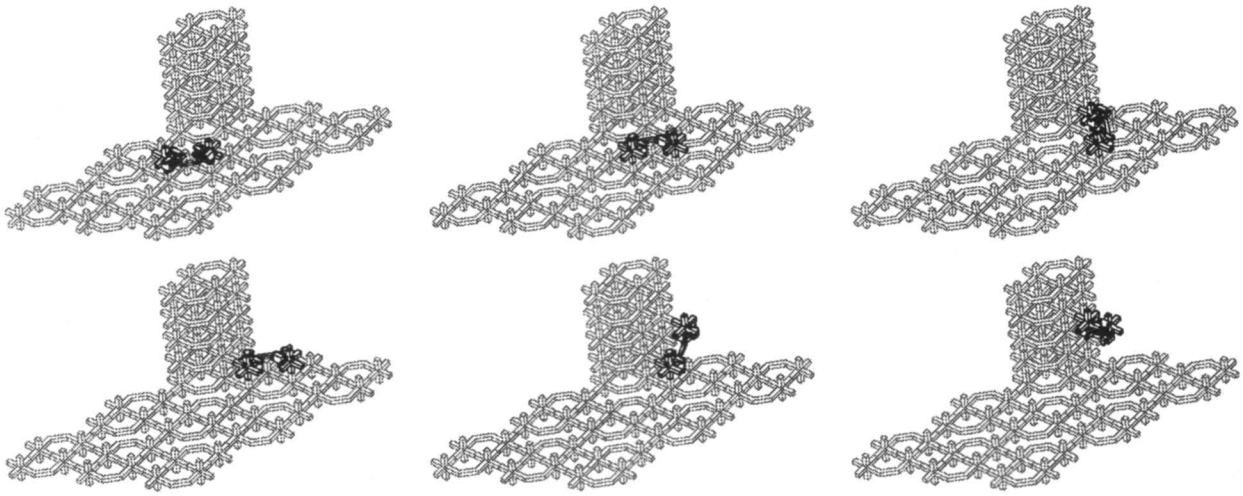


Figure 5: *This figure shows a Molecule walking on a structure composed of identical Molecules. The structure, shown in grey, consists of a horizontal plane and a vertical tower. The Molecule, shown in black, starts on the plane (top, left), rotates 180 degrees about the connector of its right atom to the plane (top, center), rotates 90 degrees about the connector of its right atom to the plane (top, right), rotates 90 degrees about the connector of its left atom to the plane (bottom, left), rotates vertically 90 degrees about the bond connector of the top atom to transition and make contact with the tower (bottom, center), and finally it rotates 180 degrees about the connector of its top atom to the wall to traverse the tower.*

5 Molecule Motion Control

In the previous section we described the class of three-dimensional objects that can be built out of Molecules. Objects belonging to this class can metamorphose into one another by using the Molecule degrees of freedom. In this section we focus on the Molecule motion control for moving an individual Molecule on a substrate of other Molecules. In Section 6 we give a geometric approach to planning Molecule trajectories on a substrate of other Molecules. In [9, 11] we present other planning bounds and algorithms for controlling the global motion of a structure made of Molecules.

Molecule motion is controlled by attaching an atom to some other Molecule and actuating one or more of the four degrees of freedom of the Molecule. Because atoms can only occupy points in a cubic lattice, only actuator positions which rotate an atom (or the entire Molecule) by 90 degrees are valid for attachment. Therefore, in theory, only discrete 90 degree actuator positions are required. However, in practice, other actuator positions are used to compensate for Molecule misalignment due to mechanical tolerances.

Our Molecule design is capable of linear walking on a planar lattice of Molecules, and concave and convex transitions to planar lattices of Molecules oriented at 90 degrees to the original surface. We describe the algorithms for these motions in the following sections.

5.1 Linear walking

Figure 6 shows the three stages in a linear walk. The arrows represent the positions of the connection point actuators. The atom with the bold outline is connected to the underlying lattice represented by the checkerboard pattern. Figure 6(Left) represents the initial state of the Molecule. Figure 6(Center) shows the result after the Molecule has pivoted clockwise 90 degrees about the connected atom (the atom with the gray arrow) and has swapped atom connections. Note that the atom designated by the black arrow also performed a 90-degree rotation of its connection point while it was being moved to its new position (if the connection point

had not rotated with respect to the atom it would be pointing right instead of up). This was done to compensate for the rotation required by the next move. Figure 6(Right) shows the result after the Molecule has rotated counterclockwise 90 degrees about the connected atom (the atom with the black arrow) and has swapped atom connections. Note again that the moving atom has also performed a 90 degree rotation of its connection point. Since Figure 6(Right) shows the Molecule in the same pose as Figure 6(Left) and the Molecule has moved forward one white square, this algorithm can repeated as often as necessary to move the Molecule a given amount in a straight line, assuming that the underlying lattice exists to support the Molecule.

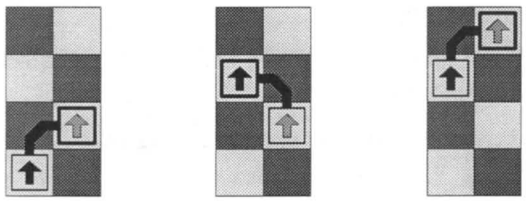


Figure 6: *A linear walk sequence.*

5.2 Concave transition

Figure 7 shows the sequence of Molecule moves necessary to perform a concave transition. The atoms are represented by large blue cubes and the small yellow, cyan, and purple cubes represent the inter-Molecule connectors. Yellow connectors can rotate, the others cannot. The checkered surface represents a lattice of Molecule connectors. Figure 7(Left) shows the Molecule in the initial position. The atom nearest to the wall is attached to the floor surface. Figure 7(Second from left) shows the pose after a 90-degree rotation about the bond connection of the attached atom. This causes the unattached atom to be suspended in midair one atom position to the left and one atom position above its original position. Figure 7(Second from right) shows the pose after a 90-degree rotation about the inter-Molecule connector of the attached atom. The result of this motion is to bring the unattached atom into contact with the wall. At this point the atoms swap connections so that the previously unattached atom is now attached, and the previously attached atom is unattached. Figure 7(Right) shows the final stage of the transition in which the lower atom is rotated to place its inter-Molecule connector on the wall. From this point the Molecule can execute a linear walk to climb the wall.

5.3 Convex transition

Figure 8 shows the sequence of moves necessary to perform a convex transition. Figure 8(Upper Left) shows the initial position of the Molecule. The atom with the cyan non-rotating inter-Molecule connectors is attached to the checkered surface by its rotating (yellow) inter-Molecule connector. The first rotation is about this connector and it causes the unattached atom to swing out over the convex edge. The next rotation about the bond connection of the unattached atom positions the rotating connector so that it faces the vertical surface. A rotation about the bond connection of the attached atom then moves the unattached atom into contact with the vertical surface below the attached atom. Atom connections are then swapped so that the atom with the purple non-rotating connectors is now attached the the lattice, and the atom with the cyan non-rotating connectors is free to move. The next three rotations move the newly unattached atom into free space above the vertical surface, position the rotating connector of the unattached atom so that it faces the vertical surface, and move the unattached atom into contact with the vertical surface. At this point the Molecule can traverse the vertical surface using the linear walk algorithm.

6 Molecule Trajectory Control

The Molecule motion algorithms are specified at the level of very basic control, as a combination of individual rotations. For more global motions, especially when the robotic system contains large numbers of Molecules, this approach is too low-level and difficult to reason about, manage, and program. When multiple Molecules move in parallel as a distributed sys-

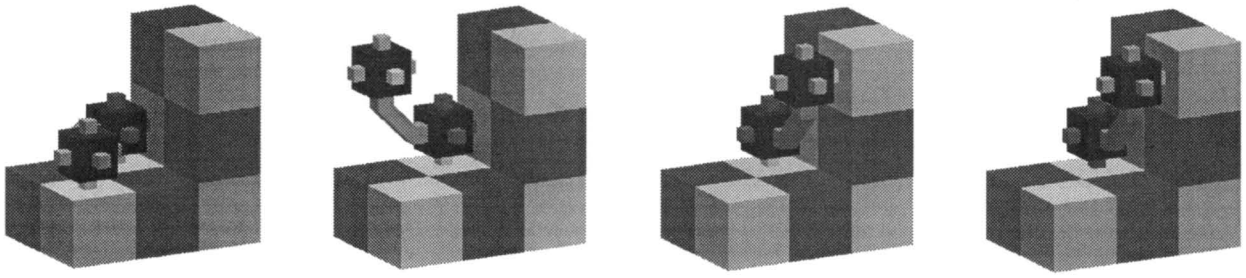


Figure 7: *A concave transition sequence. Rotating connectors are yellow, non-rotating connectors are cyan on one atom and purple on the other atom. The bond is green. The checkered surface represents a lattice of Molecule connectors.*

tem, individual motion management becomes impossible. For example, over 15 individual rotations are currently used to implement one primitive in the global (planar) walk of a two module or four module structure [9].

In this section we show a geometric formulation of Molecule motions that reduces the trajectory planning problem for individual Molecules to graph search.

6.1 A graph model

Suppose S is a connected molecular structure composed of k Molecules arranged as an arbitrary three-dimensional structure. Because Molecules connect to one another either in the plane or by stacking up, S is a Manhattan object. Let xyz be a system of coordinates that aligns the axes with the main directions of S. The unit in this system of coordinates is the length of an atom.

Consider the following coloring scheme for the Molecule structure. For every atom, let s be the sum of its x, y, and z coordinates. If s is even, assign the atom the color *blue*. If s is odd, assign the atom the color *red*. This coloring scheme defines a unique partition for S with the following properties:

1. two atoms that belong to the same Molecule have the same color (because they must be placed along some diagonal)
2. two adjacent (connected) atoms that belong to different Molecules have different color (because there is a difference of one unit in one coordinate only)

For example, if S is a plane, this coloring scheme defines a checkered surface.

Now, consider the set of planar faces that defines the exterior of S. Let F be such a face.

Theorem 3 *A blue Molecule that rests on F has to start on top of a red Molecule and it can reach any red location on F to which there is a continuous diagonal sequence of red atoms. A red Molecule that rests on F has to start on top of a blue Molecule and it can reach any blue location on F to which there is a continuous diagonal sequence of blue atoms.*

Proof: The coloring scheme justifies the initial state. The planar walking algorithm described in Section 5 is capable of traversing any diagonal and of crossing from one diagonal to any perpendicular diagonal. □

An interesting consequence of Theorem 3 follows when we observe that in order for a Molecule to move within a structure its first step is to climb on top of a Molecule colored with a different color.

Corollary 4 *Molecules do not change partitions.*

Theorem 3 and Corollary 4 suggest a graph representation for computing reachability on the face F. Let $G_b = (V_r, E_r)$ be the undirected graph that captures the reachability set of blue Molecules on F. Let

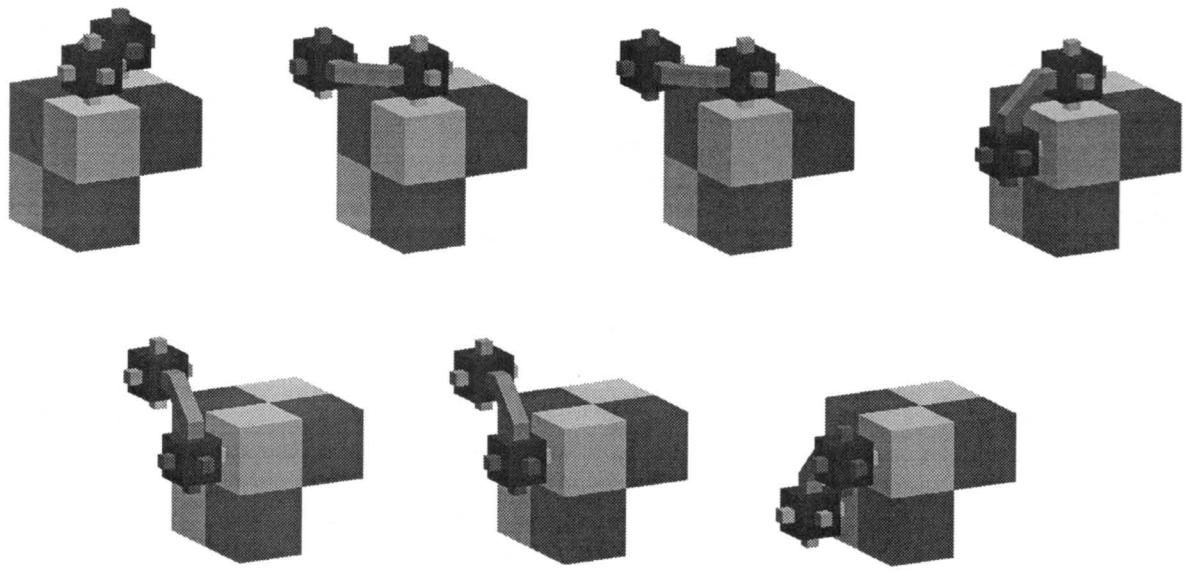


Figure 8: *A convex transition sequence.*

$G_r = (V_b, E_b)$ be the undirected graph that captures the reachability set of red Molecules on F. We will construct these graphs as follows. Without loss of generality, we will illustrate the algorithm for $G_r = (V_b, E_b)$.

1. the vertex set V_b consists of all blue atoms on F
2. the edge set E_b connects any two vertices adjacent along diagonals

It is straightforward to see that Molecule trajectories correspond to paths in these graphs.

6.2 Trajectory planning

In this section we show how to specify geometrically a three-dimensional trajectory for a Molecule.

Our first observation is that Molecules move on any exterior face of the structure. Within each face, the motion is captured by a graph as described in Section 6.1. Moving from one location to another within this face reduces to searching this graph.

The face graph construction can be generalized to capture any trajectory of the Molecule along the exterior surface of the structure as the graphs $G_b = (V_r, E_r)$ and $G_r = (V_b, E_b)$ as follows. Intuitively, the vertex set is the union of the vertex sets for each face. The edge set is the union of the edge sets for each face, plus transition edges. For any pair of adjacent faces a transition edge connects an atom from one face to an atom on the other face when it is possible to place a Molecule on top of those two atoms. The number of transition edges from any individual atom of one face to the adjacent face is constant, because each Molecule has only 4 degrees of freedom. It is easy to see that the $|E_r| = O(|V_r|)$ and $|E_b| = O(|V_b|)$.

The following result follows.

Theorem 5 *Let S be a Molecular structure of k Molecules. Let M be a Molecule on the exterior of this structure. Let I be an initial location and F be a final location for M. The shortest path from I to G can be computed in $O(k)$.*

Proof: The motion plan from I to G can be obtained by by constructing the reachability graph for M and searching the graph for the shortest path in the number of edges. The reachability graph $G = (V, E)$ for

this structure can be computed in $O(k)$. The size of the edge set $|E| = O(k)$. A breadth-first-search will compute the shortest path, or find that no path exists in $O(k)$.

To compute an actual motion plan for the Molecule, an expression in the control space of individual Molecules[3] has to be synthesized from the graph. To do this, the construction of the graph is augmented with the following step. For each computed edge, a label that indicates the type of rotation required to make the step can be determined straightforwardly by examining local geometry. The motion plan consists of an expression in the language motion obtained by concatenating the graph labels during the traversal from I to G. □

7 Mobility and Gaits

Molecules can walk on three dimensional structures by using the trajectory control and planning algorithms descried in Section 6. Figure 5 shows an example in which the Molecule travels to the base of a tower and climbs on the tower. Note that in this example, the planar surface is composed with a "pair" tiling and the tower is created by stacking "pair" tilings. In general, the Molecule motion follows a pattern that is dictated by the geometry of the underlying Molecule substrate. In this section we examine several different Molecule gaits and their properties for the three tilings described in Section 4.

7.1 Moving along a straight-line trajectory

The pattern of individual rotations that achieves a given transition in the x (or y) direction is called a *gait*. Figure 4 shows three different tilings for the base plane. Note that each generator can be partitioned in two sets and the Molecules in each set can be colored either red or blue. We choose this color encoding because it helps us visualize better where each Molecule is located. We will also show that this particular color encoding has interesting properties with respect to planning.

[3]The control space of a Molecule consists of the four rotations each Molecule is capable of executing.

Suppose there is an infinite plane generated by one of the given tiles. Suppose we wish for Molecule M composed of atoms A_1 and A_2 to effect a unit translation in the x direction. We add a system of coordinates in the following way. Each atom consists of 4 connectors oriented at 90 degree intervals. Let the x axis be defined by two colinear connectors and let the y axis be defined by the other two colinear connectors. The two axies defined in this manner are orthogonal. The origin of this system of coordinates can be anywhere in the plane. Without loss of generality, suppose the origin is the lowest left-most atom in the tiling. Given an integer d, the following algorithms will generate translations by 1 unit in the x and/or y direction of the tiled plane. Each algorithm can be iterated d times to generate translations by d units.

If the planar surface is tiled with "pairs", M will perform a rotation of 180 degrees about A_1, followed by a rotation of -90 degrees about A_2, followed by a rotation of 90 degrees about A_1. Translations in the y direction are similar.

Theorem 6 *A Molecule that starts on top of a black Molecule in a pair tiling can reach all the other black locations and none of the gray locations. A Molecule that starts on top of a gray Molecule in a pair tiling can reach all the other gray Molecules and none of the black Molecules.*

Proof: A Molecule can only start on top of another Molecule so both its atoms rest on the same color Molecules in the initial configuration. Without loss of generality, suppose the Molecule starts on top of a black Molecule. Each 180 degree rotation or 90 degree rotation allows one of the atoms of the Molecule to reach another black atom in the tiling. Rotations by 180 degrees allow the Molecule to reach other black Molecules along the diagonal. Since the Molecule can move by 1 unit to the next black position on the x axis and it can move by 1 unit to the next black position on the y axis, by composing these moves the Molecule can reach any black position in the tiling. □

Similar results hold for "ring" and "string" tiles. Note that for "ring" tilings, translations in the x direction are done with a rotation by 180 degrees about A_1.

This also works if the same rotation is performed about A_2. The translations in the y direction are similar. If the planar surface is tiled with "strings," translations in the x direction are not possible. Translations in the y direction are done with a rotation of 90 degrees about A_1 followed by a rotation of -90 degrees about A_2.

To decide which gait is most appropriate for a locomotion task, we introduce a gait complexity measure. The complexity of a gait with respect to a unit translation[4] is computed as the number and size of the distinct rotations required to do the motion.

We can combine the gait complexity measure with the reachability results in a decision procedure for choosing the appropriate underlying tiling for particular locomotion tasks. If locomotion is required along arbitrary directions, pair tilings is most appropriate. If locomotion is required along orthogonal directions, the ring tiling is most appropriate. If locomotion is required along one direction only, the string tiling is most appropriate. Pair and ring tilings are also possible, but the string tiling is the most efficient. The string tiling can be parallelized so that we can imagine a locomotion modality where such a structure could be "rolled forward".

8 Experiments

We have built one Molecule and implemented the control algorithms for linear motion, convex transition, and concave transition. Because we only have one Molecule, we simulated a lattice of Molecules by building a plexiglass structure that supports linear translations as well as convex and concave transitions. We bolted inter-Molecule connectors at the locations where we expect to have Molecules. We performed experiments in which the Molecule successfully performed linear translations and convex and concave transitions. The Molecule used dead-reckoning to locate the connectors.

[4]Unit translations can be translations in the x direction by one molecular unit, translations in the y direction by one molecular unit, or any user-defined measure that is relevant to the required locomotion task.

The errors did not seem to accumulate from step to step. We believe this is because after each individual rotation the Molecule connects with a rigid connection. The connection sheaths aligned the Molecules, minimizing the orientation errors.

9 Discussion

Our vision is to create robots by using self-reconfiguration: hundreds of small modules can autonomously organize and reorganize as geometric structures to best fit the terrain on which the robot has to move or the shape the object the robot has to manipulate. In this paper we described our design for a module we call a Molecule that can be used to aggregate as arbitrary three-dimensional self-reconfigurable structures. We described the structure of the Molecule, the control algorithms for effecting linear translations, convex transitions, and concave transitions, and discuss the potential for this Molecule to be a universal module. We discussed geometric trajectory control algorithms for planning the motion of an individual Molecule on a substrate of other Molecules. We then showed the geometric construction of a graph that reduces the Molecule trajectory planning to a graph search. Finally, we examined gaits for three different structures of the underlying Molecule substrate. An expression for the final trajectory in the rotation space of the Molecule can be extracted automatically from this graph.

Much work on global motion planning algorithms for Molecular structures needs to be done. We are currently investigating several approaches to defining gaits for moving Molecular structures of different terrain types, and transformations between these gaits.

Our current experiments should we viewed as pilot studies for the Molecular design. These experiments have established that it is possible to control the four degrees of freedom of Molecules to effect rotations of $+90$ degrees and -90 degrees. Individual Molecules can be controlled to move relative to other Molecules to effect linear translations and convex and concave transitions. These results are promissing. We plan to build 16 Molecules to evaluate our self-reconfiguring theory.

Acknowledgements

We are grateful to Bruce Donald for his careful reading of an earlier draft and for his very insightful comments and suggestions. We are also grateful to Brian Locke for his help and advice regarding the rapid prototyping machine.

This paper describes research done in the Dartmouth Robotics Laboratory. Support for this work was provided through the NSF CAREER award IRI-9624286 and the NSF award IRI-9714332.

References

[1] I. Chen and J. Burdick, Enumerating the Non-Isomorphic Assembly Configurations of a Modular Robotic System, to appear in the *International Journal of Robotics Research.*

[2] G. Chirikjian and J. Burdick, Kinematics of a hyper-redundant robot locomotion with applications to grasping, in *Proceedings of the IEEE International Conference on Robotics and Automation*, 1991.

[3] R. Cohen, M. Lipton, M. Dai, and B. Benhabib, Conceptual design of a modular robot, *Journal of Mechanical Design*, March 1992, pp. 117-125.

[4] B. Donald, J, Jennings, and D. Rus, Minimalism + Distribution = Supermodularity, in *Journal of Experimental and Theoretical Artificial Intelligence*, 9(1997)293-321, 1997.

[5] T. Fukuda and Y. Kawauchi, Cellular robotic system (CEBOT) as one of the realization of self-organizing intelligent universal manipulator, in *Proceedings of the 1990 IEEE Conference on Robotics and Automation*, pp. 662-667.

[6] G. Hamlin and A. Sanderson, Tetrabot modular robotics: prototype and experiments, in *Proceedings of the IEEE/RSJ International Symposium of Robotics Research*, pp 390-395, Osaka, Japan, 1996.

[7] K. Kotay and D. Rus, Task-reconfigurable robots: navigators and manipulators, in *Proceedings of the 1997 International Conference on Intelligent Robots and Systems*, Grenoble, France, 1997.

[8] K. Kotay, D. Rus, M. Vona, and C. McGray, The Self-Reconfiguring Robotic Molecule, in *Proceedings of the 1998 International Conference on Robotics and Automation*, Leuven, Belgium, 1998.

[9] K. Kotay and D. Rus, Trajectory Control for Self-reConfigurable Molecular Structures, submitted to the 1998 International Conference on Intelligent Robots and Systems.

[10] J. C. Latombe, *Robot Motion Planning*, Kluwer Academic Publishers 1991.

[11] C. McGray and D. Rus, Self-Reconfigurable Molecules as 3D Metamorphic Robots, submitted to the 1998 International Conference on Intelligent Robots and Systems.

[12] S. Murata, H. Kurokawa, and Shigeru Kokaji, Self-assembling machine, in *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, San Diego, 1994.

[13] B. Neville and A. Sanderson, Tetrabot family tree: modular synthesis of kinematic structures for parallel robotics, in *Proceedings of the IEEE/RSJ International Symposium of Robotics Research*, pp 382-390, Osaka, Japan, 1996.

[14] A. Pamecha, C-J. Chiang, D. Stein, and G. Chirikjian, Design and implementation of metamorphic robots, in *Proceedings of the 1996 ASME Design Engineering Technical Conference and Computers in Engineering Conference*, Irvine, CA 1996.

[15] C. Paredis and P. Khosla, Design of Modular Fault Tolerant Manipulators, in *The First Workshop on the Algorithmic Foundations of Robotics*, eds. K. Goldberg, D. Halperin, J.-C. Latombe, and R. Wilson, pp 371-383, 19 95.

[16] C. Paredis and P. Khosla, Kinematic Design of Serial Link Manipulators from Task Specifications, in *International Journal of Robotic Research*, Vol. 12, No. 3, pp 274–287, 1993.

[17] M. Yim, A reconfigurable modular robot with multiple modes of locomotion, in *Proceedings of the 1993 JSME Conference on Advanced Mechatronics*, Tokyo, Japan 1993.

[18] E. Yoshida, S. Murata, K. Tomita, H. Kurokawa, and S. Kokaji, Distributed Formation COntrol of a Modular Mechanical System, in *Proceedings of the 1997 International Conference on Intelligent Robots and Systems*, Grenoble, France, 1997.

Taylor & Francis
Taylor & Francis Group
http://taylorandfrancis.com

Taylor & Francis
Taylor & Francis Group
http://taylorandfrancis.com

Author Index

Nancy M. Amato 155
O. Burchan Bayazit 155
Robert-Paul Berretty 279
Karl F. Böhringer 1, 297
Joel W. Burdick 81
John F. Canny 21
Jingliang Chen 297
Moëz Cherif 103
Gregory S. Chirikjian 363
Howie Choset 35
John J. Craig 133
Lucia K. Dale 155
Anton Deguet 327
Yves Delnondedieu 351
Bruce R. Donald 1
Herbert Edelsbrunner 265
Michael Erdmann 119, 251
Ken Goldberg 279, 297
Leonidas J. Guibas 191
Kamal K. Gupta 103
Dan Halperin 297
Frank Hoffmann 211
Christopher Holleman 313
David Hsu 141
Wesley H. Huang 91
Christian Icking 211
Hazem Jaouni 69
Yan-Bin Jia 251
Christopher Jones 155
Ammar Joukhadar 327
Lydia E. Kavraki 141, 313
Scott D. Kelly 81
Maher Khatib 69
Rolf Klein 211
Keith Kotay 375
Klaus Kriegel 211
Shankar Krishnan 177
Alexander B. Kyatkin 363
Florent Lamiraux 313
Jean-Claude Latombe 141
Christian Laugier 327
Jean-Paul Laumond 69
Steven M. LaValle 339
Ming C. Lin 177
Jonathan Luntz 35
Kevin M. Lynch 237
Dinesh Manocha 177
Matthew T. Mason 91
Craig McGray 375
William Messner 35
Rajeev Motwani 141
Richard M. Murray 81
Mark H. Overmars 279, 297
Amol Pattekar 177
James Radford 81
John Reif 49, 223
Dan S. Reznik 21
Daniela Rus 375
Jürgen Sellen 59
Stephen Sorkin 141
A. Frank van der Stappen 279
Zheng Sun 223
Jocelyne Troccaz 351
Daniel Vallejo 155
Marsette Vona 375
Hongyan Wang 49
Joe Warren 169
Yan Zhuang 297