Efficient Motion Planning using Generalized Penetration Depth Computation

Liangjun Zhang

A dissertation submitted to the faculty of the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science.

Chapel Hill
2009

Abstract

Liangjun Zhang: Efficient Motion Planning using Generalized Penetration Depth Computation.
(Under the direction of Dinesh Manocha.)

Motion planning is a fundamental problem in robotics and also arises in other applications including virtual prototyping, navigation, animation and computational structural biology. It has been extensively studied for more than three decades, though most practical algorithms are based on randomized sampling. In this dissertation, we address two main issues that arise with respect to these algorithms: (1) there are no good practical approaches to check for path non-existence even for low degree-of-freedom (DOF) robots; (2) the performance of sampling-based planners can degrade if the free space of a robot has narrow passages.

In order to develop effective algorithms to deal with these problems, we use the concept of penetration depth (PD) computation. By quantifying the extent of the intersection between overlapping models (e.g. a robot and an obstacle), PD can provide a distance measure for the configuration space obstacle (C-obstacle). We extend the prior notion of translational PD to generalized PD, which takes into account translational as well as rotational motion to separate two overlapping models. Moreover, we formulate generalized PD computation based on appropriate model-dependent metrics and present two algorithms based on convex decomposition and local optimization. We highlight the efficiency and robustness of our PD algorithms on many complex 3D models. Based on generalized PD computation, we present the first set of practical algorithms for low DOF complete motion planning. Moreover, we use generalized PD computation to develop a retraction-based planner to effectively generate samples in narrow passages for rigid robots. The effectiveness of the resulting planner is shown by alpha puzzle benchmark and part disassembly benchmarks in virtual prototyping.

To my wife, Feiqi Su

Acknowledgments

First and foremost I am deeply grateful to my advisor Dinesh Manocha for his excellent guidance and support not only on my research but also on all professional aspects. His patience and tolerance have allowed me to learn from lessons. His quick and constructive feedbacks on my papers and presentations always amaze me. I would also like to thank all of my committee members. I thank Young J. Kim for being a coauthor and a guide of my research. I thank Ming C. Lin for her support and feedbacks. I thank Steven M. LaValle for his thoughtful discussions, inspiration, and hosting my visit in his department. I thank Mark Foskey for his suggestions.

Many of my work were done jointly with other collaborators. I thank Gokul Varadhan, Avneesh Sud, Xin Huang, Jia Pan, and Shankar Krishnan for their contributions. Many thanks to all the members of the GAMMA group for their feedbacks and comments in the weekly meetings.

I would like to thank the faculty and staff in the computer science department. They have made the department such a pleasant place to study and work at. In particular, I thank Jack Snoeyink for his advices. I thank Janet Jones, Linda Houseman, and Dawn Andres for their administrative support, and Whitney Vaughan for proofreading my papers. I also thank the funding agencies of my work: ARO, DARPA, Intel Corporation, NSF, and ONR.

I thank David Xianfeng Gu for his help and guidance in my first year study in the United States. I thank Will Alexander for proofreading and regular lunch meetings. I also thank many friends - Xueyi Wang, Lei Wei, Anish Chandak, Hua Yang, Leo Yuanxin Liu, Xiang Zhang, Qi Zhang, Feng Pan, Li Guan, and Hao Xu. Without them, the life in Chapel Hill would not have been fun.

I thank my parents, sister, brother, and parents-in-law in China for their care. Most of all, my wife Feiqi Su for her endless love and support.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Motion is ubiquitous in both the real world and the virtual world. The area of motion planning deals with design of computational tools to enable robots to automatically compute their motion in order to perform high level tasks, e.g. go safely from point A to B. Besides robotics, motion planning algorithms have also been successfully used in many other fields, such as computer animation, computational biology and CAD/CAM.

This thesis focuses on developing efficient motion planning algorithms. In this chapter, we first give an overview of the problem and review prior approaches. Next, we discuss some of the issues and challenges in designing efficient algorithms. Finally, we present an overview of our algorithms and summarize the new results of this thesis.

1.1 Motion Planning

In this thesis we mainly focus on the basic problem in robot motion planning, which is also known as path planning and defined as follows. Let A be a robot. We assume that its kinematics (i.e. degree of freedom or DOF of the robot) and geometric representation (i.e. shape of the robot) are known. The robot operates in a 2D or 3D Euclidean space, called the *workspace*, and let $B = \{B_1, B_2, ..., B_n\}$ be the collection of obstacles distributed in the workspace. We assume the obstacles are static and their geometric representation is also known. Given an initial configuration and a goal configuration of

A, the basic motion planning problem is to compute a collision-free path so that A can move from the initial configuration to the goal configuration without colliding with any obstacle B_i in the workspace, or to report failure if no such path exists. Fig. 1.1(a) shows an example of path computed for a triangle-shaped robot in the environment with one static obstacle.

The basic motion planning problem can be extended in different ways. For instance, there can be multiple robots in the environment, or the robot's motion needs to satisfy not only the collision-free constraint but also other dynamic constraints. Furthermore the obstacles may not be static; or the robot is navigating in an unknown environment and an exact geometric representation of obstacles is not known in advance. In this thesis, we mainly focus on the basic motion planning problem for two reasons: first even the basic motion planning problem is computationally challenging and there are still many important and unsolved issues with respect to that problem; second the techniques developed for the basic problem are also applicable to solve the more complex motion planning problems.

1.1.1 Configuration Space

Configuration space is an important notion used in motion planning (Lozano-Pérez, 1983; Latombe, 1991; Choset et al., 2005; LaValle, 2006). For a rigid model, the configuration space (often shortened as C-space) $\mathcal{C}$ is defined as the set of all possible positions and orientations of the model. For example, Fig. 1.1 shows a 2D triangle-shaped robot which can only translate in the 2D plane. For this robot, its C-space is 2-dimensional and each configuration $\mathbf{q} \in \mathcal{C}$ is defined by the x and y coordinates of the robot, i.e. $\mathbf{q} = (x, y)$. Therefore, every configuration maps to a point in the 2-dimensional C-space. For a rigid free-flying robot in 3D, its C-space is 6-dimensional and corresponds to SE(3), the group of the spatial rigid body transformations. For an articulated robot, the dimensionality of its C-space is determined by the number of links of the robot as well as its kinematic

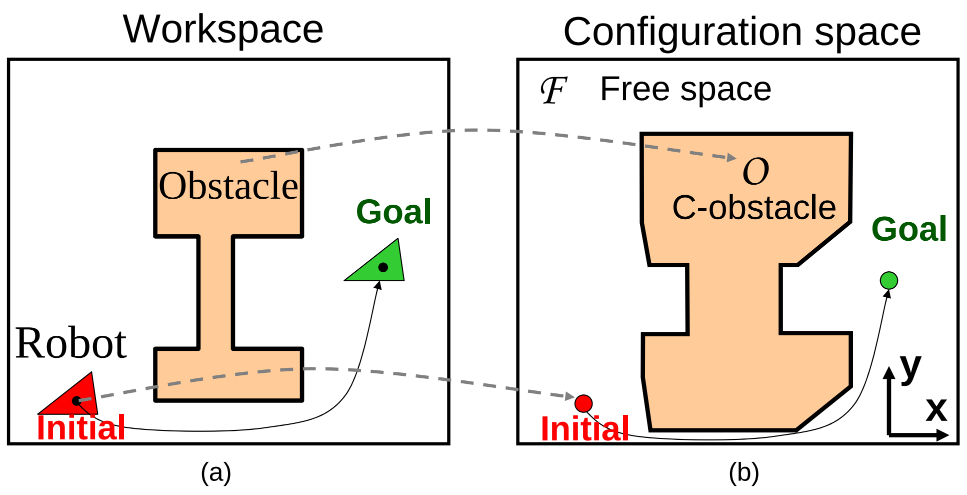


Figure 1.1: Motion planning and configuration space. *The problem of planning a collision-free motion for a robot (e.g. a 2D triangle-shaped robot translating in plane) between its initial configuration and goal configuration is reduced to finding a path connecting the initial and goal in the free space of the robot's configuration space, which is 2-dimensional for this robot.*

structures (e.g. chains or loops).

The configuration space of a robot is decomposed into *free space* - $\mathcal{F}$, *contact space* - $\mathcal{C}_{contact}$ and *configuration space obstacle* (often shortened as C-obstacle) - $\mathcal{O}$. *Free space* in C-space is defined as the set of configurations at which the robot does not intersect with any obstacle; *contact space* is defined as the set of configurations at which the robot barely touches one or more obstacles without any penetration. The complement of $\mathcal{F} \cup \mathcal{C}_{contact}$ in $\mathcal{C}$ is C-obstacle. In another way, if a configuration is in C-obstacle, the robot collides with one or more obstacles in the environment with penetration. Such configurations should be avoided when the robot moves.

Based on the notion of configuration space, the basic motion planning problem is reduced to finding a path in robot's free space connecting the given initial and goal configurations. In practice, the robot is often allowed to touch the obstacles. In this case, the path can lie in free space as well as contact space.

In general, the motion planning problem is computationally challenging due to the underlying high dimensionality of C-space as well as the combinatorial complexity and the robustness issues of geometric computation of free space. The boundary (closure) of free space is equivalent to the contact space and is determined by a set of contact surfaces. Each contact surface is the locus of configurations of a robot at which a specific boundary feature of the robot (i.e. vertex, face, or edge of a rigid robot represented as polyhedra) is in contact with a boundary feature of the obstacles. For a robot with rotational DOF, the resulting contact surfaces can be non-planar or high degree hypersurfaces. The computation of the boundary of free space reduces to an arrangement problem (Latombe, 1991; Halperin, 2002). Given a finite set of hypersurfaces in $\mathbb{R}^d$, their arrangement is the decomposition of $\mathbb{R}^d$ into cells of dimensions $0, 1, \ldots, d$. It is known that in the worst case, the combinatorial complexity of an arrangement of n hypersurfaces in $\mathbb{R}^d$ is $O(n^d)$ (Halperin, 2004). For a rigid robot represented as polyhedra with both of translational and rotational DOF, in general, there are $O(n^2)$ contact surfaces where n is the number of features (e.g. vertices, edges and faces) of the robot and obstacles. The combinatorial complexity of the boundary of free space arrangement of n^2 contact surfaces in 6-dimensional C-space can be $O(n^{12})$ in the worst case. In addition, the arrangement of contact surfaces involves the intersection computation which is difficult due to robustness issues such as floating point errors and degeneracies (Raab, 1999). Therefore, it is very challenging to compute the boundary of the free space precisely, robustly and efficiently in high dimensional C-space.

Motion planning has been intensively studied for three decades. Prior motion planning approaches can be classified according to how they represent and capture the connectivity of the robot's free space. Now we briefly overview some of these approaches.

1.1.2 Exact and Approximate Decomposition Approaches

Some of the earlier motion planning algorithms relies on an exact representation of free space. These include criticality-based algorithms for a class of robots (Lozano-Pérez and Wesley, 1979; Donald, 1987; Kedem and Sharir, 1988; Avnaim and Boissonnat, 1989b), roadmap methods (Canny, 1988), and exact cell decomposition methods (Schwartz and Sharir, 1983). The exact cell decomposition algorithms compute an exact representation of the boundary of free space. These algorithms partition the free space into a collection of simpler geometric regions and compute a connectivity graph representing the adjacency between the regions. Recently, a star-shaped roadmap representation of $\mathcal{F}$ has been proposed and applied to low-DOF robots (Varadhan and Manocha, 2005). It has been shown that the complexity of the exact boundary computation of free space is exponential in the number of robot's degrees of freedom (e.g. $O(n^{12})$ for a free-flying rigid robot) (Reif, 1979; Halperin, 2002). Furthermore, exact free space space computation is also prone to robustness issues. Therefore, most implementations of exact approaches are limited to low DOF (e.g. 2-3) robots or special shapes (e.g. spheres or ladders) (Avnaim and Boissonnat, 1989b).

A number of motion planning algorithms are based on approximate cell decomposition (ACD) (Brooks and Lozano-Pérez, 1985; Paden et al., 1989; Zhu and Latombe, 1990). ACD algorithms attempt to subdivide $\mathcal{C}$ into a collection of cells similar to exact cell decomposition. Unlike exact cell decomposition, the cells in ACD have a simple shape (e.g. rectangoloids). ACD algorithms compute a collision-free path within the cells in the free space or check for path non-existence using the cells that lie in C-obstacle. ACD algorithms are much simpler to implement as compared to exact approaches. However, since it is difficulty to check whether a cell lies entirely in the free space and the number of cells grows exponentially with the level of subdivision, most prior ACD algorithms have been limited to 2-3 DOF robots in simple environments.

1.1.3 Sampling-based Approaches

The sampling-based approaches such as probabilistic roadmaps (PRM) (Kavraki et al., 1996), rapidly-exploring random trees (RRT) (LaValle, 1998) and their variants are most widely used motion planning algorithms for many practical applications. These sampling-based approaches attempt to capture the connectivity of $\mathcal{F}$ by sampling in the free space and connecting the samples to form a roadmap or a tree. These approaches are very simple to implement and have been successfully applied to high-DOF robots. However, the performance of sampling-based planners can degrade if the robot's free space has narrow passages. We address this issue in more detail in Section 1.2.

1.1.4 Application to Virtual Prototyping

Besides robotics, motion planning techniques have also been used for many other applications. In this thesis, we also focus on virtual prototyping applications. Simulation technologies are increasingly used for virtual prototyping and PLM (product lifecycle management), where the goal is to provide efficient software solutions to problems that were traditionally solved using costly physical mockups. For instance, the simulation of assembly maintainability and mechanical part disassembly frequently arises in design and manufacturing applications. The manual generation of detailed disassembly or maintainability paths can be tedious and time consuming, particularly in environments prone to frequent design changes. The recent trend has been towards developing automated algorithmic solutions for such design problems that can automatically compute a collision-free, global path. Assembly maintainability and part disassembly can be reduced to motion planning of robots, where collision-free paths need to be computed for rigid objects with 6 DOF among static obstacles. Fig. 1.2 shows a scenario of disassembly of a seat outside of car body which arises in industry design (Ferr and Laumond, 2004).

1.2 Challenges and Goals

The major challenge in motion planning lies in capturing the connectivity of the robot's free space, which can be rather complex and high dimensional. Over the last decade

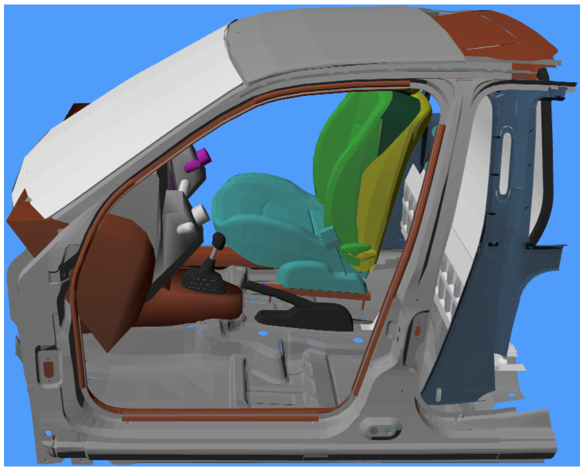


Figure 1.2: Disassembly of a seat outside a car body. *This disassembly scenario arises in industrial product design. Motion planning techniques can be applied to automatically compute a collision-free path for the seat (treated as a rigid robot) in the environment (the car body is treated as an obstacle). The major difficulties using sampling-based planners for such disassembly scenarios are due to the cluttered environment and complex geometric representation of the models, e.g.* $30K$ *triangles for the seat model and* $214K$ *triangles for the car body model.*

sampling-based planning approaches have been widely used for high dimensional motion planning problems. In this thesis, we focus on two important and unsolved problems in motion planning.

1.2.1 Path Non-existence Problem

Given the robot's initial and goal configurations, a *complete motion planning* algorithm computes a collision-free path if one exists; otherwise it reports path non-existence. Fig 1.3 shows challenging scenarios of a gear-shaped robot in a 2D rectangular region with two gear-shaped obstacles. A complete motion planner should be able to compute a path for the left scenario and determine path non-existence for the right scenario.

In terms of state of the art, there are no known practical complete motion planning approaches even for low 3-4 DOF robots. Earlier exact approaches are complete but their implementation are limited to low 2-3 DOF robots or special shapes such as spheres or ladders due to the high combinatorial complexity and robustness issues

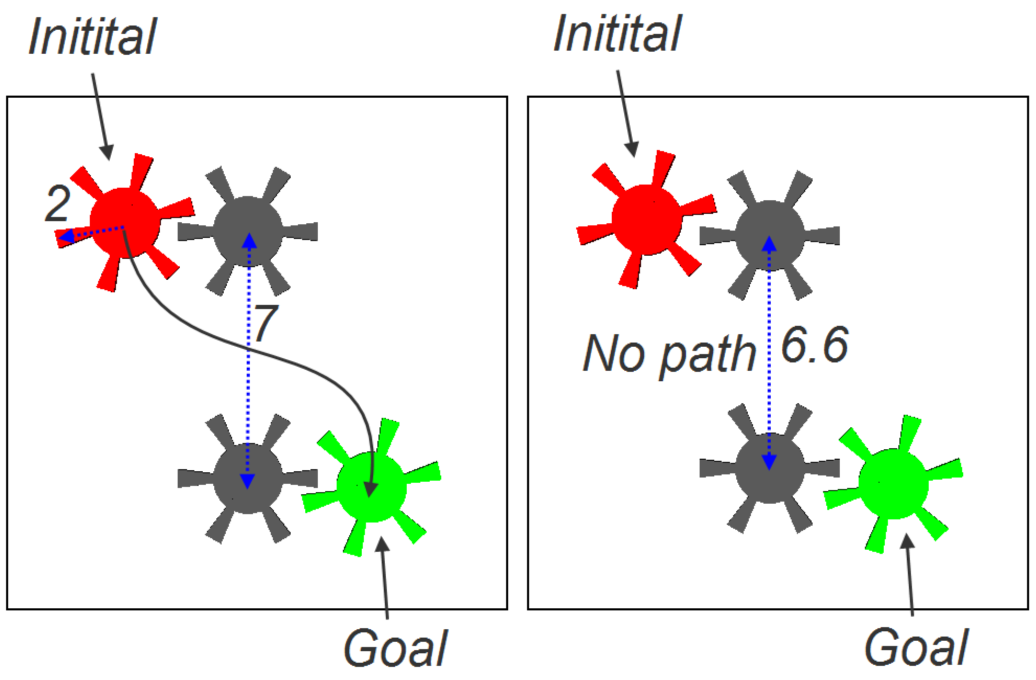


Figure 1.3: Complete motion planning. *A gear-shaped robot with* 2 *translational and* 1 *rotational DOF needs to move from its initial configuration to goal configuration. A complete motion planner should be able to compute a path for the left scenario and determine the path non-existence for the right scenario. These are interesting scenarios since even for humans it is not intuitive to check for the existence of paths in each scenario.*

of geometric computation (Avnaim and Boissonnat, 1989b). Approximate cell decomposition approaches are simple to implement. Provided the number of subdivisions is sufficiently high, these approaches can find a pth or report path non-existence in finite time. However, there are no known efficient cell labelling algorithms (i.e. to determine whether a cell lies entirely in the free space or C-obstacle), and the current techniques may be overly conservative (Brooks and Lozano-Pérez, 1985; Paden et al., 1989; Zhu and Latombe, 1990). Sampling-based randomized approaches may not terminate when no collision-free path exists in the free space. In practice, when such a planner does not terminate, it is hard to distinguish whether such situation arises due to path non-existence of the problem or due to inadequate sampling.

In practice, motion planners with the capability of checking for path non-existence can be useful for many applications such as verification in product design. For the scenario of disassembly of a seat of a car, the question arisen in design phase about whether the seat can be taken out of car body can be as important as the question of how to disassemble the seat. One goal in our work is to advance the state of the art and to design practical algorithms to check for path non-existence.

1.2.2 Narrow Passage Problem

The sampling-based approaches can solve motion planning for high DOF robots. However, the performance of these planners can degrade if the free space has narrow passages. The narrow passages are classified as regions, whose removal or perturbation can change the connectivity of the free space (Hsu et al., 1998). Figure 1.4 shows the well-known alpha puzzle benchmark, which is widely regarded as a challenging benchmark for motion planning algorithms. The problem is mainly caused by the small volume and poor visibility of narrow passages in free space. Because the volume is small, it may be difficult to generate adequate number of samples in these regions in the free space by performing uniform or randomized sampling in C-space. Furthermore, narrow passages may also exhibit the poor visibility, i.e. nearby samples are more difficulty to be connected by straight lines in the free space and more samples are needed to capture the overall connectivity in these regions.

The motion planning scenarios that arise in part removal and part disassembly simulations are rather challenging in terms of narrow passages (e.g. Fig. 1.2). Furthermore, the underlying models are complex and may be represented using thousands of polygons. Many times the models are given as polygon soup models which don't have connectivity or topology information. It is important for virtual prototyping applications that the motion planner should be able to handle such datasets automatically.

In order to address the issue of the narrow passages, a number of sampling strategies have been proposed, including dense sampling along obstacle boundaries (Amato

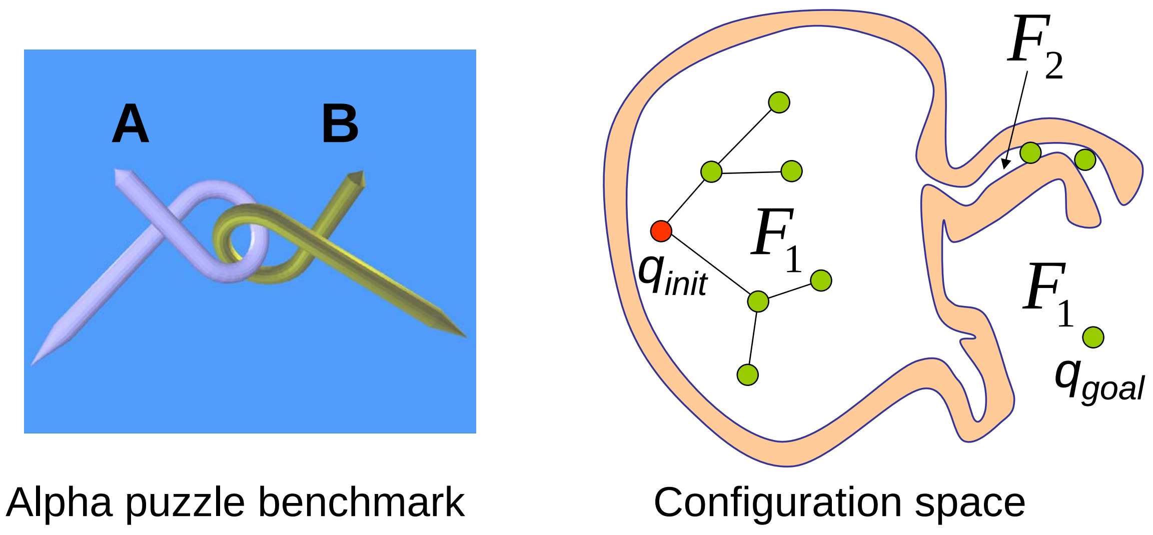


Figure 1.4: Narrow passage problem in alpha puzzle benchmark. *The goal is to separate the two intertwined alpha-shaped models. This problem reduces to the motion of the rigid robot A, while treating B as a static obstacle. The 6-dimensional free space of this problem is decomposed into two types of regions: F_1 and F_2 and this figure shows a 2-dimensional projection of those regions. F_1 is open and samples inside this region can be easily generated. F_2 is a narrow passage and the solution path lies in this narrow passage. It is difficult for sampling-based planners to generate and connect samples in such narrow passage due to its small volume and poor visibility.*

et al., 1998), medial axis-based sampling (Guibas et al., 1999), visibility-based techniques (Simeon et al., 2000), using workspace information (Kurniawati and Hsu, 2006), dilation of free space (Hsu et al., 1998), and using filtering strategies (Boor et al., 1999). In practice, it is still difficulty for these approaches to deal with challenging benchmarks such as alpha puzzle and the part disassembly scenarios. One goal of our work is to improve the performance of sampling-based planners for such challenging scenarios.

1.3 Relevance of Proximity Queries to Motion Planning

At a high-level, two different formulations for motion planning to model the underlying C-space have been used. Exact motion planning approaches explicitly compute a repre-

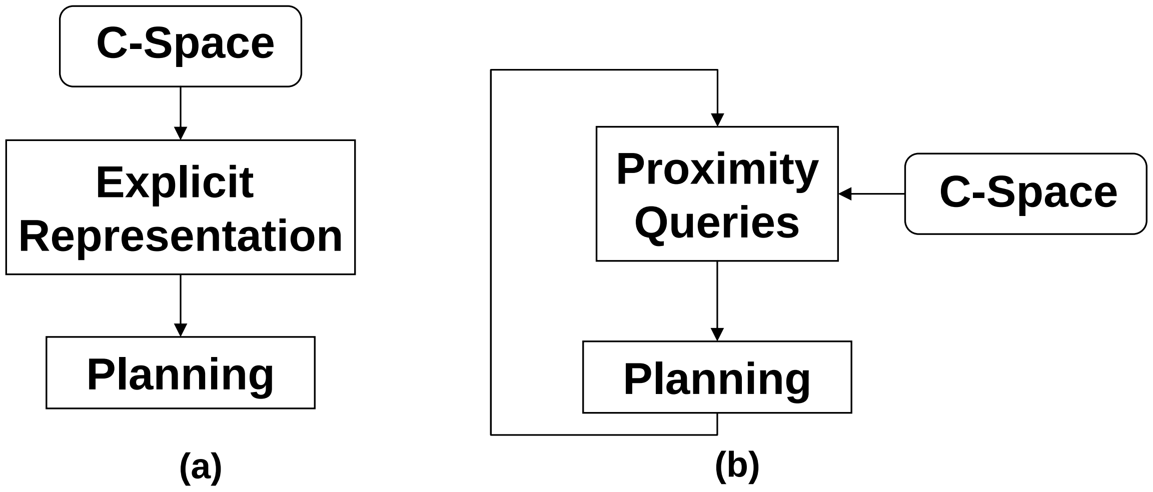


Figure 1.5: Two formulations of motion planning. *(a) Exact approaches explicitly compute the boundary of the free space and use it for planning. (b) Rather than explicit representation, other approaches including sampling-based motion planning use proximity queries such as collision detection to incrementally reason about C-space and conduct the search.*

sentation of the boundary of the free space of C-space, and other methods incrementally reason about C-space (Fig. 1.5). Due to the exponential complexity and robustness issues in computing the boundary, the exact approaches are difficult for robots with high DOF and most of their applications are limited to low 2-3 DOF robots.

On the other hand, by incrementally reasoning about robot's C-space via randomized sampling, sampling-based motion planning approaches have been successful and can solve many challenging and high DOF problems. As compared to the exact approaches, these approaches are easier to implement. Rather than computing an explicit representation of the free space, these approaches use proximity queries to determine spatial relationship among the robot and the obstacles for incrementally reasoning about the C-space and search for a feasible solution. One widely used query is collision detection, which determines whether a configuration lies in the free space or C-obstacle (Fig. 1.7). Efficient collision detection algorithms have been developed for handling complex geometric models (Lin and Manocha, 2003).

Approximate cell decomposition can also be categorized as an incremental approach.

C-space is decomposed into cells and the decomposition can be refined until a path is found or path non-existence is reported. The primitive query used by approximate cell decomposition is cell labelling, which determines whether a cell lies entirely in free space or C-obstacle. In this thesis, we refer to labelling a cell entirely in C-obstacle as *C-obstacle cell query* or often shortened as *C-obstacle query* (Fig. 1.7). Given that the time and space complexity of approximate cell decomposition methods grow quickly with the level of subdivision, it is important to identify cells that lie in C-obstacle space such that no further subdivision is performed for them. However, C-obstacle query is harder to be performed than collision detection because the query needs to assert that at every configuration over this cell, the robot intersects with some obstacle. There are no prior efficient algorithms to perform C-obstacle query (Zhang et al., 2008c).

1.3.1 Penetration Depth Computation

In this thesis, we explore other C-space queries along with collision detection to design better algorithms for motion planning. C-space queries often corresponds to proximity queries or determining the spatial relationship among geometric objects in the workspace. Most prior work on proximity queries is on collision detection. However, collision detection only returns a boolean answer and does not report quantitative information about the extent of separation or intersection. In this regard, separation distance, which is defined the minimum distance between models, has been used for quantifying the extent of separation. There are efficient algorithms to compute separation distance (Larsen et al., 1999).

Penetration depth is widely used for quantifying of the extent of intersection between models. Fig. 1.6 shows an example of the intersection between two models. Most prior work has been restricted to compute translational PD or the minimal translation needed to separate two intersecting rigid models. Translational PD is not sufficient for motion planning application as it does not take into account the rotational motion of the robot.

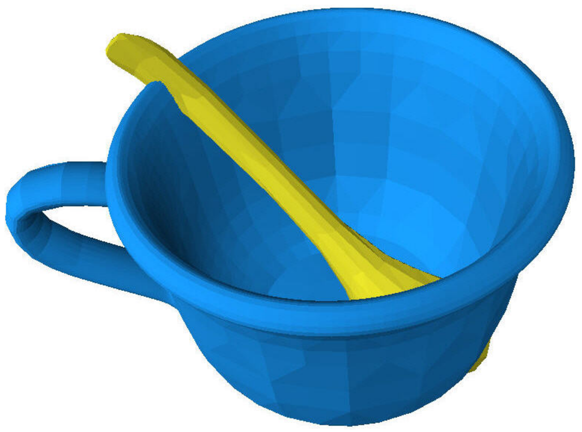


Figure 1.6: The problem of quantifying the extent of intersection (e.g. between the 'cup' and 'spoon' models) arises in motion planning, haptic rendering, dynamic simulation and tolerance verification.

In (Ong, 1997), a penetration measure is proposed that considers both translational and rotational motion. However, no efficient algorithms exist to compute this measure. In this thesis, we extend the notion of translational PD to generalized PD by taking into both translational and rotational motion as mentioned in (Ong, 1997). We present efficient algorithms to compute generalized PD between rigid models.

In terms of using penetration depth for motion planning, some authors have observed that penetration depth computation may be useful for improving the performance of sampling-based planners in narrow passages in earlier works such as (Hsu et al., 1998). However, due to the difficulty of penetration depth computation, the use of such algorithms has been limited.

1.4 Thesis Statement

Generalized penetration depth for quantifying the extent of intersection between rigid models can be efficiently computed and used to develop planning algorithms that can efficiently check for path non-existence for low DOF robots or compute collision-free paths in cluttered environments.

1.5 Main Results

In this thesis, we present efficient algorithms for proximity queries that are used to reason about the robot's configuration space. We show that generalized penetration depth is another important query in addition to collision detection, and can be used to infer the distance information in C-space, especially in C-obstacle. Based our generalized penetration depth computation and other C-space query algorithms, we design practical motion planning approaches: which are either complete for general low DOF robots, or are efficient in terms of solving difficult planning problems for rigid models with narrow passages. We apply our approach to part removal or disassembly problems in virtual prototyping and CAD.

1.5.1 Generalized Penetration Depth Computation

Most prior work has been restricted to compute translational PD or the minimal translation to separate two intersecting rigid models. We propose a formulation of generalized PD that takes into account the translational and rotational motion to describe the extent of intersection between the models (Zhang et al., 2007a,c). Generalized PD is defined as the minimal translational and rotational motion for the separation and is formulated using model-dependent C-space distance metrics. We present an efficient algorithm to compute a distance metric defined as the maximum displacement on the points of the model when it is transformed (Zhang et al., 2008b).

We present two new algorithms to compute generalized PD between rigid models. We first present a convexity-based algorithm using convex decomposition and containment optimization (Zhang et al., 2007c). We show that for two overlapping convex polytopes, generalized PD is equivalent to translational PD. Otherwise, when the complement of one of the objects is convex, we pose the generalized PD computation as a variant of the convex containment problem and compute an upper bound using optimization techniques. When both of the objects are non-convex, we treat them as a combination

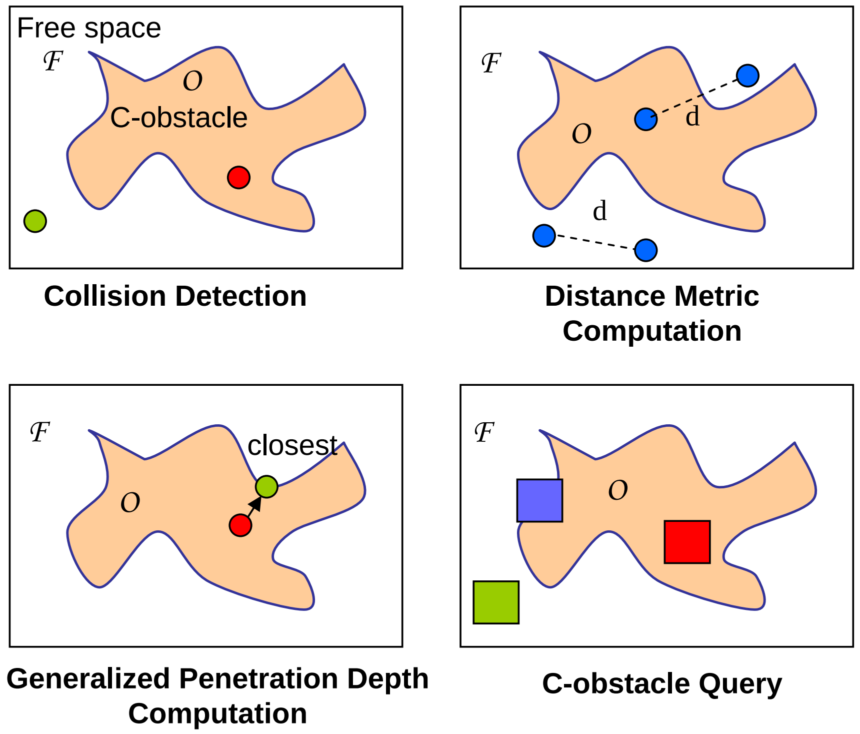


Figure 1.7: C-Space queries. *It is computationally prohibitive to explicitly compute the boundary of C-obstacle. Rather than explicitly computation, practical motion planning approaches use primitive queries to reason about C-space. Collision detection is one such query, which determines whether a configuration lies in C-obstacle or not. In this proposal, we show that other queries to C-space are also useful for motion planning. The queries are* C-space distance metric computation, generalized penetration depth computation *and* C-obstacle query. *We present efficient algorithms for these queries and use them for motion planning.*

of the above two cases and present algorithms that compute a lower bound and an upper bound on the generalized PD.

We further present a constrained optimization based algorithm to compute generalized PD (Zhang et al., 2007a). We use global approaches to find an initial guess for optimization and present efficient techniques to compute a local approximation of the contact space for iterative refinement. As compared to the convexity-based algorithm, this algorithm does not need to perform convex decomposition and can be applicable to rigid objects represented as polygonal soup models.

We highlight the efficiency and robustness of both algorithms on many complex 3D models.

1.5.2 Complete Motion Planning

We present a complete motion planning algorithm that can compute a path if it exists and report the path non-existence otherwise for low DOF robots among static obstacles (Zhang et al., 2008c). Our algorithm is based on approximate cell decomposition of C-space. We use C-obstacle cell query to check whether a cell lies entirely inside the C-obstacle. This reduces the problem of checking for path existence to checking whether there exists a path through the set of all cells that do not lie entirely inside C-obstacle. We present a simple and efficient algorithm to perform C-obstacle cell query using generalized PD computation. As compared to prior approaches, our algorithm is practical for 2-3 low DOF robots in complex environments. The algorithm is robust and simple to implement. Although our C-obstacle query algorithms are conservative, we prove that the overvall planner is still complete.

In order to further improve the performance of complete motion planning, we also present a hybrid approach that combines the completeness of approximate cell decomposition (ACD) with the efficiency of probabilistic roadmaps (PRM) (Zhang et al., 2007b). Our approach uses ACD to subdivide C-space into cells and computes localized roadmaps by generating samples within each of these cells. We have applied our approach to 3-4 DOF robots. In practice, we observe up to 10 times improvement in scenarios with narrow passages or no collision-free paths over our first complete motion planner.

Our complete motion approaches are also extended to feedback motion planning (Zhang et al., 2009). Our algorithm computes a global vector field in the entire free space as a feedback plan, which ensures that the robot at any collision-free configuration knows the direction to move in order to reach the goal. We compute a local vector field for each cell in the free space and address the issue of the smooth composition of the local vector fields between the non-uniform adjacent cells. As compared to prior approaches, our algorithm works well on non-convex robots and obstacles. We demonstrate its performance on planar robots with 2 or 3 DOF, articulated robots composed of 3 serial links and multi-robot systems with 6 DOF.

1.5.3 An Efficient Retraction-based Sampling Planner

We present a novel optimization-based retraction algorithm to improve the performance of sampling-based planners in narrow passages for 3D rigid robots (Zhang and Manocha, 2008b). The retraction step, which is equivalent to generalized PD computation, is formulated as a constrained optimization problem using an appropriate C-space distance metric. Our algorithm computes samples near the boundary of C-obstacle using local contact analysis and uses those samples to improve the performance of RRT planners in narrow passages. We analyze the performance of our planner using Voronoi diagrams and show that the generate tree tends to grow closely towards any randomly generated sample. Our algorithm is general and applicable to all polygonal models. In practice, we observe significant speedups over prior RRT planners on challenging scenarios with narrow passages.

We apply our retraction-based planning algorithm to part removal or disassembly problems, where a part usually needs to be extracted from a very cluttered environment (Zhang et al., 2008a). Based on retraction-based sampling, we show that our planner can handle complex CAD scenarios with narrow passages and composed of a few hundreds of thousands triangles.

In this thesis, we mainly focus on using our novel generalized PD algorithms between rigid models for efficient motion planning. In future, it is interesting to extend our generalized PD and motion planning algorithms to articulated robots. It is also interesting to apply the generalized PD algorithms to other applications, such as dynamic simulation and haptic rendering.

1.6 Thesis Organization

The rest of the thesis is organized into two parts. The first part (Chapters 2,3 and 4) addresses the problem of generalized PD computation and the second part (Chapters 5 and 6) focuses on designing efficient motion planning approaches using generalized PD computation. More specifically,

- **Chapter 2** introduces the problem of quantifying the extent of the intersection between overlapping objects and presents the notion and basic formulation of generalized PD computation;
- **Chapter 3** presents an efficient C-space distance computation algorithm for rigid or articulated robots;
- **Chapter 4** presents two algorithms for generalized PD computation: the convexity-based algorithm and the second algorithm based on constrained optimization;
- **Chapter 5** presents our approaches for complete motion planning and feedback motion planning for low DOF robots by using generalized PD computation;
- **Chapter 6** presents an efficient sampling-based planner using retraction strategy by performing generalized penetration depth computation. The chapter also demonstrates the applications of our planner to part disassembly simulation.
- **Chapter 7** concludes this thesis, discusses the limitations of our algorithms and suggests directions for future research.

Chapter 2

Notion and Formulation of Generalized Penetration Depth

Penetration depth (PD) is a distance measure for quantifying the extent of intersection between overlapping objects. Along with other proximity queries such as collision detection and separation distance, PD is useful for robot motion planning and many other applications, including physically-based simulation, haptics, and CAD/CAM. In these applications, the problem of quantifying the intersection arises frequently. When using computers to simulate objects in the physical world, physical constraints may often be violated, which can result in incorrect simulation results. For instance, the constraint of non-interpenetration between simulated objects may often be invalid, i.e. the objects may intersect with each other (e.g. Fig. 1.6). In these cases, an effective measure of the extent of intersection can be useful to restore the simulated objects to their valid positions. In robot motion planning, a robot needs to avoid collision with the obstacles in the environment. Collision detection is used to determine whether a configuration of the robot is collision-free or not. By further quantifying the extent of intersection for any invalid colliding configuration of the robot, PD computation can compute the distance for any colliding configuration to its nearest collision-free configuration. Such distance information in the invalid search space of the robot can be useful for designing efficient motion planning algorithms as shown in Chapters 5 and 6.

Most of the prior work on PD computation, however, has been restricted to *translational PD* (Cameron and Culley, 1986; Dobkin et al., 1993; van den Bergen, 2001; Kim et al., 2002a). Translational PD between two overlapping objects is often defined as the maximum translational distance needed to separate them. Translational PD computation is not sufficient for robot motion planning and other applications as it does not take the rotational motion into account. In this thesis, we extend the notion of translational PD to *generalized PD* by taking into account translational as well as rotational motion to separate the overlapping objects.

In this chapter, we first introduce the problem of penetration depth (PD). After a brief survey of previous work on PD computation, we present our notion of generalized PD. In the rest of this chapter, we focus on issues of choosing good distance metrics and formulating generalized PD based on these metrics. In the next two chapters, we present an algorithm for computing a meaningful distance metric and present two algorithms for generalized PD computation.

2.1 Penetration Depth: A Measure of Interpenetration between Intersecting Objects

There are many ways to quantify the extent of intersection. As shown in Fig. 2.1, given two intersecting objects A and B, one can calculate the volume of the intersection as a measure. One can also shrink the model A uniformly and compute the scale when the scaled model A' is barely touching B (Ong and Gilbert, 1996). Compared to these measures, penetration depth (PD) has been more widely used since it can more accurately measure the extent of intersection especially for non-convex models.

Most of the previous work on PD computation focuses on *translational PD*. For two overlapping objects A and B, translational PD is defined as the minimum translation to be applied to one of the objects A to separate it from the other one B.

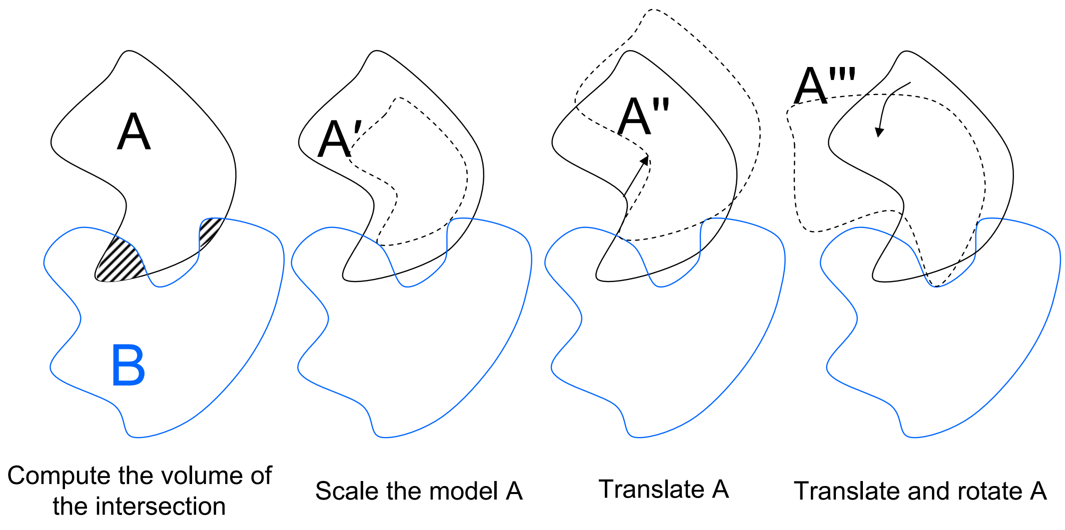


Figure 2.1: Different measures for quantifying the extent of intersection between two overlapping models A and B.

$$\mathrm{PD}^t(A,B) = \min\{\parallel \mathbf{d} \parallel \,|\, interior(A+\mathbf{d}) \cap B = \emptyset, \mathbf{d} \in \mathcal{R}^3\}. \tag{2.1}$$

Many good algorithms to compute translational PD between convex and non-convex polyhedra are known (van den Bergen, 2001; Kim et al., 2002b). Most these algorithms for computing translational PD are based on the formulation of Minkowski sum which is shown in detail in Fig. 2.2.

Despite the computational efficiency, translational PD computation is not sufficient for many applications as it does not take the rotational motion into account. One intuitive example is shown in Fig. 2.3, where the model A can be more 'easily' separated using both the translational and rotational motion as compared to only the translational motion. Therefore, in order to compute a more accurate measure of the extent of intersection, we need to take into account both translational and rotational motion during PD computation. In rigid body dynamics simulations, objects undergo both translational and rotational motion due to external forces and torques. In order to compute an accurate collision response, we also need to take into account rotational

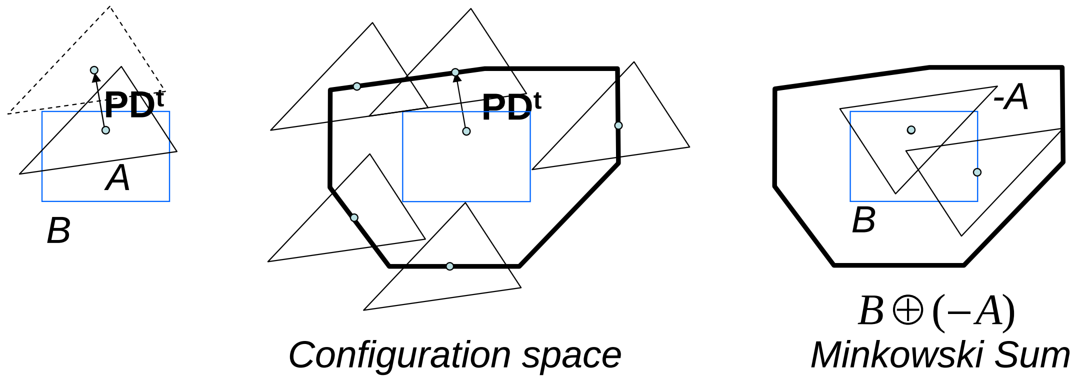


Figure 2.2: Translational penetration depth. *Translational penetration depth can be formulated using the notion of configuration space, where the object B is treated as the fixed obstacle and A as a robot, which can only translate. The contact space or the boundary of C-obstacle is defined as the locus of every configuration where the robot A is touching the obstacle B at its boundary. Translational PD (*PD^t*) is defined as the closest distance between the querying configuration of the robot to this boundary. C-obstacle can be formulated as a* Minkowski sum *operation between B and $-A$, i.e. $B \oplus (-A) = \{b - a, b \in B, a \in A\}$.*

motion. Similarly, in 6-DOF haptic rendering (McNeely et al., 1999), the rotational component in penalty forces, such as torque, should be considered in order to compute the response force. Also, in motion planning, since the configuration space of a rigid free-flying robot is 6-dimensional, it is necessary to consider rotational motion during PD computation.

2.2 Previous Work on Penetration Depth Computation

There has been considerable research work done on proximity queries including collision detection, separation distance, and PD computation between two or more objects in robotics, computer graphics and computational geometry (Lin and Manocha, 2003; Ericson, 2004). In this section, we briefly discuss prior approaches to PD computation.

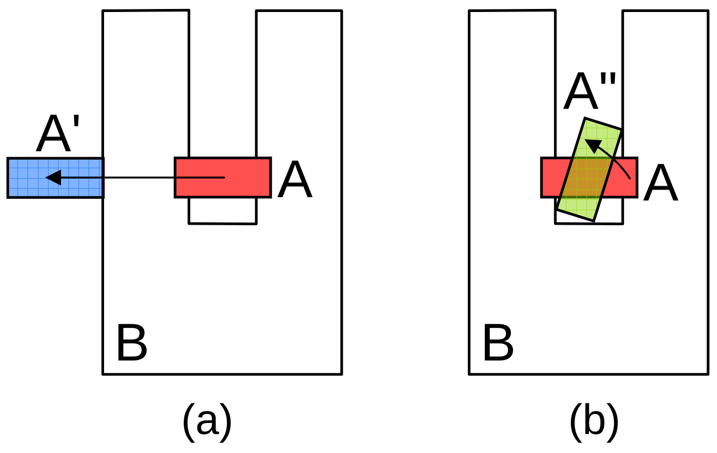


Figure 2.3: Separating intersecting models. *In this example, it is 'easier' to separate two intersecting models A and B by both translational and rotational motion (from A to A'') as compared to translational motion (from A to A'). Therefore, it is more accurate to measure the extent of intersection by taking into account both translational and rotational motion.*

2.2.1 Translational Penetration Depth

Translational PD can be formulated in terms of the *Minkowski sum* of two objects, and the computation of PD^t is equivalent to finding a nearest point on the surface of the Minkowski sum to the origin (Dobkin et al., 1993). Several algorithms have been proposed for exact or approximate computation of PD^t. van den Bergen (2001) proposes a quick lower bound estimation to PD^t between two convex polytopes by iteratively expanding a polyhedral approximation of the *Minkowski sum*. Kim et al. (2002a) present an incremental algorithm to estimate a tight upper bound on PD^t between convex polytopes by walking to a "locally optimal solution". They have also presented an algorithm to compute an approximation of global PD^t between two general polyhedral models by using hierarchical refinement (2002b). The running time for the best known theoretical algorithm to compute PD^t between convex polytopes is $O(n_1^{\frac{3}{4}+\epsilon} n_2^{\frac{3}{4}+\epsilon} + n_1^{1+\epsilon} + n_2^{1+\epsilon})$ for any positive constant ϵ, where n_1 and n_2 denote the number of features in the two polytopes (Agarwal et al., 2000). However, we are not aware of any implementation of this algorithm. In the case of general non-convex polyhedral models, the computational complexity of solving PD^t can be $O(n^6)$, where n is the number of features in the polytopes. Due to the difficulty of computing a global PD^t between non-convex models, some fast local PD^t algorithms using graphics hardware have been proposed (Redon and Lin, 2005; Sud et al., 2006).

2.2.2 Considering both Translational and Rotational Motion

Only a few authors have addressed the problem of generalizing the measure of penetration by taking into account both translational and rotational motion. In the literature, Ong's work (1997) can be considered as one of the earliest attempts, and has shown that, in the case of convex objects, the measure is equivalent to the translational one. A new distance measure based on scaling of a model, namely *growth distance*, has also been presented (Ong and Gilbert, 1996). The growth distance can unify the notion of separation and penetration distances and can be efficiently computed for convex objects.

A class of closely related work to the generalized PD computation are the polygon containment algorithms (Chazelle, 1983; Avnaim and Boissonnat, 1989a; Grinde and Cavalier, 1996; Agarwal et al., 1998; Milenkovic, 1999) and rotational overlapping minimization algorithms (Milenkovic, 1998; Milenkovic and Schmidl, 2001), if we view the problem of separating the object A from B as placing A into $\bar{B}$ - the complement space of B. The standard 2D polygon containment problem is to check whether a polygon Q with n_1 vertices can contain another polygon P with n_2 vertices. For general non-convex polygons, the time complexity of this problem is $O(n_1^3 n_2^3 log(n_1 n_2))$ (Avnaim and Boissonnat, 1989a). When restricted to convex objects, the time complexity of the 2D containment problem can be significantly improved. Chazelle (1983) proposes an enumerative algorithm with an $O(n_1 n_2^2)$ time complexity. Milenkovic (1999) and Grinde and Cavalier (1996) use mathematical programming techniques to compute an optimal solution.

Given an overlapping layout of polygons inside a container polygon, the rotational overlapping minimization problem is to compute the translational and rotational motion to minimize their overlap. This problem is solved by mathematical programming methods (Milenkovic, 1998). By using the non-overlapping property as a hard constraint, Milenkovic and Schmidl (2001) minimize a quadratic function of the position and orientation of objects to compute a non-overlapping layout based on quadratic programming.

2.3 Generalized Penetration Depth

We extend the notion of translational PD by taking into account translational as well as rotational motion to separate the overlapping polyhedral models namely, *generalized penetration depth* (generalized PD) (Zhang et al., 2007a,c).

2.3.1 Notation

We first introduce some terms and notation used throughout the rest of this section. We use a bold face letter, such as the origin $\mathbf{o}$, to distinguish a vector quantity from a scalar quantity. We use a sextuple (x, y, z, ϕ, θ, ψ) to encode the 6-dimensional *configuration* of a 3D rigid object, where x, y and z represent the translational component, and ϕ, θ and ψ represent Euler angles for the rotational component. The rotational component can be also represented as a rotation vector $r = (r_1, r_2, r_3)^T = \alpha\hat{\mathbf{a}}$, where α is the rotation angle and $\hat{\mathbf{a}}$ is the rotation vector. $A(\mathbf{q})$ is a placement of an object A at configuration $\mathbf{q}$, and $\mathbf{p}(\mathbf{q})$ is the corresponding position of a point $\mathbf{p}$ on A.

2.3.2 Definition

Given two overlapping objects A and B, we assume B is fixed and A is movable, i.e. A is treated as a robot and B is treated as an obstacle. When A at a configuration $\mathbf{q_o}$ intersects with B, their generalized penetration depth is defined as:

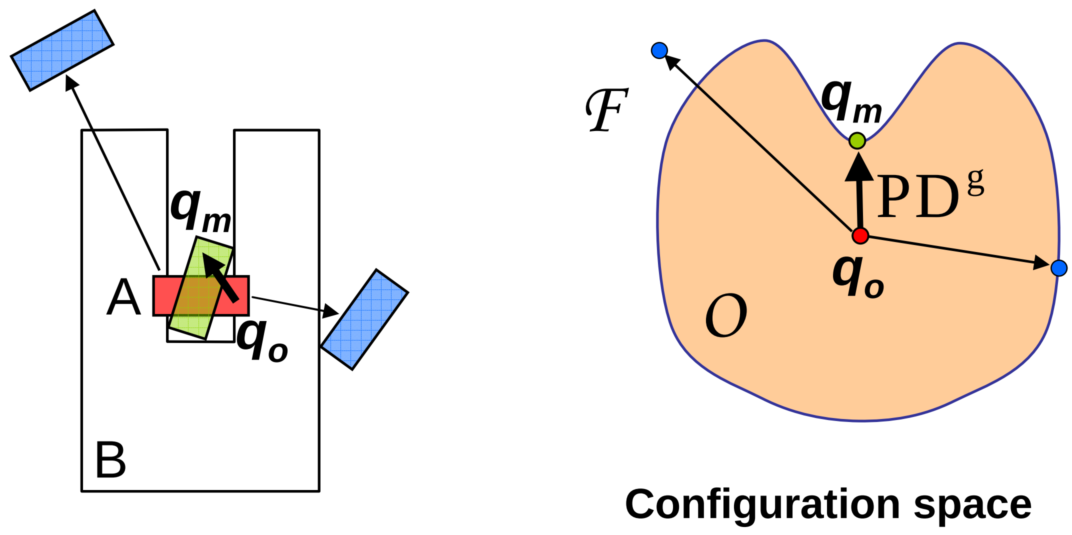


Figure 2.4: Generalized penetration depth. *(a) Generalized PD (*PD^g*) is a distance measure that quantifies the extent of intersection between two overlapping objects (e.g. A at* $\mathbf{q_o}$ *and B). Intuitively,* PD^g *is defined as the "minimal translational and rotational motion" to separate A from B (e.g. moving A from* $\mathbf{q_o}$ *to* $\mathbf{q_m}$*). (b) In configuration space,* PD^g *is defined as the minimal distance from the colliding configuration* $\mathbf{q_o}$ *of A to non-colliding (collision-free or contact) configurations, with respect to a distance metric.*

$$\mathrm{PD}^g_\delta(A,B) = min\{\delta(\mathbf{q_o},\mathbf{q}) \,|\, \mathrm{interior}(A(\mathbf{q})) \cap B = \emptyset, \mathbf{q} \in \mathcal{C}\}, \tag{2.2}$$

where δ is any distance metric in C-space and the computation of generalized PD (shortened as PD$^g_\delta$) is equivalent to minimizing a distance under δ metric, constrained by non-interpenetration between A and B. Fig. 2.4 gives an intuitive illustration of the notion of generalized PD. For a given colliding configuration $\mathbf{q_o}$, the goal is to compute the shortest distance to non-colliding (collision-free or contact) configurations.

Translational PD is a special case of generalized PD. When the model A can only translate, we can choose the Euclidean metric for δ. In this case, the computation of generalized PD reduces to the translational PD defined in Eq. (2.1).

For rigid robots, our notion on generalized PD takes into account the translational and rotational motion to describe the extent of intersection as the chosen metric can measure the distance in SE(3). Similarly, with a distance metric defined for an articulated model, the notion of generalized PD is also applicable to quantify the intersection between an articulated robot and the environment.

As we can see, one important issue on defining and formulating generalized PD is on choosing an appropriate metric. We address this issue in detail in the next section.

2.4 Distance Metrics in Configuration Space

Although any distance metric in C-space can be used to define generalized PD, we desire good mathematical properties with the chosen metric. In this section, we address the issue of choosing an appropriate distance metric for generalized PD. The problem of choosing good distance metrics also arises in many other applications and the choice of the metric can affect the performance of the overall algorithms. For instance, in sampling-based motion planning algorithms (Kavraki et al., 1996; Kuffner and LaValle, 2000; LaValle, 2006), a distance metric is used to determine nearby samples which need to be connected. The choice of distance metric can affect the connectivity of the roadmap and the performance of planners (Amato et al., 2000; Kuffner, 2004). The distance metric in configuration space is also used to evaluate the properties of generated samples (e.g. dispersion) in the planning algorithm (LaValle, 2006).

Specifically, a distance metric in C-space is used to compute a measure of distance between two configurations in C-space of a rigid or articulated object. Intuitively speaking, for a rigid object, the measure of a distance in configuration space under an appropriate distance metric can be used to quantify how far this object is displaced from one configuration to another. Defining and calculating the distance for a rigid object undergoing only translational motion is relatively easy, because the well-defined Euclidean distance metric can be employed in this case. However, it is harder to just define a distance metric for a rigid object undergoing both translational and rotational motion (Zhang et al., 2008b). The main challenge is how to combine the translational and rotational components naturally, such that the metric is *bi-invariant* with the choice of the inertial and body-fixed reference frames for the object, and is independent of the representation of the configuration space (Park, 1995). We can classify most distance metrics into model-independent and model-dependent based on whether or not the shape of an object is considered in defining distance metrics. It is well-known that *model-independent* metrics are not bi-invariant, and thus most approaches use *model-dependent metrics* for proximity computations (Latombe, 1991; Lin and Burdick, 2000; Zhang et al., 2008b). Moreover, for practical applications, it often requires that the distance metric can be efficiently computed (Plaku and Kavraki, 2006).

Different distance metrics have been proposed for rigid and articulated models, in particular in the area of robotics and kinematics (Latombe, 1991; Tchon and Duleba, 1994; Park, 1995; Lin and Burdick, 2000). We mainly consider three distance metrics for our generalized PD computation due to their good mathematical properties. These distance metrics are: DISP metric (Latombe, 1991) and *object norm* - OBNO metric (Kazerounian and Rastegar, 1992), which are based on the displacements of the points on the model when it moves, and TRAJ metric (Zhang et al., 2007c), which is based on the length of the trajectory traced by the points on the moving model.

2.4.1 Distance Metrics in SE(3)

The spatial rigid body displacements form a group of rigid body motion, SE(3). Throughout the rest of the thesis, we will refer to a model-independent distance metric in SE(3) as a distance metric in SE(3). The L_p ($p \geq 1$) weighted sum metrics have been widely used for SE(3) of rigid objects (LaValle, 2006). In theory, there is no natural choice for distance metrics in SE(3) (Loncaric, 1987; Park, 1995). Loncaric (1985) shows that there is no bi-invariant Riemannian metric in SE(3).

2.4.2 Model-dependent Distance Metrics: DISP, OBNO and TRAJ

In the problems of generalized PD computation and robot motion planning, since models of robots are usually known in advance, one define a model-dependent distance metric. The notation of *displacement vector* or *trajectory length* of any point of the moving model can be used to defined a meaningful metric (Latombe, 1991; Kazerounian and Rastegar, 1992; Zhang et al., 2007c).

Using the notion of a *displacement vector* for each point in the model, the DISP distance metric is defined as the maximum length over all the displacement vectors (Latombe, 1991; LaValle, 2006; Zhang et al., 2008b). The object norm, proposed by (Kazerounian and Rastegar, 1992), is defined as an average squared length of all displacement vectors. Hofer and Pottmann (2004) use a similar metric, but consider only a set of feature points in the model. All of these displacement vector-based metrics can be efficiently evaluated. The length of a trajectory travelled by a point on a moving model can be also used to define model-dependent metrics (Hsu et al., 1999; Zhang et al., 2007c). However, it is difficult to compute the exact value of these metrics.

DISP Metric

The displacement metric or shorten as DISP is defined by the maximum Euclidean displacement for all of the points of a model when the model is placed at two different configurations (Latombe, 1991; LaValle, 2006; Zhang et al., 2008b) (Fig. 2.5):

$$\mathrm{DISP}(\mathbf{q_0}, \mathbf{q_1}) = \max_{\mathbf{p} \in A} ||\mathbf{p}(\mathbf{q_1}) - \mathbf{p}(\mathbf{q_0})||. \tag{2.3}$$

This distance metric can naturally combine translational and rotational motion without relying on any weighting factor. Additionally, it is invariant with the choices of reference frames, and is independent of the representation of the underlying configura-

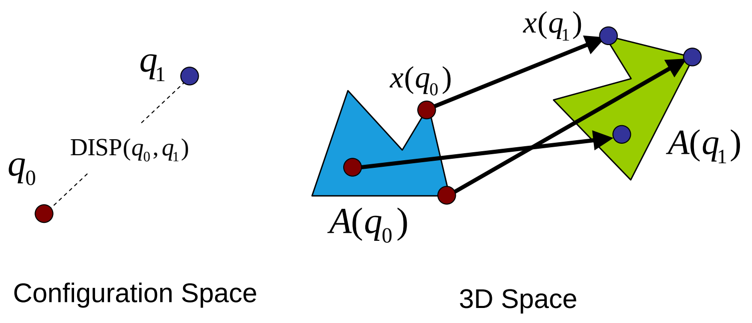


Figure 2.5: DISP metric: *The* DISP *distance of a model A between two configurations* $\mathbf{q_0}$ *and* $\mathbf{q_1}$ *in configuration space is defined as the maximum length of the displacement vector between* $\mathbf{p}(\mathbf{q_0})$ *and* $\mathbf{p}(\mathbf{q_1})$*, where* $\mathbf{p}$ *is any point on A (Latombe, 1991).*

tion space. However, to the best of our knowledge, no prior algorithms are known to efficiently compute DISP for complex models.

OBNO **Metric - Object Norm**

Also based on displacement vectors, Kazerounian and Rastegar (1992) make use of an integral operator to define the *object norm*:

$$\sigma(\mathbf{q_0}, \mathbf{q_1}) = \frac{1}{V}\int_A \rho(\mathbf{p})||\mathbf{p}(\mathbf{q_1}) - \mathbf{p}(\mathbf{q_0})||^2\, dV, \tag{2.4}$$

where V is the volume of A and $\rho(\mathbf{p})$ is the mass distribution function.

By using a quaternion representation, we can simplify the formula originally derived by Kazerounian and Rastegar (1992) into:

$$\sigma_A(\mathbf{q_0}, \mathbf{q_1}) = \frac{4}{V}(I_{xx}b^2 + I_{yy}c^2 + I_{zz}d^2) + x^2 + y^2 + z^2, \tag{2.5}$$

where $diag(I_{xx}, I_{yy}, I_{zz})$ forms a diagonal matrix computed by diagonalizing the inertia matrix I of A. (a, b, c, d) is the quaternion for the **relative** orientation of A between $\mathbf{q_0}$ and $\mathbf{q_1}$, and (x, y, z) is the relative translation.

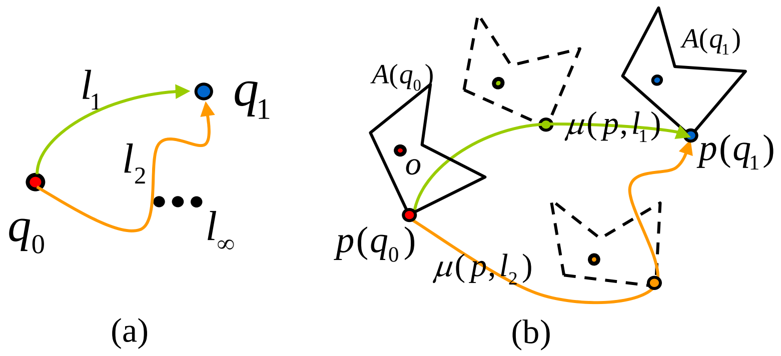


Figure 2.6: TRAJ Distance Metric. *(a) There are an infinite number of curves, such as l_1, l_2, that connect two configurations $\mathbf{q}_0$ and $\mathbf{q}_1$ in C-space. (b) When the configuration of the object A moves along any curve l, any given point on A will trace out a distinctive trajectory in the 3D Euclidean space. This sub-figure shows the trajectories traced by $\mathbf{p}$ when A travels along l_1 and along l_2, while $\mu(\mathbf{p}, l_1)$ and $\mu(\mathbf{p}, l_2)$ are the arc-lengths of these trajectories, respectively. For each curve l, some point on A corresponds to the longest trajectory length as compared to all other points on A. The distance metric* TRAJ$(\mathbf{q}_0, \mathbf{q}_1)$ *is defined as the minimum over longest trajectory lengths over all curves connecting $\mathbf{q}_0$ and $\mathbf{q}_1$.*

TRAJ **Metric**

Let l_i be a curve in C-space, which connects two configurations $\mathbf{q_0}$ and $\mathbf{q_1}$ (Fig. 2.6(a)) and is parameterized in t. When the configuration of A changes along the curve l, any point $\mathbf{p}$ on A will trace out a trajectory in 3D Euclidean space shown in Fig. 2.6(b). Let this trajectory as $r = \mathbf{p}(l(t))$. Its arc-length $\mu(\mathbf{p}, l)$, which is denoted as *trajectory length*, can be calculated by:

$$\mu(\mathbf{p}, l) = \int ||\dot{\mathbf{p}}(l(t))||\, d(t).$$

As Fig. 2.6(a) shows, there can be multiple curves connecting two configurations $\mathbf{q_0}$ and $\mathbf{q_1}$. When A moves along any such curve, some point on A corresponds to the longest *trajectory length* as compared to all other points on A. For each C-space curve connecting $\mathbf{q_0}$ and $\mathbf{q_1}$, we consider the corresponding longest *trajectory length*. We define the distance metric TRAJ$(\mathbf{q_0}, \mathbf{q_1})$ as the minimum over all longest *trajectory lengths* (Fig. 2.6(b)):

$$\text{TRAJ}(\mathbf{q}_0, \mathbf{q}_1) = min(\{max(\{\mu(\mathbf{p}, l) | \mathbf{p} \in A\}) | l \in L\}), \tag{2.6}$$

where L is a set of all of the curves in C-space connecting $\mathbf{q_0}$ and $\mathbf{q_1}$.

In general, TRAJ metric is difficult to compute. We compute a lower bound and an upper bound for TRAJ metric.

Lower bound on TRAJ($\mathbf{q_0}, \mathbf{q_1}$)**.** DISP($\mathbf{q_0}, \mathbf{q_1}$) is a lower bound of TRAJ($\mathbf{q_0}, \mathbf{q_1}$):

$$\text{TRAJ}(\mathbf{q_0}, \mathbf{q_1}) \geq \text{DISP}(\mathbf{q_0}, \mathbf{q_1}).$$

This holds since the length of any curve between two points in 3D Euclidean space is larger than or equal to the length of the displacement vector between the two points.

Upper bound on TRAJ($\mathbf{q_0}, \mathbf{q_1}$)**.** In order to compute an upper bound for a 3D rigid object with translational and rotational DOF, we first consider computing the TRAJ by only varying a single DOF. Then, when we vary all the DOF simultaneously, the final TRAJ would be less than or equal to the sum of the TRAJ's computed with respect to each DOF (Schwarzer et al., 2005; LaValle, 2006).

When Euler angles are used to represent rotation, the upper bound on TRAJ can be calculated as:

$$\text{TRAJ}(\mathbf{q_0}, \mathbf{q_1}) \leq \Delta_x + \Delta_y + \Delta_z + R_\phi \Delta_\phi + R_\theta \Delta_\theta + R_\psi \Delta_\psi, \tag{2.7}$$

where the Lipshitz constants R_ϕ, R_θ, and R_ψ are the maximum Euclidean distances from any point on A to X, Y, and Z axes in the local coordinate system, respectively. Δ denotes the absolute value of the difference of each DOF (x, y, z and Euler angles ϕ, θ, ψ) between these two configurations.

If the rotation vector is used to represent rotation, the upper bound can be calculated

Distance Metrics	Weighted Sum	DISP	TRAJ	Object Norm
Weights required	Yes[1]	No	No	No
Model-independent	Yes[2]	No	No	No
Computational complexity	Constant	Linear[3]	Expensive [4]	Constant
Extendable to articulated models	Yes	Yes	Yes	Yes

Table 2.1: Comparison of distance metrics for rigid models. [1] *The major disadvantage for weighted sum metrics lies in the difficulty for users to choose weights to combine rotational and translational components.* [2] *The radius of the model may be used for choosing weights. Then the metric is classified as model-dependent.* [3] *we propose an efficient method to compute* DISP *using its convexity property and bounding volume hierarchy in Chapter 3.* [4] *Computing the exact value of this metric is difficult. Its upper bound and lower bound can be computed in constant time.*

by:

$$\mathrm{TRAJ}(\mathbf{q_0}, \mathbf{q_1}) \leq \Delta_x + \Delta_y + \Delta_z + R \sum_{k=1}^{3} \Delta r_k, \tag{2.8}$$

where the constant R is the maximum Euclidean distance from the origin of A to every point on A. In Chapter 4, we use these two upper bound formulae to compute an upper bound on generalized PD.

2.4.3 Metric Properties

DISP, OBNO and TRAJ can combine the translational and rotational components without relying on the choice of any weighting factor. We highlight other mathematical properties of these metrics.

Properties of Metric Space. The distance metrics DISP, OBNO and TRAJ have the following properties (LaValle, 2006):

- **Non-negativity:** $\delta(\mathbf{q_0}, \mathbf{q_1}) \geq 0$**,**
- **Reflexivity:** $\delta(\mathbf{q_0}, \mathbf{q_1}) = 0 \iff \mathbf{q_0} = \mathbf{q_1}$**,**
- **Symmetry:**$\delta(\mathbf{q_0}, \mathbf{q_1}) = \delta(\mathbf{q_1}, \mathbf{q_0})$,
- **Triangle inequality:** $\delta(\mathbf{q_0}, \mathbf{q_1}) + \delta(\mathbf{q_1}, \mathbf{q_2}) \geq \delta(\mathbf{q_0}.\mathbf{q_2})$**.**

Most properties follow from the definition of metrics. We only prove the triangle inequality property for DISP metric.

Proof. Suppose the point $\mathbf{p}$ on A has the maximum displacement given as DISP($\mathbf{q_0}, \mathbf{q_2}$). In other words, $||\mathbf{p}(\mathbf{q_0}) - \mathbf{p}(\mathbf{q_2})|| = \text{DISP}(\mathbf{q_0}, \mathbf{q_2})$. In the Euclidean space, every point on A satisfies the triangle inequality, so does the point $\mathbf{p}$. Therefore, $||\mathbf{p}(\mathbf{q_0}) - \mathbf{p}(\mathbf{q_1})||$ $+ \ ||\mathbf{p}(\mathbf{q_1}) - \mathbf{p}(\mathbf{q_2})|| \geq \text{DISP}(\mathbf{q_0}, \mathbf{q_2})$. Since $\text{DISP}(\mathbf{q_0}, \mathbf{q_1}) \geq ||\mathbf{p}(\mathbf{q_0}) - \mathbf{p}(\mathbf{q_1})||$ and $\text{DISP}(\mathbf{q_1}, \mathbf{q_2}) \geq ||\mathbf{p}(\mathbf{q_1}) - \mathbf{p}(\mathbf{q_2})||$, $\text{DISP}(\mathbf{q_0}, \mathbf{q_1}) + \text{DISP}(\mathbf{q_1}, \mathbf{q_2}) \geq \text{DISP}(\mathbf{q_0}.\mathbf{q_2})$. □

As a result, the configuration space $\mathcal{C}$ with DISP is a metric space, and algorithms that are based on the properties of a metric space are also applicable to $\mathcal{C}$.

Invariance Properties. Since these metrics are defined by using displacement vectors or trajectory lengths, they have some invariance properties:

- **Invariance of reference frames:** DISP, OBNO, or TRAJ is independent of the choice of inertial reference frame and body-fixed reference frame (Lin and Burdick, 2000). Regardless of the choice of the frames, the distance defined by DISP, OBNO, or TRAJ for a model between two configurations does not change.

- **Independence of configuration space representation:** DISP, OBNO, or TRAJ is independent of the underlying representation of the configuration space. In case of a rigid model, there are many choices to represent the rotational degrees of freedom such as Euler angles, quaternions, or transformation matrices. The distance computed using DISP, OBNO, or TRAJ is independent of these representations, as well.

These invariance properties hold, since in the Euclidean space the length of the displacement vector of any point on A or the trajectory length is invariant of the choice of reference frames and independent of the configuration space representation. Due to the underlying distance metric, $\mathrm{PD}^g_{\mathrm{DISP}}$, $\mathrm{PD}^g_{\mathrm{OBNO}}$ or $\mathrm{PD}^g_{\mathrm{TRAJ}}$ is independent of the choice of inertial and body-fixed reference frames. In practice, these invariance properties are useful since one can choose arbitrary reference frames and representation of the configuration space to compute these metrics and their generalize PD.

Translational PD defined by Eq. (2.1) is essentially a special case of $\mathrm{PD}^g_{\mathrm{DISP}}$, $\mathrm{PD}^g_{\mathrm{OBNO}}$ or $\mathrm{PD}^g_{\mathrm{TRAJ}}$. When an object A can only translate, all the points on A traverse the same distance and displace in the same amount. As a result, the distance $\mathrm{DISP}(\mathbf{q_0}, \mathbf{q_1})$ (OBNO or TRAJ) is equal to the Euclidean distance $\| \mathbf{q_1} - \mathbf{q_0} \|$. In this case, Eq. (2.2) can be simplified to Eq. (2.1).

DISP and OBNO metrics can be computed efficiently. In (Zhang et al., 2008b), we show that for a rigid model, the DISP distance is realized by a vertex on its convex hull. This leads to an efficient algorithm, C-DIST, to compute DISP. In Table 2.1, we summarize these distance metrics.

2.5 Formulation of Generalized Penetration Depth

We formulate the computation of generalized PD which is defined by Eq. (2.2) using an appropriate metric such as DISP, OBNO or TRAJ. According to this equation, generalized PD can be formulated as an optimization problem under the non-interpenetration constraint, where the optimization objective is described by the metric. In general, generalized PD computation is difficult since it involves non-linear optimization due to the non-linear rotational term embedded in the definition. Furthermore, generalized PD computation has high combinatorial complexity due to non-convex geometric models which we need to handle.

In this section, we present the contact space realization property for generalized PD computation. We prove that the optimal configuration realizing generalized PD using each of our chosen metrics must lie in the contact space. We discuss the combinato-

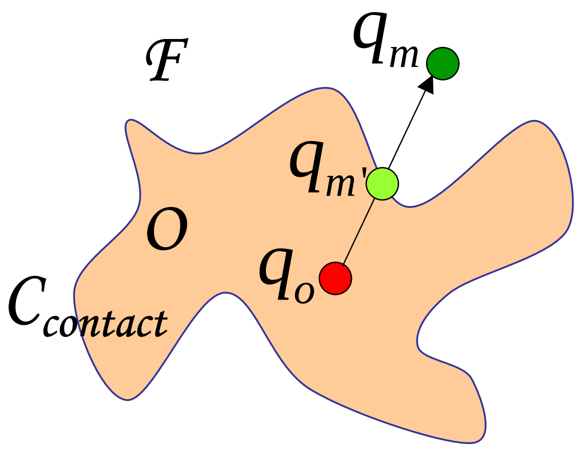


Figure 2.7: Contact space realization of generalized PD. *The optimal configuration* $\mathbf{q_m}$*, which realizes* $\mathrm{PD}^g_{\mathrm{DISP}}$ *(or* $\mathrm{PD}^g_{\mathrm{OBNO}}$*,* $\mathrm{PD}^g_{\mathrm{TRAJ}}$*), must be in the contact space* $\mathcal{C}_{contact}$*; otherwise, one can compute the contact configuration* $\mathbf{q_m}'$*, which further reduces the metric function.*

rial complexity of the contact space for polyhedral models, which governs the overall complexity of generalized PD computation. In Chapter 4, we present two algorithms to compute the generalized PD either exploring the convexity of the geometric models or using the formulation of constrained optimization.

2.5.1 Contact Space Realization

For rigid models, $\mathrm{PD}^g_{\mathrm{DISP}}$ ($\mathrm{PD}^g_{\mathrm{OBNO}}$ or $\mathrm{PD}^g_{\mathrm{TRAJ}}$) has a contact space realization property. This property implies that any non-colliding configuration $\mathbf{q_m}$ that minimizes the objective DISP (OBNO or TRAJ) for PD^g must lie in the contact space of A and B, or equivalently, at this configuration $\mathbf{q_m}$, A and B barely touch with each other.

Theorem 1. (Contact Space Realization) *For a rigid model A placed at* $\mathbf{q_o}$*, and a rigid model B, if* $\mathrm{interior}(A(\mathbf{q_m})) \cap B = \emptyset$ *and* $\mathrm{DISP}(\mathbf{q_o}, \mathbf{q_m}) = \mathrm{PD}^g_{\mathrm{DISP}}(A, B)$*, then* $\mathbf{q_m} \in \mathcal{C}_{contact}$*. This property also holds for* $\mathrm{PD}^g_{\mathrm{OBNO}}$ *and* $\mathrm{PD}^g_{\mathrm{TRAJ}}$*.*

Proof. We prove it by contradiction. Suppose the optimal configuration $\mathbf{q_m}$ realizing $\mathrm{PD}^g_{\mathrm{DISP}}$ does not lie in the contact space $\mathcal{C}_{contact}$. Then, $\mathbf{q_m}$ must lie in the free space $\mathcal{F}$

(Fig. 3.1(b)).

We use Chasles' theorem in Screw theory (Murray et al., 1994), which states that a rigid body transformation between any two configurations can be realized by rotation about an axis followed by translation parallel to that axis, where the amount of rotation is within $[0, \pi]$. The screw motion is a curve in C-space, and we denote that curve between $\mathbf{q_o}$ to $\mathbf{q_m}$ as $s(t)$, where $s(0) = \mathbf{q_o}$ and $s(1) = \mathbf{q_m}$. Since $\mathbf{q_o}$ is in $\mathcal{O}$, and $\mathbf{q_m}$ is in $\mathcal{F}$, there is at least one intersection $s(t'), t' \in (0, 1)$ between the curve $s(t)$ and the contact space (Fig. 3.1). We denote the intersection point as $\mathbf{q_m}'$.

Based on Chasles theorem, we can compute the length of the displacement vector for any point $\mathbf{p}$ on A between $\mathbf{q_o}$ and any configuration on the screw motion $s(t)$. Furthermore, we can show that this length strictly increases with the parameter t. Therefore, for each point on A, the length of the displacement vector between $\mathbf{q_o}$ and $\mathbf{q_m}$ is less than the one between $\mathbf{q_o}$ and $\mathbf{q_m}'$. Since DISP metric uses the maximum operator for the length of the displacement vector over all points on A, we can infer that $\text{DISP}(\mathbf{q_o}, \mathbf{q_m}') < \text{DISP}(\mathbf{q_o}, \mathbf{q_m})$. This contradicts our assumption that $\mathbf{q_m}$ is the realization for $\text{PD}^g_{\text{DISP}}$.

Similarly, we can infer $\text{OBNO}(\mathbf{q_o}, \mathbf{q_m}') < \text{OBNO}(\mathbf{q_o}, \mathbf{q_m})$, and thus prove the property for $\text{PD}^g_{\text{OBNO}}$.

For $\text{PD}^g_{\text{TRAJ}}$, we can also prove the contact space realization property by contradiction. Suppose l_m be the C-space curve which realizes $\text{PD}^g_{\text{TRAJ}}$, i.e $l_m(0) = \mathbf{q_o}$ and $l_m(1) = \mathbf{q_m}$. If $\mathbf{q_m}$ does not lie in the contact space, we can obtain the $\mathbf{q_m}'$ which is the intersection between l_m and the contact space. The trajectory length for each point A is shorter when moving from $\mathbf{q_o}$ to $\mathbf{q_m}'$ than to $\mathbf{q_m}$. Therefore, $\text{TRAJ}(\mathbf{q_o}, \mathbf{q_m}') < \text{TRAJ}(\mathbf{q_o}, \mathbf{q_m})$. This contradicts our assumption that $\mathbf{q_m}$ is the realization for $\text{PD}^g_{\text{DISP}}$. □

According to Thm. 1, in order to compute PD^g, it is sufficient to search only the contact space $\mathcal{C}_{contact}$, which is one dimension lower than that of $\mathcal{C}$. In Chapter 4, we present a generalized PD algorithm using this property for constrained optimization.

2.5.2 Complexity of Generalized PD Computation

The overall complexity of generalized PD computation is governed by the computation of contact space which is defined as the set of every configuration for A at which it barely touches B. The computation of contact space usually boils down to two steps: enumerating contact surfaces and computing their arrangement (Latombe, 1991; Halperin, 2004). A contact surface is defined to be the locus of configurations of A at which a specific feature of A (vertex, face or edge) is in contact with a feature of B. For a rigid robot which can only translate, its contact surfaces are planar; otherwise, the contact surfaces are non-planar. The computation of contact space formed by these contact surfaces entails an arrangement problem. Given a finite set of hypersurfaces $\mathbf{S}$ (e.g. contact surfaces in the contact space computation) in a d-dimensional space, their arrangement $\mathbf{A}(\mathbf{S})$ is the decomposition of the d-dimensional space into cells $\mathbf{C}$ of dimensions $0, 1, \ldots, d$. Here, a k-dimensional cell $\mathbf{C}^k$ in $\mathbf{A}(\mathbf{S})$ is a maximally connected set contained in the intersection of a subset of the hypersurfaces in $\mathbf{S}$ that is not intersected by any other hypersurfaces in $\mathbf{S}$ (Halperin, 2004). It is well known that in the worst case, the combinatorial complexity of an arrangement of n hypersurfaces in a d-dimensional space is $O(n^d)$.

When a rigid polyhedral model only translates in 3D, its contact space lies in $\mathbb{R}^3$. At this time, the computation of its contact space is equivalent to the well-known *Minkowski sum* computation. The size of its contact space or *Minkowski sum* has an $O(n^2)$ combinatorial complexity for convex polytopes where n is the number of features in the objects; for two non-convex polyhedra, it has $O(n^6)$ complexity due to the arrangement of $O(n^2)$ planar contact surfaces in $\mathbb{R}^3$. Furthermore, the complexity of translational PD computation, which entails the search for the minimal distance to the contact space, has the same complexity as the contact space: $O(n^2)$ for two convex polytopes and $O(n^6)$

for two non-convex polyhedra in the worst case.

In general, computing generalized PD between two non-convex polyhedra is more difficult than computing the translational PD due to the 6-dimensional SE(3) space as well as non-linear rotational term embedded in its definition. Each contact surface for a 3D rigid model which translates and rotates is a 5-dimensional hypersurface. Therefore, its contact space, which can be computed by arrangement, has the $O(n^{12})$ combinatorial complexity for non-convex polyhedra in the worst case. Furthermore, the complexity of computing the minimal distance between a point and a contact surface, which is a non-planar hypersurface, is dependent on the chosen distance metric in SE(3). In addition, due to the non-linearity of the hypersurface, such computation is more expensive than computing the minimal distance between a point and a hyperplane whose running time is constant. Therefore, we conjecture that the overall complexity of generalized PD computation can be higher than $O(n^{12})$.

In addition to the high complexity of contact space computation for formulating generalized PD, the overall computation can also suffer from robustness issues. The arrangement of contact surfaces involves the intersection computation which can be difficult due to robustness issues (Raab, 1999). Even in $\mathbb{R}^3$, such intersection computation suffers from the numerical errors and can have the difficulty to produce numerically and topologically accurate results (Varadhan et al., 2006). For the contact space computation in a 6 dimensional $SE(3)$ space where each contact surface is non-linear, it is more difficulty to compute their arrangement precisely and robustly.

Chapter 3

Efficient C-Space Distance Computation

The notion of configuration space (C-space) is widely used in motion planning and other fields such CAD/CAM (Joskowicz and Sacks, 1999), dynamic simulation and virtual environments (Ruspini and Khatib, 2000). A fundamental problem is computation of a measure of the distance in C-space between two arbitrary configurations $\mathbf{q_a}$ and $\mathbf{q_b}$ for a model. Intuitively, the distance computed using some metric is used to quantify the extent of transformation of the model between two configurations $\mathbf{q_a}$ and $\mathbf{q_b}$. Many distance metrics have been proposed. In particular, the DISP distance metric (Fig. 2.5), which is defined as the maximum length of the displacement vector over the points of the model between two configurations and is described in Chapter 2, has several useful mathematical properties and is applicable to both rigid and articulated models. This distance metric can meaningfully combine translational and rotational motions without relying on any weighting factor. Additionally, it is invariant with the choices of reference frames and independent of the representation of the underlying configuration space. Given that no practical algorithms are known for computing DISP efficiently, its applications have been limited.

In this chapter, we present an efficient algorithm (C-DIST) to compute the DISP distance between two configurations of a rigid or articulated model (Zhang et al., 2008b). Our algorithm is based on Chasles theorem in Screw theory, and we show that for a rigid

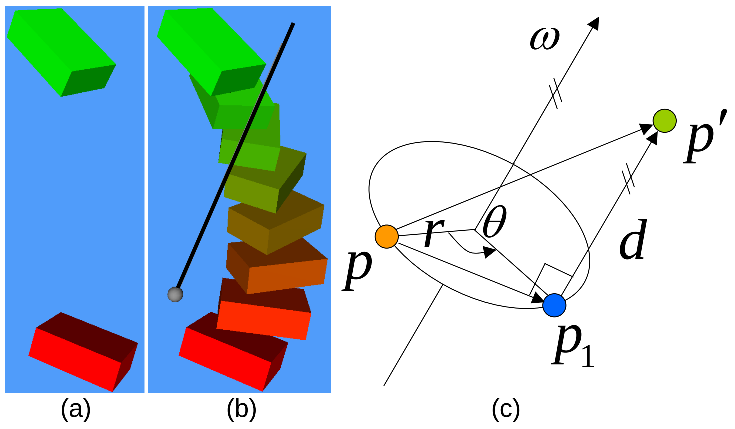


Figure 3.1: Chasles theorem in screw theory. *(a) shows a rigid body transformation of a rectangular bar. (b) The Chasles theorem states that any rigid transformation can be realized by rotation about an axis followed by translation parallel to that axis. Such axis is called a* screw axis. *(c) According to the Chasles theorem, when a model is transformed, the length of the displacement vector for any point* $\mathbf{p}$ *on the model can be calculated by first considering the rotation about the screw axis* ω *(from* $\mathbf{p}$ *to* $\mathbf{p}_1$*), then the translation along* ω *(from* $\mathbf{p}_1$ *to* $\mathbf{p}'$*).*

model the maximum distance is realized by one of the vertices on the convex hull. We use this formulation to compute the distance, and present two acceleration techniques: incremental walking on the dual space of the convex hull and culling vertices on the convex hull using a bounding volume hierarchy (BVH). Our algorithm can be easily extended to articulated models by maximizing the distance over its each link and we also present culling techniques to accelerate the computation. We have implemented our algorithm. We highlight its performance on many complex rigid models with hundreds of thousands of triangles and articulated models. In practice, the distance computation takes tens of micro-seconds on a high-end PC.

3.1 C-DIST Computation for Rigid Models

In this section, we present a novel formulation of this metric and an efficient algorithm, C-DIST, to compute the distance for rigid models. We first show that the DISP distance of a rigid polyhedral model is equal to the maximum length of the displacement vectors over the vertices on its convex hull (CH). Following this formulation, a straightforward algorithm is to maximize the displacement vectors over all the vertices on the CH, which has a linear complexity in the size of the CH. Finally, we present two different techniques to accelerate the distance computation: incremental walking on the dual space of the CH, and culling vertices on the CH using a bounding volume hierarchy (BVH) structure. In practice, the culling technique can improve the performance by an order of magnitude.

3.1.1 Convexity in C-DIST Computation

We first present a convex realization theorem for the DISP distance for a rigid model:

Theorem 2. (Convex Realization) *Given a rigid polyhedral model A, the* DISP *distance of A between two arbitrary configurations is equal to the maximum length of the displacement vectors over all the vertices on the convex hull of A.*

Proof. We use the Chasles theorem from screw theory as shown in Fig. 3.1 (a) and (b) (Ball, 1876; Murray et al., 1994). It states that a rigid body transformation from a configuration $\mathbf{q_a}$ to a configuration $\mathbf{q_b}$ can be realized by rotation about an axis followed by translation parallel to that axis. Such an axis is called a *screw axis.* As Fig. 3.1 (c) shows, when a model first rotates around the screw axis ω by θ, and translates along ω by d, a point $\mathbf{p}$ on the model will be displaced to $\mathbf{p}_1$, then to $\mathbf{p}'$.

We compute the length of the displacement vector $\overrightarrow{\mathbf{pp}'}$. Let us represent the distance from the point $\mathbf{p}$ to the axis ω as r. Given that the vector $\overrightarrow{\mathbf{pp}_1}$ is orthogonal to $\overrightarrow{\mathbf{p}_1\mathbf{p}'}$, the squared length of the displacement vector $\overrightarrow{\mathbf{pp}'}$ is given as:

$$\begin{aligned} ||\overrightarrow{\mathbf{pp}'}||^2 &= ||\overrightarrow{\mathbf{pp}_1}||^2 + ||\overrightarrow{\mathbf{p}_1\mathbf{p}'}||^2 \\ &= 4r^2 sin^2(\theta/2) + d^2 \\ &= 2(1 - cos\theta)r^2 + d^2. \end{aligned} \tag{3.1}$$

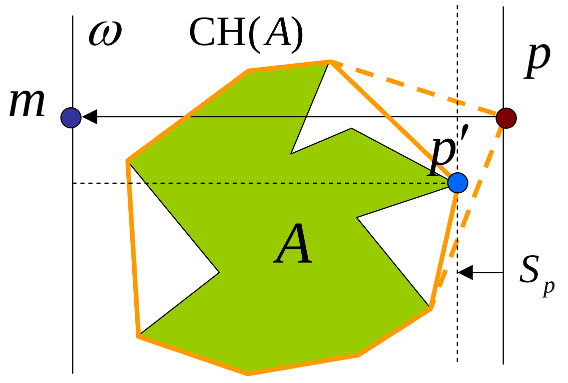


Figure 3.2: Maximum distance between a polyhedra and a line. *$\eta(A, \omega)$, the maximum distance from points on a polyhedral model A to a line ω is equal to the maximum distance from the vertices on its convex hull to ω.*

In Eq. (3.1), θ and d are independent of the model A and are solely governed by the input configurations $\mathbf{q_a}$ and $\mathbf{q_b}$. However, the distance r for every point on A to the screw axis ω varies. Since $(1 - cos(\theta)) \geq 0$ for any θ, a larger value of r implies a larger value of the length of the displacement vector. If we denote the maximum distance from every point on A to the screw axis ω as $\eta(A, \omega)$, DISP can be written as:

$$\text{DISP}(\mathbf{q_0}, \mathbf{q_1}) = \sqrt{2(1 - cos\theta)\eta^2(A, \omega) + d^2}. \tag{3.2}$$

According to Eq. (3.2), proving Thm. 2 is equivalent to proving the following lemma.

□

Lemma 1. *The maximum distance from points on a polyhedral model A to a line ω is equal to the maximum distance from the vertices on the convex hull of A to ω.*

We prove Lem.1 by contradiction. Throughout the rest of the proof, we distinguish a vertex which comprises a corner of the polyhedral model from a point which can be designated anywhere on the model.

It can be easily shown that Lem.1 holds when A is convex. If A is non-convex, we first find a **vertex p**, which is on the convex hull of A or CH(A), and is farthest to the line ω. Denote the projection of the vertex $\mathbf{p}$ on ω is $\mathbf{m}$. We construct a plane $S_\mathbf{p}$ which passes through $\mathbf{p}$ and is orthogonal to $\overrightarrow{\mathbf{pm}}$. Since $\mathbf{p}$ is the farthest point on $\mathrm{CH}(A)$ to ω and $A \subseteq \mathrm{CH}(A)$, A and ω must be located on the same half-space of S_p (see Fig. 3.2)

Assume that Lem.1 does not hold for non-convex A. This means that $\eta(A, \omega) \neq \eta(\mathrm{CH}(A), \omega)$. Since $A \subseteq \mathrm{CH}(A)$, we have $\eta(A, \omega) < \eta(\mathrm{CH}(A), \omega)$. As a result, A does not intersect with $S_{\mathbf{p}}$ (Fig. 3.2). Next, we translate the plane S_p along the direction of $\overrightarrow{\mathbf{pm}}$ until it touches a point $\mathbf{p}'$ on A. Denote S_p' as the resulting plane which is passing through $\mathbf{p}'$ and parallel to S_p. Because A lies entirely on one side of the S_p', we get a different convex hull for A. This contradicts the fact that the convex hull of an object is unique. As a result, Lem. 1 holds. □

Similarly to the proof for Thm. 2, we can also prove the following proposition for general smooth models.

Proposition 1. $\mathrm{DISP}(\mathbf{q_0}, \mathbf{q_1})$ *distance of a rigid smooth model* A *between two configurations* $\mathbf{q_0}$ *and* $\mathbf{q_1}$ *is equal to the maximum length of the displacement vectors over all the boundary points of* $\mathrm{CH}(A)$*;* $\mathrm{DISP}(\mathbf{q_0}, \mathbf{q_1})$ *can be calculated using Eq. (3.2), where* r *is the maximum distance from the boundary points on* $\mathrm{CH}(A)$ *to the screw axis.*

3.1.2 C-DIST Computation Algorithm

Two simple algorithms to compute $\mathrm{DISP}(\mathbf{q_0}, \mathbf{q_1})$ for a polyhedral model A follow directly from Thm. 2 and Eq. (3.2).

Maximization of Displacement Vectors. According to Thm. 2, one can compute the convex hull $\mathrm{CH}(A)$ of A and find the maximum length of displacement vectors for all the vertices on $\mathrm{CH}(A)$. In many applications, we can compute the convex hull of a rigid model as a preprocessing step. At runtime, DISP can be efficiently computed by only considering the vertices on the convex hull. If the size of the convex hull is small, it is plausible to consider all the vertices on the convex hull, compare the length of all displacement vectors and compute their maximum.

Maximization of Distance to Screw Axis. One can also use Eq. (3.2) to compute the DISP. In this equation, θ and d are determined by the motion between two configurations $\mathbf{q_a}$ to $\mathbf{q_b}$. $\eta(A, \omega)$, which depends on the underlying model A and screw axis can be computed by visiting all the vertices of CH(A) and computing the maximum distance to the screw axis ω.

Each of the two methods described above has a linear complexity in the size of the convex hull, and is efficient for moderately complex models. Furthermore, these two methods are robust and simple to implement. Finally, these methods do not impose any topological requirements on the model. Even for a model represented as a triangle soup, i.e., without connectivity information or as a point cloud, DISP can be computed easily.

In practice, however, the number of vertices on the convex hull can be rather high. Therefore, we present techniques to accelerate our algorithm by reducing the number of accesses to the vertices on the convex hull. In the following Sections 3.1.3 and 3.1.4, we present two acceleration techniques: incremental walking on the dual space of CH, and culling vertices on CH by using a bounding volume hierarchy (BVH) structure.

3.1.3 Accelerating C-DIST Computation by Incremental Walking

In order to reduce the number of accesses to the vertices on the convex hull, we present an optimization-based algorithm that performs feature walking on the dual space of the convex hull. We use the properties of Gauss map to compute the extremal vertices of convex hull, which are orthogonal to the screw axis.

Given Eq. (3.2), it follows that DISP is realized by one of the vertices on the convex hull CH(A), which has the maximum distance to the screw axis ω. We use this property to compute DISP in two steps:

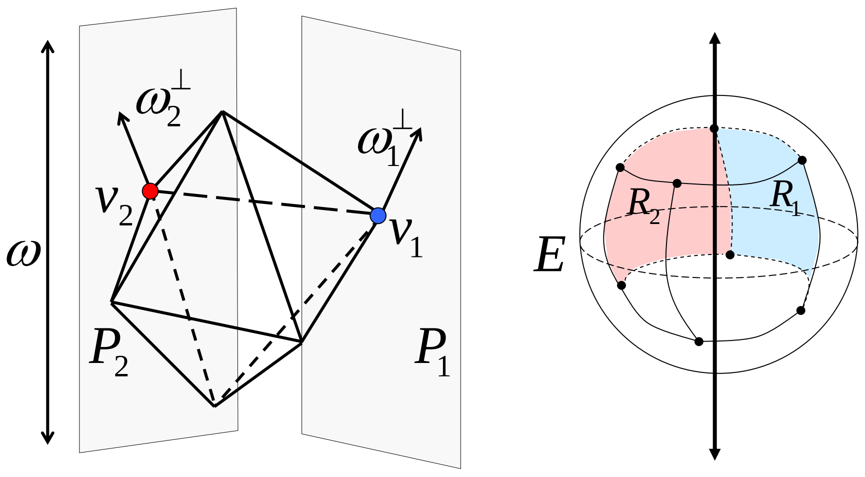


Figure 3.3: Walking Algorithm. *The maximum distance $\eta(A,\omega)$ from every point on A to the screw axis ω can be computed by visiting all the vertices on the convex hull whose support plane can have a normal $\omega^\perp$ orthogonal to ω. (Left) an initial vertex $v_k = v_1$ for the walking algorithm is located since v_1 can have a supporting plane P_1 whose normal is $\omega_1^\perp$ orthogonal to ω. The next search vertex $v_{k+1} := v_2$ is found since v_2 is adjacent to v_1 and has a supporting plane P_2 whose normal is $\omega_2^\perp$ orthogonal to ω. This search process is iterated until v_{k+1} becomes equal to v_1 again. (Right) The same process can be explained based on the Gauss map of the given convex hull. Enumerating all the vertices whose supporting plane can have a normal orthogonal to ω is equivalent to finding all the regions (including R_1, R_2 that are mapped from v_1, v_2, respectively) on the Gauss map that intersect with the equator E when ω is mapped to the pole of the Gauss map.*

1. Enumerate all the vertices v_i on CH(A) supported by a plane whose normal is orthogonal to ω.
2. Find a vertex in v_i that corresponds to $\eta(A,\omega)$.

The main computational task in the above algorithm lies in the first step. A relatively straightforward way to implement this step is (Fig. 3.3):

1. Choose any direction $\omega^\perp$ orthogonal to ω. Find a vertex v_1 whose supporting plane has a normal parallel to $\omega^\perp$ and set v_1 as the current search vertex $v_k := v_1$. Computing v_1 is known as support mapping of $\omega^\perp$ or extremal vertex query along $\omega^\perp$, and it can be computed in logarithmic time in the number of vertices of the convex hull (de Berg et al., 1997). In practice, the support mapping can be

efficiently implemented using a lookup table.

2. Walk to the neighboring vertices v_{k+1} of v_k, if v_{k+1} has a supporting plane with a normal orthogonal to ω.

3. Repeat the above two steps, until v_{k+1} becomes equal to v_1.

Alternatively, we can compute all v_i's based on the Gauss map of CH(A). The mapping is defined from the feature space of an object to the surface of a unit sphere $\mathbb{S}^2$ as: a vertex is mapped to a region, a face to a point and an edge to a great arc (Spivak, 1999). The task of enumerating all v_i's boils down to finding the intersecting regions on the Gauss map with its equator when the north or south pole of the Gauss map corresponds to the direction of ω. The computational complexity of this algorithm is governed by two factors: finding an initial vertex v_1 and the walking step itself. Finding v_1 can have a logarithmic complexity in terms of the number of vertices of the convex hull and the walking step has a linear complexity in the number of vertices that are traversed.

In practice, computing the intersections between Gauss map regions and the equator can be performed by centrally projecting both the equator and the Gauss map to a plane and finding the intersections of convex polygons (projected Gauss regions) and a line (projected equator).

3.1.4 Accelerating C-DIST Computation using a Bounding Volume Hierarchy (BVH)

In practice, the vertices of the convex hull of the models are not distributed uniformly in 3D space. As a result, the walking scheme highlighted above can result in robustness problems, especially when accessing those vertices that are very close with each other. In this section, we present a different acceleration technique for C-DIST computation. Our method uses a bounding volume hierarchy (BVH) tree structure to cluster the vertices

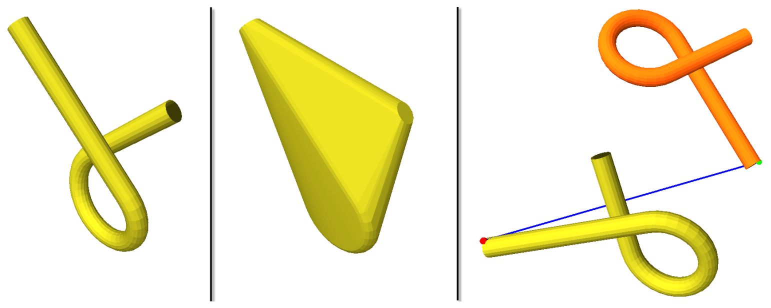


Figure 3.4: C-DIST computation. *The leftmost figure shows the 'alpha' model with* 1, 008 *triangles, and the middles its convex hull with* 311 *vertices. The rightmost figure shows a scenario when the model is placed at two arbitrary configurations. The vertex on this model, which realizes the longest displacement vector is found by our* C-DIST *algorithm and highlighted. The line segment connecting this vertex at the two configurations is also highlighted. Using SSV-Tree, our* C-DIST *algorithm can perform distance query for this model within* $5.6\mu s$.

on a convex hull. The BVH enables us to efficiently compute $\eta(A, \omega)$, the maximum distance between from every point on a model A to an axis ω. The acceleration is achieved because a node in the BVH as well as its descendant nodes can be culled if the distance η from this node to the axis is less than the global maximum distance.

In practice, we use the BVH of swept sphere volumes (SSV) (Larsen et al., 1999). SSV includes three different types of bounding volumes (BVs): point swept sphere (PSS), line swept sphere (LSS), and rectangle swept sphere (RSS). PSS, LSS, and RSS are created by sweeping a sphere along a point, a line and a rectangle in three-dimensional space, respectively.

SSV-Tree Construction for a Point Set. Let us denote S as a set of vertices on the convex hull of the given model A. As a preprocessing step, our algorithm recursively builds an SSV-tree for the point set S from top to bottom. We first compute its swept sphere volume, $SSV(S)$, as the root node of the SSV-Tree. To decide whether to further subdivide a node N into two children nodes, we measure the density ρ defined as the number of vertices inside a node over its volume. If ρ is larger than some given threshold, we terminate the subdivision. Otherwise, we partition the point set Q inside

N into two subsets Q_1 and Q_2 in a way to maximize the sum of densities for $SSV(Q_1)$ and $SSV(Q_2)$. For the purpose of maximization, we sweep a partitioning plane along the longest dimension of the node N and evaluate $\rho(Q_1) + \rho(Q_2)$ of two resulting point subsets, Q_1, Q_2. We choose a partitioning plane that maximizes this sum.

SSV-Tree Traversal and SSV-Axis Distance Query. We use the SSV-Tree structure to efficiently query the maximum distance from a point set S to the axis ω. By initializing the global maximum distance as $-\infty$ and starting from the root node, our algorithm traverses its associated SSV-Tree in the depth-first order. During the traversal, we compute the maximum distance from the visited SSV node to the axis ω. Depending on the type of the underlying SSV, we need to compute the maximum distance between $1, 2$ or 4 corner spheres of the SSV and the axis. If this distance is not greater than the global maximum distance, we need not check the node as well as its descendant nodes any more, and can cull them. Otherwise, the depth-first order traversal continues. During the traversal, if a leaf SSV node is reached, we check whether the distance from any point contained in the SSV node to the axis is larger than the global maximum distance; if yes, we update the global maximum distance.

3.2 C-DIST Computation for Articulated Models

The C-DIST computation algorithm for rigid models can be extended to articulated models, whose links can form serial or parallel chains, tree structure, or closed loops. In order to compute the DISP distance for an articulated model between two arbitrary configurations, we consider each of its links as a separate rigid body; the maximum DISP distance over all articulated links is the DISP distance for this articulated model.

This algorithm can be improved by conservatively estimating the DISP distance for each link using a bounding volume and comparing it with the DISP for other links that have been already calculated. Specifically, for a link L in an articulated model A, we

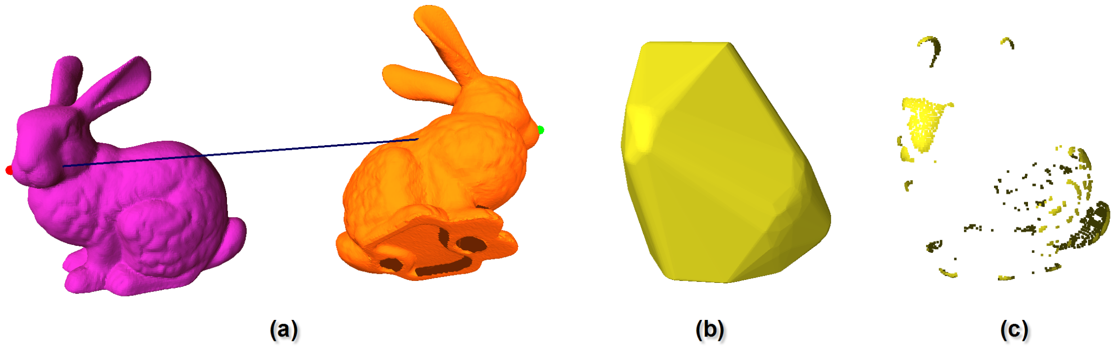


Figure 3.5: C-DIST computation. *The rigid model - 'bunny' has* 69, 451 *triangles. (a) shows a scenario of the* DISP *distance query between two different configurations of the bunny model. Our* C-DIST *algorithm can perform the query within* $8.0\mu s$ *on average. (b) shows the convex hull of this model. (c) shows the vertices on this convex hull, which are not uniformly distributed in 3D space. This can result in the robustness problems for the* $CDIST$ *algorithm based on incremental walking.*

precompute its bounding volume (e.g., an *oriented bounding box* of link L, $OBB(L)$). If DISP for all the links that have been already computed is greater than DISP for $OBB(L)$, we need not compute the exact DISP for link L and can cull it away.

In case of an articulated robot forming a serial chain, a link farther from the base of the robot typically undergoes a larger displacement as compared to the ones that are nearer to the base. Therefore, a simple heuristic to accelerate DISP computation for such an articulated robot is to first compute DISP for links that are farther from the base.

3.3 Implementation and Performance

We have implemented our C-DIST computation algorithm and applied it to various rigid and articulated models. In this section, we highlight its performance on these complex benchmarks. All the timings reported in this section were taken on a 2.8GHz Pentium IV PC with 2GB of memory.

	Alpha Puzzle	Cup	Bunny	Hand	Dragon
#Tri	1,008	4,226	69,451	86,361	871,414
#V	3,024	3,000	208,353	259,803	2,614,242
#V of CH	311	1,019	1,504	1,836	2,448
t_{pre} (s)	0.016	0.002	0.584	0.661	9.517
t_{pre_qhull} (s)	0.016	0.000	0.581	0.651	9.508
t_{pre_ssv}(s)	0.000	0.002	0.003	0.010	0.009
t_{bf} (μs)	184.8	231.1	13,633	16,730	169,062
t_{ch} (μs)	19.1	64.1	85.3	105.0	153.0
t_{op_ssv} (μs)	5.6	10.2	8.0	18.2	15.2
Speedup (t_{ch}/t_{op_ssv})	3.4	6.3	10.7	5.76	10.1

Table 3.1: Performance of C-DIST for rigid models. *$\#V$, $\#V$ of* CH *denote the number of the vertices on each input model and the number of vertices on its convex hull, respectively. t_{pre} is the time for the preprocessing step, including the computation of convex hull and building the BVH structure. t_{bf}, t_{ch}, t_{bf} are the average* DISP *query time, based on three different methods. t_{bf} is the running time of the brute-force method that checks all the vertices of the input model. t_{ch} is the running time of our* C-DIST *method, which checks all the vertices on the convex hull. t_{op} is the running time of our accelerated* C-DIST *method that uses a SSV-Tree.*

3.3.1 Rigid Models

In our implementation, we precompute the convex hull for an input rigid model using *QHull* [1]. We further build a BVH structure - *swept sphere volumes* (SSV) tree for the vertices on the convex hull. For BVH-based DISP computation, we use the SSV-Tree to compute the maximum distance from the model to the screw axis.

We test our C-DIST implementation on a set of rigid polyhedral models, including triangular mesh models, such as the *Cup* model, and triangle soup models without connectivity information, such as *Alpha Puzzle*, *Bunny*, *Hand* and *Dragon*. The model complexity and the performance is summarized in Table 3.1.

Fig. 3.4 shows the *Alpha Puzzle* model as well as its convex hull. In this test, we place this model at two arbitrary configurations and use our C-DIST algorithm to compute the DISP distance. We highlight the line segment that connects the vertex with the

[1]http://www.qhull.org/

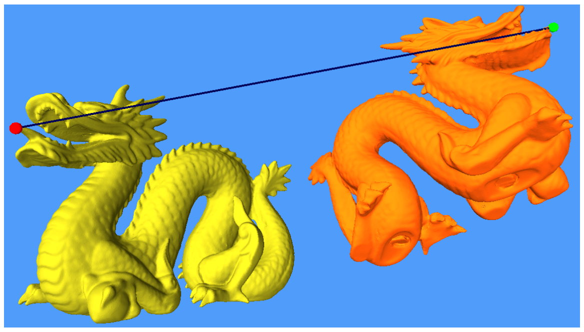


Figure 3.6: C-DIST computation. *The rigid model - 'dragon' has* 871, 414 *triangles. Our* C-DIST *algorithm can perform the* DISP *distance query at two different configurations. Our algorithm takes about* $15.2\mu s$, *on average, to perform this query.*

largest displacement at the two configurations. For this model, the brute force method of visiting all the vertices on the model takes $184.8\mu s$ on average, for each DISP query. Our C-DIST algorithm that compares all the vertices on the convex hull can perform the query in $19.1\mu s$. Using SSV-Tree, our C-DIST computation can perform the query in $5.6\mu s$.

Figs. 3.5 and 3.6 highlight the application of C-DIST algorithm to complex models consisting of tens of thousands of triangles. According to Table 3.1, our C-DIST computation based on SSV-Tree can perform each distance query within $20\mu s$ for these models. We achieve up to 10 times speedup over an algorithm that computes the displacement for each vertex of the convex hull.

3.3.2 Articulated Models

Our C-DIST implementation for articulated models is built on top of C-DIST computation for rigid models. We construct an OBB for each link of the model, and use them for culling.

Fig. 3.8 highlights a 9-DOF articulated model - *Puma* manipulator with 6 joints as well as 3 degree of freedoms of its base. Our C-DIST algorithm can perform the distance query in $27.91\mu s$ on average. Fig. 3.9 shows a complex articulated model with 6 DOF -

	DOF	Tri	#V of CH	t_{ch} (μs)	t_{obb} (μs)	Speedup
Puma	9	868	296	42.30	27.91	1.5
IRB-2400	6	3,791	531	73.89	21.90	3.4

Table 3.2: Performance of C-DIST for articulated models. *Using OBB culling, our* C-DIST *can perform the* DISP *query for these two examples within* $30\mu s$.

IRB2400 with an arcgun. Our algorithm can perform DISP distance query in $21.90\mu s$. Tab. 3.2 summarizes the performance of C-DIST for these two articulated models.

3.3.3 Comparison

As compared to other distance metrics that are used for configuration space distance computation, such as any weighted metric combing the translational and rotational components, DISP has many elegant mathematical properties. For example, the users need not choose any weighting factor between the translational and rotational components. DISP metric shares many properties with another model-dependent metric - object norm, which is easier to compute than DISP metric. However, each of these metrics is suitable for different applications. Since object norm is defined as an average squared length of all displacement vectors of a model, it could characterize the variation of energy when a model is transformed from one configuration to another configuration. In contrast, due to the geometric property of DISP metric implied by the maximum operation in its definition, DISP metric can be more useful for proximity queries and path planning.

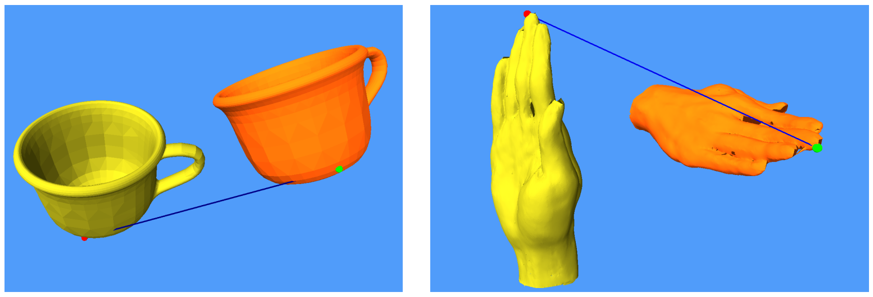


Figure 3.7: C-DIST computation. *Our algorithm can handle any polyhedral model represented as a triangular mesh, a triangle soup, or a point-set model. The 'Cup' model on the left is represented as a triangular mesh, while the 'Hand' model on the right is a scanned model and represented as a triangle soup with* $86,361$ *triangles. Our algorithm can perform the distance query for these models within* $10.2\mu s$ *and* $18.2\mu s$*, respectively.*

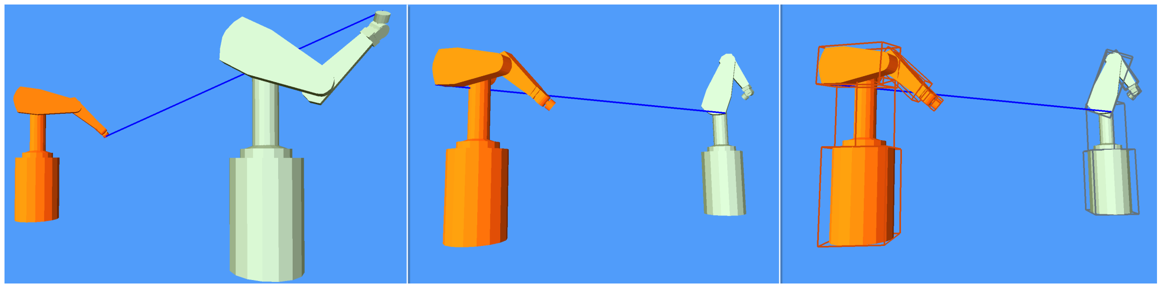


Figure 3.8: C-DIST computation for an articulated model - 'puma'. Left: the model with 9 DOF include 6 articulated joints and 3 DOF of its base. Middle: the result of the distance query by our C-DIST algorithm is highlighted. Right: the OBB associated with each link. For this model, our algorithm can perform the query within $27.91\mu s$, on average.

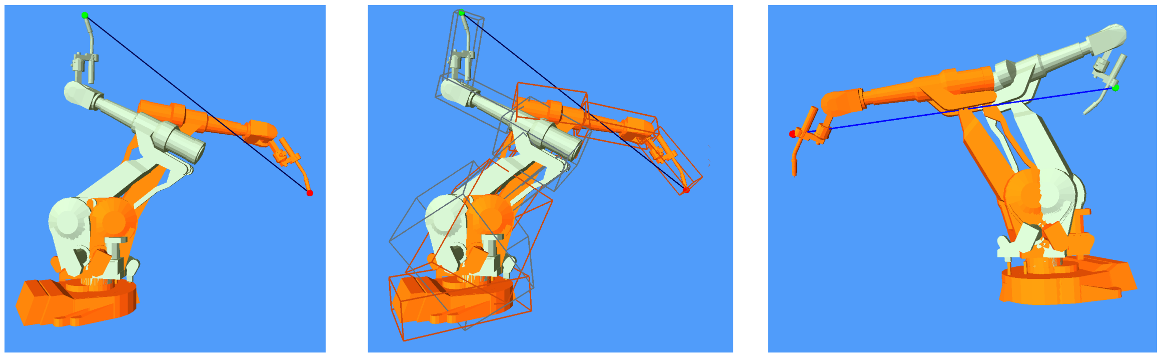


Figure 3.9: C-DIST computation for an articulated model - 'IRB2400' with an arcgun. *This model has* 6 *joints and* $3,791$ *triangles. Our algorithm can perform the query within* $27.91\mu s$ *on average. In the middle, we highlight the OBB associated with each link which can be used for efficient culling.*

Chapter 4

Generalized Penetration Depth Computation

In Chapters 2 and 3, we have presented the notion of generalized PD for quantifying the extent of intersection and addressed its formulation by choosing appropriate C-space distance metrics. In general, generalized PD computation has very high combinatorial complexity for 3D non-convex models and no prior efficient algorithms are known. In this chapter, we present two generalized PD algorithms for 3D models. The convexity-based algorithm explores the convexity of the models (Zhang et al., 2007c). The algorithm uses convex decomposition and containment optimization to compute lower and upper bounds on generalized PD. The second algorithm is based on constrained optimization (Zhang et al., 2007a). We present efficient techniques to compute a local approximation of contact space of models for iterative refinement. The second algorithm dose not perform convex decomposition and is applicable to polygon soup models. We highlight the efficiency and robustness of both algorithms on many complex 3D models. In Chapter 5, we use generalized PD computation to perform C-obstacle query for complete motion planning. In Chapter 6, we use generalized PD computation to perform retraction-based sampling for cluttered environments.

4.1 A Convexity-based Generalized PD Algorithm

In this section we present an algorithm for computing generalized PD for polyhedral models using TRAJ metric (Eq. 2.6) (Zhang et al., 2007c). Our algorithm is based on convex decomposition of the given models. The algorithm can also be extended for DISP and OBNO metrics. We present three new results to compute $\mathrm{PD}^g_{\mathrm{TRAJ}}$. First, we show that for two overlapping convex polytopes, $\mathrm{PD}^g_{\mathrm{TRAJ}}$ is equivalent to PD^t. Second, when the complement of one of the objects is convex, we pose the generalized PD computation as a variant of the convex containment problem and compute an upper bound using optimization techniques. Finally, when both objects are non-convex, we treat them as a combination of the above two cases and present an algorithm that computes a lower bound and an upper bound on the generalized PD. We have implemented our algorithm and applied it to many non-convex 3D models undergoing rigid motion. The running time varies with the complexity and the relative configurations of the two models. In practice, our algorithm takes about 2 ms to 6 ms on a 2.8 GHz PC to compute the lower bound on generalized PD, and 20 ms to 1 sec for the upper bound.

4.1.1 Convex Objects

We first consider the problem of computing generalized PD between two convex objects. In this case, it was shown by Ong (1997) that generalized PD using DISP metric is equivalent to PD^t. Similarly, we can prove that $\mathrm{PD}^g_{\mathrm{TRAJ}}$ is also equivalent to PD^t. As a result, the well-known algorithms to compute PD^t between convex polytopes (van den Bergen, 2001; Kim et al., 2002a) are directly applicable to generalized PD.

Theorem 3. (Convexity of Generalized PD Computation) *Given two convex objects A and B, we have:*

$$\mathrm{PD}^g_{\mathrm{TRAJ}}(A, B) = \mathrm{PD}^t(A, B).$$

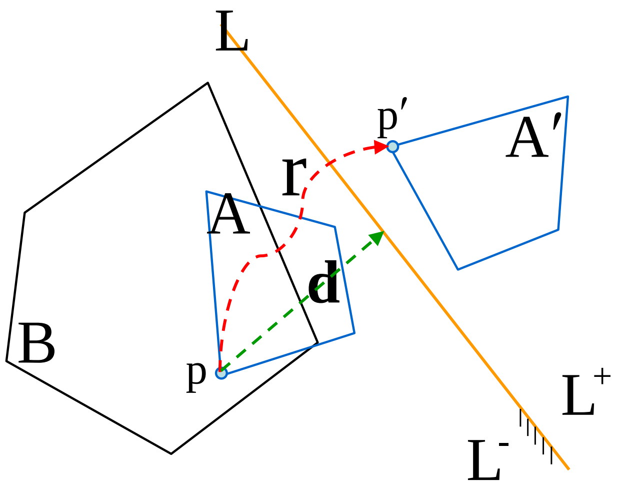


Figure 4.1: *Proof for* $\mathrm{PD}^t(A,B) = \mathrm{PD}^g_{\mathrm{TRAJ}}(A,B)$ *for convex objects* A *and* B*. Let* A' *a placement of* A *which realizes* $\mathrm{PD}^g_{\mathrm{TRAJ}}$*.* L *is an arbitrary separating plane between* A' *and* B*, which divides the space into two half-spaces* L^- *and* L^+*. For any* L*, there always exists a point* $\mathbf{p}$ *on* A *on* L^- *side with* $||d|| \geq \mathrm{PD}^t(A,B)$*. As a result, we cannot move* A *towards* $L+$ *side with a traveling distance that is less than* $\mathrm{PD}^t(A,B)$*, even when rotational DOF are allowed. Therefore, generalized PD based on* TRAJ *metric is equal to translational PD for convex objects.*

Proof. Let us assume that A and B intersect; Otherwise, it is trivial to show that $\mathrm{PD}^g_{\mathrm{TRAJ}} = \mathrm{PD}^t = 0$.

First of all, we can say that $\mathrm{PD}^g_{\mathrm{TRAJ}} \leq \mathrm{PD}^t$, as $\mathrm{PD}^g_{\mathrm{TRAJ}}$ is realized under more DOF than PD^t . Next, we show that $\mathrm{PD}^g_{\mathrm{TRAJ}} < \mathrm{PD}^t$ is not possible and therefore, we can conclude $\mathrm{PD}^g_{\mathrm{TRAJ}} = \mathrm{PD}^t$. We use a proof by contradiction.

Suppose $\mathrm{PD}^g_{\mathrm{TRAJ}} < \mathrm{PD}^t$. Let us call A' as the placement of A that realizes $\mathrm{PD}^g_{\mathrm{TRAJ}}$, implying that A' is disjoint from B (Fig. 4.1). Since A' and B are convex, there exists a separating plane L that separates A' and B. Moreover, let L divide the entire space into two half-spaces: L^-, which contains B and L^+, which contains A'. Let $\mathbf{p}$ be the farthest point on A on L^- side from the separating plane L, and $\mathbf{d}$ be the vector from $\mathbf{p}$ to its nearest point on L. As a result, $||\mathbf{d}|| \geq \mathrm{PD}^t$. Otherwise, we could separate A and B by translating A by $\mathbf{d}$, which would result in a smaller PD^t (i.e. $\mathbf{d}$), and this contradicts the definition of PD^t in Eq. (2.1).

Since $\mathbf{p}$, which is on L^- side, is at least PD^t far away from L, $\mathbf{p}$ must travel at least by PD^t to reach the new position $\mathbf{p}'$, which can be lying on L or contained in L^+. However, according to the definition of $\mathrm{PD}^g_{\mathrm{TRAJ}}$ and the assumption of $\mathrm{PD}^g_{\mathrm{TRAJ}} < \mathrm{PD}^t$, there must exist a trajectory l connecting $\mathbf{p}$ and $\mathbf{p}'$, whose arc-length is less than PD^t. This means that $\mathbf{p}$ could be moved to L or within L^+ by less than the amount of PD^t, which is contradictory to the earlier observation that $\mathbf{p}$ must travel at least by PD^t. Therefore, we conclude that L can not be a separating plane between A' and B.

The previous deduction shows that under the assumption that $\mathrm{PD}^g_{\mathrm{TRAJ}} < \mathrm{PD}^t$, no separating plane can exist. This contradicts the fact that a separating plane must exist when convex objects are disjoint. Therefore, $\mathrm{PD}^g_{\mathrm{TRAJ}} < \mathrm{PD}^t$ is not possible and hence $\mathrm{PD}^g_{\mathrm{TRAJ}} = \mathrm{PD}^t$. □

Corollary 1. *For two convex objects A and B, their generalized PD is commutative; i.e.,*

$$\mathrm{PD}^g_{\mathrm{TRAJ}}(A, B) = \mathrm{PD}^g_{\mathrm{TRAJ}}(B, A).$$

Proof. For convex objects, $\mathrm{PD}^g_{\mathrm{TRAJ}}(A, B) = \mathrm{PD}^t(A, B)$ and $\mathrm{PD}^g_{\mathrm{TRAJ}}(B, A) = \mathrm{PD}^t(B, A)$. Since PD^t is commutative such that $\mathrm{PD}^t(A, B) = \mathrm{PD}^t(B, A)$, it follows that $\mathrm{PD}^g_{\mathrm{TRAJ}}(A, B) = \mathrm{PD}^g_{\mathrm{TRAJ}}(B, A)$. □

Non-Convex objects. Note that, for non-convex objects, $\mathrm{PD}^g_{\mathrm{TRAJ}}(A, B)$ is not necessarily equal to $\mathrm{PD}^t(A, B)$. Figs. 2.3 and 4.2 show such examples. In Fig. 2.3, $\mathrm{PD}^g_{\mathrm{TRAJ}}(A, B) < \mathrm{PD}^t(A, B)$, because when both translation and rotation are allowed in (b), the trajectory length that any point on A travels is shorter than its corresponding length when only translation is allowed in (a). In Fig. 4.2, an object B, which could be infinitely large with a hole inside, can contain A only when A adjusts its initial orientation. Hence, the $\mathrm{PD}^t(A, B) = \infty$ (i.e. the height of B), but $\mathrm{PD}^g_{\mathrm{TRAJ}}(A, B)$ is not ∞ (i.e. is much smaller than the height). So, $\mathrm{PD}^g_{\mathrm{TRAJ}}(A, B) < \mathrm{PD}^t(A, B)$. We can also see that $\mathrm{PD}^g_{\mathrm{TRAJ}}(A, B)$ is not necessarily equal to $\mathrm{PD}^g_{\mathrm{TRAJ}}(B, A)$ in this example. If B

is movable, the TRAJ metric for B at any two distinctive orientations is always ∞, because B is unbounded. Therefore, $\mathrm{PD}^g_{\mathrm{TRAJ}}(B, A)$ is ∞ in this case, while $\mathrm{PD}^g_{\mathrm{TRAJ}}(A, B)$ is not ∞.

4.1.2 A Convex Object and a Convex Complement

In this section, we show how to pose the generalized PD computation as a containment problem. Using this formulation, we investigate a special case of generalized PD where a movable object A and the complement of a fixed object B (i.e. $\bar{B}$) are both convex (as shown in Fig. 4.2). Instead of computing an exact solution for this case, we compute an upper bound of $\mathrm{PD}^g_{\mathrm{TRAJ}}$ by using a two-level optimization algorithm based on linear programming.

Relationship between Generalize PD and Object Containment Problems

The general *object containment problem* can be stated as follows: given two objects P and Q, determine whether Q can contain P by performing translation and rotation transformation on P. Generalized PD defined in Eq. (2.2) is closely related to the object containment problem. That is, testing $interior(A(\mathbf{q})) \cap B = \emptyset$ in Eq.(2.2) can be reduced to a containment query: whether $\bar{B}$ can contain A, as shown in Fig. 4.2. However, there are a few differences between these two problems. The object containment problem finds one instance of a placement of P that can fit inside of Q, whereas generalized PD computation needs to search through all containment configurations to find a configuration that minimizes the objective function defined by the distance metric δ.

The standard object containment problem is known to be difficult even for 2D polygonal models. However, if the primitives are convex, computational complexity of containment reduces from $O(n_1^3 n_2^3 log(n_1 n_2))$ to $O(n_1 n_2^2)$ for polygons with n_1 and n_2 vertices (Chazelle, 1983; Avnaim and Boissonnat, 1989a). As a result, we consider the case when

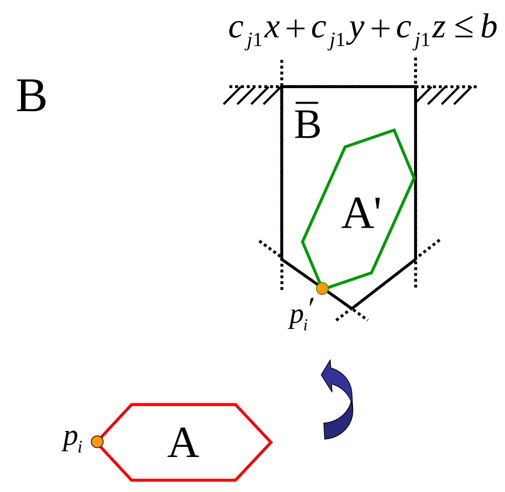


Figure 4.2: Generalized PD computation between the convex object A and the object B whose complement - $\bar{B}$ is convex. *In this case, the* $\text{PD}^g_{\text{TRAJ}}(A,B) \neq \text{PD}^t(A,B) = \infty$ *and* $\text{PD}^g_{\text{TRAJ}}(A,B) \neq \text{PD}^g_{\text{TRAJ}}(B,A) = \infty$*. We compute an upper bound on* $\text{PD}^g_{\text{TRAJ}}$ *by reducing the problem to a variant of the convex containment problem by using linear programming.*

a movable polyhedron A is convex and the complement of a fixed polyhedra B is convex as well. To compute an upper bound of $\text{PD}^g_{\text{TRAJ}}$ for this case, our algorithm performs two levels of optimizations:

1. We compute a configuration $\mathbf{q_1}$ for A so that the convex container $\bar{B}$ contains $A(\mathbf{q_1})$. This is performed by minimizing their overlap. The computed containment configuration yields an upper bound of $\text{PD}^g_{\text{TRAJ}}$, which may not be tight.

2. We iteratively compute a configuration, $\mathbf{q_2}$, to yield a tighter upper bound of $\text{PD}^g_{\text{TRAJ}}$ by setting the upper bound of TRAJ metric in Eq. (2.8) as the objective function for optimization. The algorithm terminates until a locally-optimal configuration is computed.

Computing a Containment

In this section, we introduce the formulation of the convex containment problem and extend the 2D optimization-based algorithm described by Grinde and Cavalier (1996) and Milenkovic (1999) to 3D objects, both of which serve as a foundation of finding a locally-optimal containment.

To check whether A lies fully inside $\bar{B}$ can be mathematically formulated as follows. The convex object, $\bar{B}$ with n faces is represented as an intersection of n half-spaces $\mathbf{c}_j\mathbf{x} \leq b_j, j = 1, ..., n$. A placement of A lies fully inside $\bar{B}$ if and only if every vertex $\mathbf{p}_i (i = 1, ..., m)$ of A lies inside all the half-spaces, i.e. $\mathbf{c}_j\mathbf{p}_i \leq b_j, i = 1, ..., m, j = 1, ..., n$, or:

$$\mathbf{C}\mathbf{p}_i \leq \mathbf{b}, i = 1, ..., m, \tag{4.1}$$

where $\mathbf{C} = [\mathbf{c}_1, ..., \mathbf{c}_m]$; $\mathbf{c}_j$ is normalized so that for a given point $\mathbf{p}$, $|\mathbf{c}_j \cdot \mathbf{p} - b_j|$ is the Euclidean distance from $\mathbf{p}$ to the corresponding face j.

Denote $\mathbf{R}$ as the *rotation matrix* when A is rotated around an arbitrary axis from its origin $\mathbf{o}$. When A is rotated by $\mathbf{R}$, followed by the translation of $\mathbf{t}$, the new position of $\mathbf{p}$ on A can be calculated as:

$$\mathbf{p}' = \mathbf{R}(\mathbf{p} - \mathbf{o}) + \mathbf{o} + \mathbf{t}. \tag{4.2}$$

The 3D containment problem now can be stated as finding a solution to the following system:

$$\mathbf{C}\mathbf{p}'_\mathbf{i} \leq \mathbf{b}, i = 1, ..., m. \tag{4.3}$$

The 3D containment computation stated by Eq. (4.3) is a non-linear problem, as the rotation matrix $\mathbf{R}$ is embedded with non-linear terms. These non-linear terms could be linearized by using a *small-angle approximation* (Milenkovic and Schmidl, 2001). When A is rotated by α around an arbitrary axis $\hat{\mathbf{a}}$, its rotation vector $\mathbf{r}$ is equal to $\alpha\hat{\mathbf{a}}$. If the variation of a rotation angle α is small enough, we can get a linearized approximation for Eq. (4.2):

$$\tilde{\mathbf{p}} \approx \mathbf{p} + \mathbf{r} \times (\mathbf{p} - \mathbf{o}) + \mathbf{t}. \tag{4.4}$$

By replacing $\mathbf{p}'$ by its approximation $\tilde{\mathbf{p}}$, the non-linear system in Eq. (4.3) is simplified to a linear one:

$$g_{ij} = \mathbf{c}_j \cdot \mathbf{t} - (\mathbf{c}_j \times (\mathbf{p}_i - \mathbf{o})) \cdot \mathbf{r} + (\mathbf{c}_j \cdot \mathbf{p}_i - b_j) \leq 0, \ \forall i, j. \tag{4.5}$$

Here $\mathbf{t}$ and $\mathbf{r}$ are the unknown vectors. g_{ij}, which is called as *containment function*, is defined for each pair of the vertex of A and the face of $\bar{B}$:
In order to solve the linear system defined in Eq. (4.5), we introduce a slack variable d_{ij} for each pair (i,j), representing the distance from $\tilde{\mathbf{p}}_i$, the approximate position of $\mathbf{p}_i$ of A after it is transformed, to the j'th face on $\bar{B}$. In this case, the 3D convex containment constraint for A and $\bar{B}$ can be approximated as a linear programming problem (LP1):

$$\begin{aligned} &\min \ \ Z = \sum_{i=1}^{m} \sum_{j=1}^{n} d_{ij}, \\ &\text{subject to} \ \ g_{ij}(\mathbf{t}, \mathbf{r}) - d_{ij} = 0 \ \forall i, j. \end{aligned} \tag{4.6}$$

If the solutions d_{ij} for this optimization problem are less than or equal to zero, we end up computing a solution to Eq. (4.5).

Given A and $\bar{B}$, we construct a linear programming problem defined as in Eq. (4.6) and apply the standard linear programming technique to optimize its objective function Z. We compute the solution, say $(\mathbf{t}, \mathbf{r})$, and place A at A'. Then a new linear programming problem for A' is constructed and solved. These steps are iterated until a local minimum for Z is computed. As the algorithm iterates, the small-angle approximation for the rotation matrix $\mathbf{R}$ becomes more accurate. When the solutions d_{ij} are less than or equal to zero, a valid containment of A at configuration $\mathbf{q}_1$ has been found.

Computing a Locally-Optimal Containment

The optimization algorithm highlighted earlier can only find a valid containing configuration for A, which yields an upper bound for $\mathrm{PD}^g_{\mathrm{TRAJ}}$. We perform the second level of optimization to compute an even tighter upper bound for $\mathrm{PD}^g_{\mathrm{TRAJ}}$ by using the result of the first level optimization as an initial condition.

Let $\mathbf{q}_0 = (\mathbf{t}_0, \mathbf{r}_0)$ be the initial configuration of A used in the first level optimization. Let $\mathbf{q}_1 = (\mathbf{t}_1, \mathbf{r}_1)$ be the resulting configuration of the first level optimization. Our goal is to compute $\Delta\mathbf{q} = (\Delta\mathbf{t}, \Delta\mathbf{r})$, such that $\mathbf{q}_2 = (\mathbf{t}_1 + \Delta\mathbf{t}, \mathbf{r}_1 + \Delta\mathbf{r})$ yields another containing placement of A while $\mathrm{TRAJ}(\mathbf{q}_0, \mathbf{q}_2) < \mathrm{TRAJ}(\mathbf{q}_0, \mathbf{q}_1)$.

We perform the second level optimization by setting the upper bound on the TRAJ metric in Eq. (2.8) as an optimization objective function. Here we do not choose Eq. (2.7), because the 3D containment computation uses the notation of rotation vector. Setting the containment of A by $\bar{B}$ as a hard constraint, we get the system:

$$\begin{aligned} &\min \;\; Z = \sum_{k=1}^{3} |\Delta t_k + t_{1,k} - t_{0,k}| + \mathbf{R} \sum_{k=1}^{3} |\Delta r_k + r_{1,k} - r_{0,k}|, \\ &\text{subject to} \;\; g_{ij}(\Delta\mathbf{t}, \Delta\mathbf{r}) \leq 0 \;\; \forall i, j, \end{aligned} \tag{4.7}$$

where $t_{0,k}$ or $r_{0,k}$ are, respectively, the kth translational or rotational DOF for an initial configuration of A; similarly, $t_{1,k}$ or $r_{1,k}$ is the kth DOF for the resulting configuration after the first level optimization; Δt_k and Δr_k are the variables. Note, here the *containment function* g_{ij} is computed from every vertex of $A(\mathbf{q}_1)$ (instead of $A(\mathbf{q}_0)$) and each face of $\bar{B}$. Let us further set $u_k = \Delta t_k + t_{1,k} - t_{0,k}$ and $v_k = \Delta r_k + r_{1,k} - r_{0,k}$. In this case, we can rewrite the second level optimization problem in Eq. (4.7) as:

$$\begin{aligned} &\min \;\; Z = \sum_{k=1}^{3} |u_k| + \mathbf{R} \sum_{k=1}^{3} |v_k|, \\ &\text{subject to} \;\; g^1_{ij}(\mathbf{u}, \mathbf{v}) \leq 0 \;\; \forall i, j, \end{aligned} \tag{4.8}$$

where g^1_{ij} is obtained from g_{ij} in Eq. (4.7) by the change of variables: $\mathbf{u} = \Delta\mathbf{t} + \mathbf{t}_1 - \mathbf{t}_0$ and $\mathbf{v} = \Delta\mathbf{r} + \mathbf{r}_1 - \mathbf{r}_0$.

The objective function in the optimization system (Eq. 4.8) contains absolute arithmetic operations. We replace $|u_k|$ with $u_k^+ + u_k^-$ in the objective function, and u_k with $u_k^+ - u_k^-$ in the containment function g_{ij}^1, where $u_k^+, u_k^- \geq 0$ for $k = 1, 2, 3$. A similar replacement is performed for v_k. Finally, we formulate this optimization problem as a linear programming problem (LP2):

$$\begin{aligned}
&\min \;\; Z = \sum_k (u_k^+ + u_k^-) + \mathbf{R}(v_k^+ + v_k^-), \\
&\text{subject to } \;\; g_{ij}^2(\mathbf{u}^+, \mathbf{u}^-, \mathbf{u}^+, \mathbf{u}^-) \leq 0 \;\; \forall i, j \qquad (4.9) \\
&u_k^+, u_k^-, v_k^+, v_k^- \geq 0, k = 1, 2, 3,
\end{aligned}$$

where g_{ij}^2 is obtained from g_{ij}^1 by the change of variables.

By solving Eq. (4.9), we get $u_k^+, u_k^-, v_k^+, v_k^-$. Using the solution, we can compute $\Delta t_{1,k}^+, \Delta t_{1,k}^-, \Delta r_{1,k}^+$ and $\Delta r_{1,k}^-$, which yields a new configuration $\mathbf{q}_2$. We replace $\mathbf{q}_1$ by $\mathbf{q}_2$, and iterate this optimization process until the objective Z in Eq. (4.9) converges to a local minimum. At this stage, since $u_k^+, u_k^-, v_k^+, v_k^-$ are zeroes, our small-angle approximation becomes accurate and A is forced to be disjoint from B. After computing an optimal containing placement $\mathbf{q_2}$ for A, we compute an upper bound on $\mathrm{PD}_{\mathrm{TRAJ}}^g$ using Eq. (2.7).

4.1.3 Lower Bound for Non-Convex Objects

Our algorithm for computing a lower bound on $\mathrm{PD}_{\mathrm{TRAJ}}^g$ is based on the fact that $\mathrm{PD}_{\mathrm{TRAJ}}^g$ is equivalent to PD^t for convex polyhedra (Section 4.1.1). As a result, we compute a lower bound of $\mathrm{PD}_{\mathrm{TRAJ}}^g$ by first performing convex decomposition of the models. Next, we take the maximum value of PD_i^t's between all pairwise combinations of convex pieces. The overall algorithm proceeds as:

1. As a preprocessing, perform convex decomposition for A and B i.e., $\cup A_i = A$ and $\cup B_i = B$. Here A_i, B_i are convex sets; how they are not necessarily disjoint from

Algorithm 1 Lower bound on generalized PD computation
Input: The robot A, the obstacle B and the configuration $\mathbf{q}$
Output: The lower bound on $\mathrm{PD}^g_{\mathrm{TRAJ}}$ between $A(\mathbf{q})$ and B.

```
1: // During preprocessing
2: Decompose $A$ and $B$ into $m$ and $n$ convex pieces; i.e., $A = \cup A_i$ and $B = \cup B_j$.
3: // During run-time query
4: for each pair of $(A_i(\mathbf{q}), B_j)$ do
5:     $k = (i-1)n + j$
6:     if $A_i(\mathbf{q})$ collides with $B_j$ then
7:         ${}^k\mathrm{PD}^g_{\mathrm{TRAJ}} = \mathrm{PD}^t((A_i(\mathbf{q}), B_j)$
8:     else
9:         ${}^k\mathrm{PD}^g_{\mathrm{TRAJ}} = 0$
10:    end if
11: end for
12: return max$({}^k\mathrm{PD}^g_{\mathrm{TRAJ}})$ for all $k$.
```

each other.

2. During the run-time query, place A_i at the configuration $\mathbf{q}$, i.e. compute $A_i(\mathbf{q})$.

3. For each pair of $(A_i(\mathbf{q}), B_j)$ where $i = 1, \ldots, M$ and $j = 1, \ldots, N$,

   (a) Perform collision detection to check for overlaps.

   (b) If the pair overlaps, let ${}^k\mathrm{PD}^g_{\mathrm{TRAJ}} = \mathrm{PD}^t(A_i(\mathbf{q}), B_j)$; otherwise ${}^k\mathrm{PD}^g_{\mathrm{TRAJ}} = 0$, where $k = 1, \ldots, MN$.

4. Finally, $\mathrm{PD}^g_{\mathrm{TRAJ}} = \max({}^k\mathrm{PD}^g_{\mathrm{TRAJ}})$ for all k.

The resulting lower bound generalized PD algorithm is summarized in Algorithm 1.

Translational Penetration Depth Computation

In our method, the lower bound on generalized $\mathrm{PD}^g_{\mathrm{TRAJ}}$ computation is decomposed into a set of PD^t queries among convex primitives. The PD^t between two convex polyhedra can be computed using the algorithms presented in (Cameron, 1997; van den Bergen, 2001; Kim et al., 2002a). These methods compute PD^t by calculating the minimum distance from the origin to the surface of the Minkowski sum of the two convex polyhedra.

Since we are computing a lower bound to $\mathrm{PD}^g_{\mathrm{TRAJ}}$, this requires that the PD^t computation algorithm used by our method should compute an exact value or a lower bound to the PD^t. In particular, the algorithm proposed by Cameron (1997) satisfies this requirement, and van den Bergen (2001) algorithm also provides a tight lower bound.

Acceleration using Bounding Volume Hierarchy

Our computation of a lower bound on $\mathrm{PD}^g_{\mathrm{TRAJ}}$ can be accelerated by employing a standard bounding volume hierarchy. For a pair of convex pieces (A_i, B_j) which are disjoint, PD^t corresponds to zero. In practice, there are many disjoint pairwise combinations of convex pieces (A_i, B_j). We detect such disjoint pairs by using an oriented bounding box (OBB) hierarchy (Gottschalk et al., 1996), and prune them away.

4.1.4 Upper Bound for Non-convex Objects

One simple way to compute an upper bound to $\mathrm{PD}^g_{\mathrm{TRAJ}}$ for general non-convex objects is to compute the PD^t between their convex hulls. This corresponds to an upper bound because $\mathrm{PD}^g_{\mathrm{TRAJ}}(A, B) \leq \mathrm{PD}^g_{\mathrm{TRAJ}}(\mathrm{CH}(A), \mathrm{CH}(B))$, and the latter is equal to $\mathrm{PD}^t(\mathrm{CH}(A), \mathrm{CH}(B))$, thanks to Theorem 3. In practice, this upper bound is relative simple to compute. However, this algorithm could be overly conservative for non-convex models, as shown in Figs. 2.3 and 4.2.

$\mathrm{PD}^t(A, B)$ is also an upper bound on $\mathrm{PD}^g_{\mathrm{TRAJ}}(A, B)$. However, this can result in a conservative upper bound in practice too. Since the computational complexity of exact computation of $\mathrm{PD}^t(A, B)$ for non-convex models can be high, current approaches typically compute an upper bound of $\mathrm{PD}^t(A, B)$ (Kim et al., 2002b).

We present an algorithm to compute an upper bound on $\mathrm{PD}^g_{\mathrm{TRAJ}}$ for non-convex polyhedra by reducing this problem to a set of containment optimization sub-problems (as defined in Section 4.1.2).

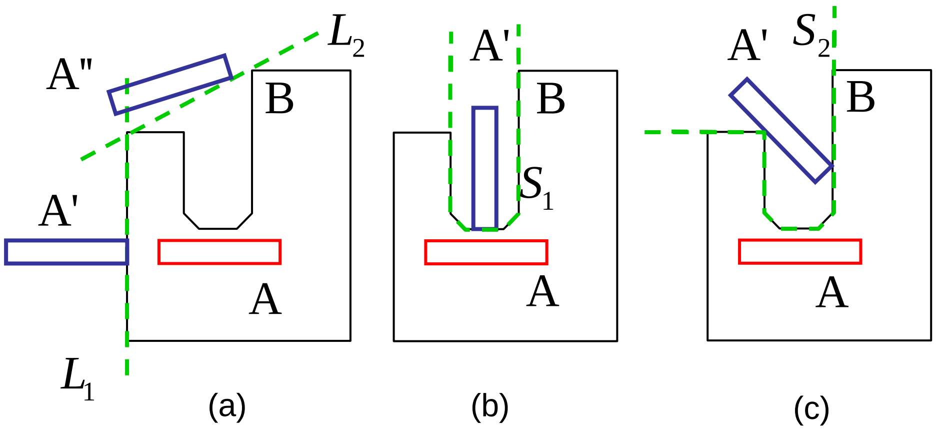


Figure 4.3: Separating plane, convex separator and non-convex separator. *(a). L_1 and L_2 are separating planes, which separate A' and B, and A'' and B respectively. (b). S_1 is a separator, which is composed by a set of piece-wise linear plane. S_1 separates A' from B. A separator is called convex (i.e. S_1), if it lies on the boundary of its convex hull. (c). A non-convex separator S_2 separates A from B'.*

Algorithm Overview

Given two disjoint non-convex objects A and B, there is either a single separating plane between the objects (as shown in Fig. 4.3(a)), or there is a set of piecewise linear surfaces, which is called a *separator* (Mount, 1992) (Figs. 4.3(b) and (c)). More precisely, a separator is defined as a simple piece-wise linear surface that divides the space into two half-spaces. The separator can be an open surface or a closed surface. A separator S is convex if and only $S \subset \partial(CH(S))$, as shown in Fig. 4.3(b). Otherwise, the separator is non-convex, as shown in Fig. 4.3(c). A single separating plane can be regarded as a special case of a separator. However, we specifically use the term separator to refer to the non-plane separator.

Our upper bound $\mathrm{PD}^g_{\mathrm{TRAJ}}(A, B)$ computation algorithm proceeds as follows: during the preprocessing phase, we enumerate all possible separating planes and convex separators by analyzing the convexity of the boundary of B. During the query phase, for each separating plane L (or each convex separator S), we compute an upper bound

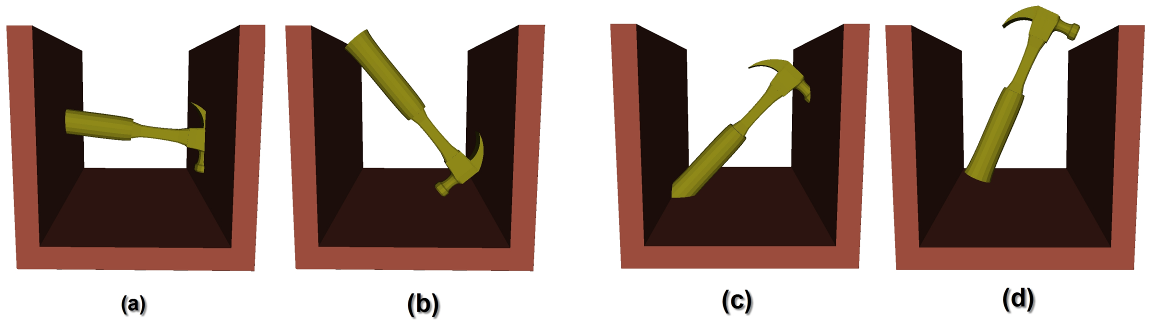


Figure 4.4: The 'hammer' example. *(a) When the 'hammer' is at time t=0, it collides with the 'notch'. (b) The collision-free placement of the 'hammer' for scenario (a). We use our containment optimization algorithm to get this free configuration, which realizes the $UB_1(\mathrm{PD}^g_{\mathrm{TRAJ}})$. (c) The 'hammer' at time t=0.5. (d) The collision-free placement is computed for scenario to get the $UB_1(\mathrm{PD}^g_{\mathrm{TRAJ}})$*

on TRAJ distance when A is separated from B with respect to the separating plane L (or separator S), using the technique described in Sec. 4.1.2. The minimum of all these upper bounds yields a global upper bound on $\mathrm{PD}^g_{\mathrm{TRAJ}}$. Now we explain how to efficiently enumerate L and S as part of the preprocessing step.

Separating Planes

The set of all possible separating planes is included in the complement of the convex hull of B. According to Theorem 3, $\mathrm{PD}^g_{\mathrm{TRAJ}} = \mathrm{PD}^t$ for convex objects, and the minimum TRAJ distance with respect to all these separating planes is $\mathrm{PD}^t(\mathrm{CH}(A), \mathrm{CH}(B))$. This means that the computation of $\mathrm{PD}^t(\mathrm{CH}(A), \mathrm{CH}(B))$ implicitly takes into account all possible separating planes. Therefore, we need not enumerate any separating planes explicitly during the preprocessing phase.

Convex Separators

Any separator S divides the whole space into two half-spaces. One half-space would include the object B. We can regard the other half-space as a container. Placing A inside the container is equivalent to making A and B disjoint with respect to each separator S. Therefore, the computation of the minimum TRAJ distance for S can be regarded as a 3D convex containment optimization problem. By applying two levels of linear programming optimization algorithm discussed in Sec. 4.1.2, we compute an upper bound of $\mathrm{PD}^g_{\mathrm{TRAJ}}$ for each convex separator S. The minimum of all $\mathrm{PD}^g_{\mathrm{TRAJ}}$ over all enumerated convex separators yields an upper bound on $\mathrm{PD}^g_{\mathrm{TRAJ}}$.

Convex Separators Enumeration

Enumerating convex separators of B can be performed as a preprocessing. This step can be regarded as computing a convex decomposition of the complement space of B. Given the fact that we are computing an upper bound of $\mathrm{PD}^g_{\mathrm{TRAJ}}$, the conservativeness of the separator enumeration does not affect the correctness of our algorithm.

We use the surface convex decomposition for the complement space of B (Ehmann and Lin, 2001). We discard the surface with one face from the surface decomposition, since these planes have been processed as separating planes.

Moreover, if the geometry of input A and B is very complex we simplify each primitive to compute a coarser model A', B'. If $A \subseteq A'$ and $\bar{B} \subseteq \bar{B}'$, it is easy to prove that $\mathrm{PD}^g_{\mathrm{TRAJ}}(A, B) \leq \mathrm{PD}^g_{\mathrm{TRAJ}}(A', B')$. Therefore, we can compute the upper bound by applying our algorithm on these simplified models.

Separator Culling

We can cull some of the separators by making use of the currently known upper bound on $\mathrm{PD}^g_{\mathrm{TRAJ}}$ during any stage of the algorithm. If the minimum distance between the separator and the object A is larger than the current upper bound, we can discard this separator. We use the PD^t between the two convex hulls of input models as an initial upper bound of $\mathrm{PD}^g_{\mathrm{TRAJ}}$.

4.1.5 Complexity

The complexity of the lower bound algorithm is governed by the number of convex pieces that are obtained from the decomposition, and the geometric complexity of each convex piece, e.g. the number of vertices of the convex piece. Let n_1, n_2 denote the number of convex pieces of the robot A and the obstacle B, respectively. Let the geometric complexity of all convex pieces of A and B be a and b, respectively. Then, the average numbers of features in each piece of A and B are $\frac{a}{n_1}$ and $\frac{b}{n_2}$, respectively. Using the complexity of translational PD, we can derive that the complexity of lower bound generalized PD computation for 2D rigid objects A and B is $O(an_2 + bn_1)$, and for 3D rigid objects is $O(ab)$. The main computational component of the upper bound algorithm is the containment optimization using linear programming. Each iteration within the containment optimization is governed by the number of convex separators as well as their geometric complexity (i.e. the number of facets). However, the overall convergence of the optimization algorithm is difficult to analyze since it is dependent on the shapes of A and the convex separators.

4.1.6 Implementation and Performance

We have implemented our lower and upper bound computation algorithms for generalized PD between 3D rigid non-convex models. We have tested our algorithms to compute on a set of benchmarks. In this section, we only present the implementation and experimental results on generalized PD based on TRAJ metric. The implementation can also be extended for other C-space distance metrics. All the timings reported in this section were taken on a 2.8GHz Pentium IV PC with 2GB of memory.

Implementation of Lower Bound Algorithm

In our implementation, the convex decomposition is performed as a preprocessing step. Currently, we use the convex surface decomposition algorithm proposed by Ehmann and

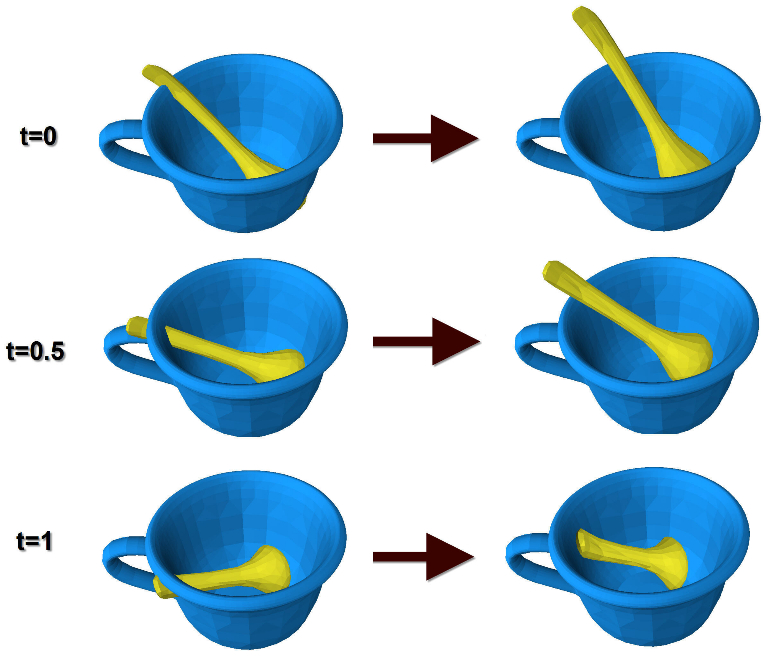


Figure 4.5: The 'cup' example. *The left column shows the placements of the 'spoon' in the 'cup', when t=0.0, t=0.5, and t=1.0, respectively. At all of these placements, the 'spoon' collides with the 'cup'. The right column shows the collision-free configurations which are realized for* $UB_1(\mathrm{PD}^g)$ *at each t.*

Lin (2001), which can be regarded as a special case of convex decomposition. In order to compute PD^t between two convex polytopes, we use the implementation available as a part of SOLID package (van den Bergen, 2001). In order to accelerate this algorithm, we precompute an OBB hierarchy (Gottschalk et al., 1996) and use the bounding volumes to conservatively cull convex pairs that do not intersect with each other.

Implementation of Upper Bound Algorithm

The preprocessing step of convex separator enumeration can be regarded as convex decomposition of the complement of the input model. In our implementation, we use the surface decomposition algorithm to generate a set of convex surfaces (Ehmann and Lin, 2001) and discard the surfaces that have only one face. For each convex separator, we use the containment optimization technique developed in Sec. 4.1.2 to compute

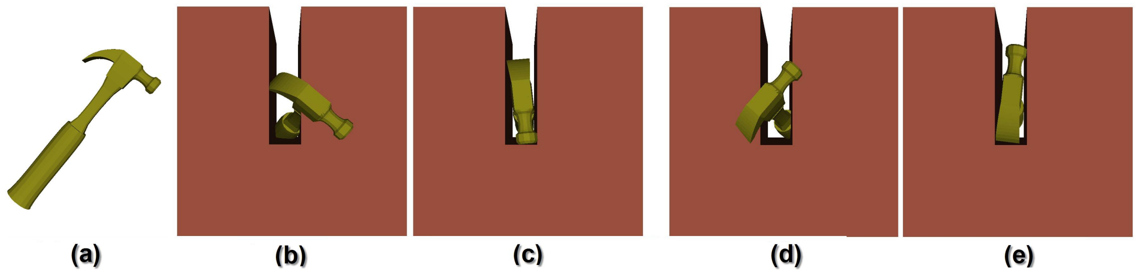


Figure 4.6: The 'hammer in narrow notch' example. *This example is modified from the 'hammer' example, where the size of the notch is decreased such that there is only narrow space for the 'hammer' to fit inside. (b) and (d) show the placement of the 'hammer' at t=0 and t=0.5. (c) and (e) are their corresponding configurations, respectively, which realize the* $UB_1(\mathrm{PD}^g)$. *The computed* $UB_1(\mathrm{PD}^g)$ *is tighter than the* $UB_2(\mathrm{PD}^g)$ *for most of time* t.

an upper bound on $\mathrm{PD}^g_{\mathrm{TRAJ}}$. In particular, we use the *QSopt* [1] package to solve the linear programming problems. In order to accelerate the upper bound computation, we conservatively cull any convex separator, if the minimum distance between this separator and the object A is larger than the current upper bound on generalized PD.

Performance

We use different benchmarks to test the performance. Our experimental setup is as follows: each benchmark includes two polyhedral models A and B, where A is movable and B is fixed. The model A is assigned a starting configuration $\mathbf{q}_0$ and an end configuration $\mathbf{q}_1$. We linearly interpolate between these two configurations with n intermediate configurations (i.e. n samples). For each interpolated configuration $\mathbf{q} = (1-t)\mathbf{q}_0 + t\mathbf{q}_1$, $t \in [0,1]$, we compute various bounds for generalized PD between $A(\mathbf{q})$ and B, including:

1. $LB(\mathrm{PD}^g)$: The lower bound on $\mathrm{PD}^g_{\mathrm{TRAJ}}$ based on pairwise translational PD^t computation.
2. $UB_1(\mathrm{PD}^g)$: The upper bound on $\mathrm{PD}^g_{\mathrm{TRAJ}}$ computed by containment optimization.

[1]http://www2.isye.gatech.edu/~wcook/qsopt/

3. $UB_2(\mathrm{PD}^g)$. The upper bound on generalized PD based on the translational PD computation between their convex hulls.

In order to get accurate timing profiling, we run our PD algorithms for each configuration with a batch number b. The average time for each bound computation is the total running time on all samples, divided by the number of samples and the batch number b.

'Hammer' example. Fig. 4.4, Tab. 4.1, and Fig. 4.8(a) show the results and timings for the 'hammer' example. In this case, the 'hammer' model has 1,692 triangles, and is decomposed into 214 convex pieces. The 'notch' model has 28 triangles. It is decomposed into 3 convex pieces, including a notch (i.e. convex separator) in the center of the 'notch' model. Initially (at t=0), the 'hammer' intersects with the 'notch' as shown in Fig. 4.4(a). Fig. 4.4(b) shows a collision-free placement of the 'hammer', which corresponds to the position after moving by $UB_1(PD^g)$. According to Fig. 4.8(a), the value is $UB_1(\mathrm{PD}^g) = 4.577083$, which is greater than $LB(\mathrm{PD}^g)$ (0.744020) and less than $UB_2(\mathrm{PD}^g)$ (6.601070).

For this example, we generate 101 samples for the 'hammer' when it is rotated around the Z axis. The rotation motion is linearly interpolated from the configuration $(0, 0, 0)^T$ to $(0, 0, \pi)^T$. Fig. 4.4(c) shows the placement of the 'hammer' at $t = 0.5$. Fig. 4.4(d) is the corresponding collision-free placement, which realizes the $UB_1(\mathrm{PD}^g)$.

We also compare the lower and upper bounds on generalized PD over all the configurations. In Fig. 4.8(a), the solid green curve highlights the value of $UB_1(\mathrm{PD}^g)$ between the 'hammer' and the 'notch' over all interpolated configurations. The dashed red curve, which corresponds to $UB_1(\mathrm{PD}^g)$, always lies between $LB(\mathrm{PD}^g)$ and $UB_2(\mathrm{PD}^g)$. In this example, $UB_1(\mathrm{PD}^g)$ is less than $UB_2(\mathrm{PD}^g)$.

The timing for this example is shown in Tab. 4.1. We run the generalized PD algorithm 5 times (b=5) for all the configurations (n=101). The average timing for $LB(\mathrm{PD}^g)$, $UB_1(\mathrm{PD}^g)$, and $UB_2(\mathrm{PD}^g)$ is 1.901ms, 21.664ms and 0.039ms, respectively.

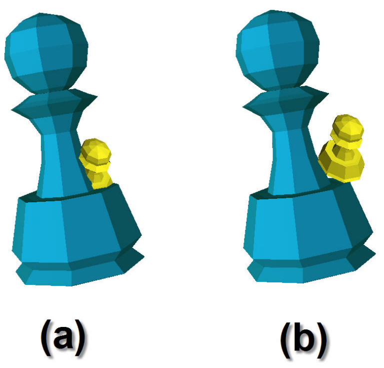


Figure 4.7: The ‘pawn’ example. *The large ‘pawn’ is fixed and the small one is movable. (a) shows the colliding placement of the ‘pawn’ at $t = 0$. (b) shows its corresponding collision-free placement, which is computed based on $UB_1(\mathrm{PD}^g)$.*

‘Hammer in narrow notch’ example. We perform a similar experiment on ‘Hammer in narrow notch’ example (Fig. 4.6) to test the robustness of our algorithm. This example is modified from the ‘hammer’ example, where the size of the notch is decreased such that there is only narrow space for the ‘hammer’ to fit inside. Our algorithm can robustly compute the lower and upper bounds on generalized PD for this example. Fig. 4.8(b) compare the lower and upper bounds over all sampled configurations (n=101). The third row of Tab. 4.1 shows the performance of our algorithm for this example.

‘Spoon in cup’ example. We apply our algorithm to a more complex scenario shown in Fig. 4.5. In this example, the ‘spoon’ model has 336 triangles and is decomposed into 28 convex pieces. The ‘cup’ model has 8,452 triangles and is simplified into a model with 1,000 triangles. We get 94 convex pieces and 53 convex separators from the simplified model.

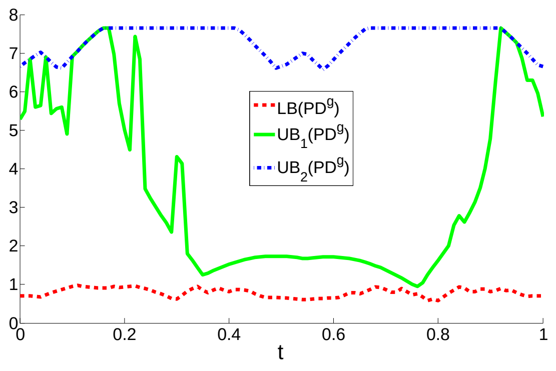
In Fig. 4.5, the left column shows the placements of the ‘spoon’ in the ‘cup’, corresponding to $t = 0.0$, $t = 0.5$, and $t = 1.0$, respectively. At all these placements, the ‘spoon’ collides with the ‘cup’. The right column of this figure shows the collision-free configurations that are computed based on $UB_1(\mathrm{PD}^g)$ in each case. We also compare our computed lower bound and upper bounds over all the samples (n=101), which are

	Hammer	H2 $\star$	Spoon	Pawn
A	Hammer	Hammer	Spoon	Small pawn
tris #	1,692	1,692	336	304
convex pieces #	215	215	28	44
B	Notch	Notch	Cup	Large panw
tris #	28	28	8,452	304
convex pieces #	3	3	94	44
separator #	1	1	53	43
sample # (n)	101	101	101	101
batch # (b)	5	5	5	5
t for LB (ms)	1.901	4.300	6.127	4.112
t for UB_1 (ms)	21.664	108.024	1027.014	482.511
t for UB_2 (ms)	0.039	0.053	0.154	0.055

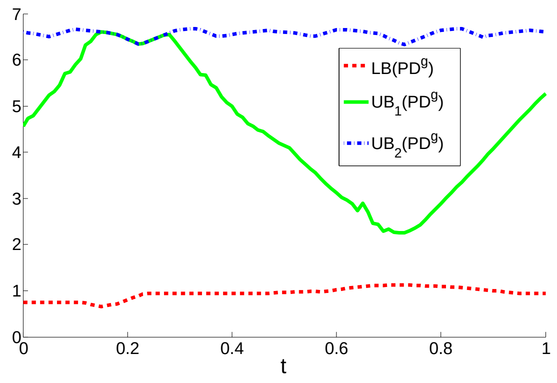
Table 4.1: Performance of convexity-based generalized PD computation. *This table highlights the benchmarks used to test the performance of our convexity-based algorithm. The top rows in the table list the model complexity, and the bottom rows report the time taken to compute the lower and upper bounds on generalized PD. 'H2$\star$' is the example 'hammer in narrow notch'.*

shown in Fig. 4.8(c). The timing performance for this example is also listed on Tab. 4.1.

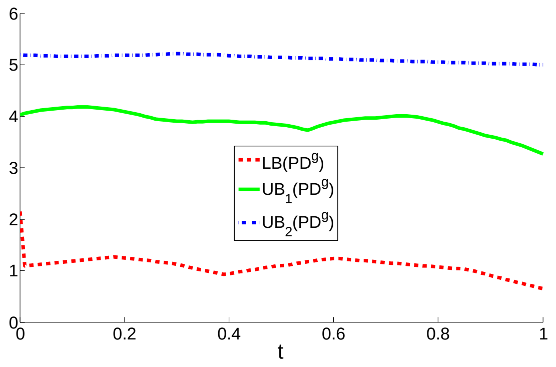
'Pawn' example. The last benchmark used to demonstrate the performance of our algorithm is the 'pawn' example. As Fig. 4.7 shows, the large 'pawn' is fixed, while the small one is moving. The 'pawn' model has 304 triangles and is decomposed into 44 convex pieces. The large 'pawn' has 43 convex separators. Fig. 4.7(a) shows the colliding placement of the 'pawn' at t = 0. Fig. 4.7(b) shows its corresponding collision-free placement, which is computed based on $UB_1(\mathrm{PD}^g)$. Fig. 4.8(d) highlights the comparison of the lower bound and upper bounds over the sampled configurations (n=101). Tab. 4.1 shows the average time to compute the lower and upper bounds over all configurations.



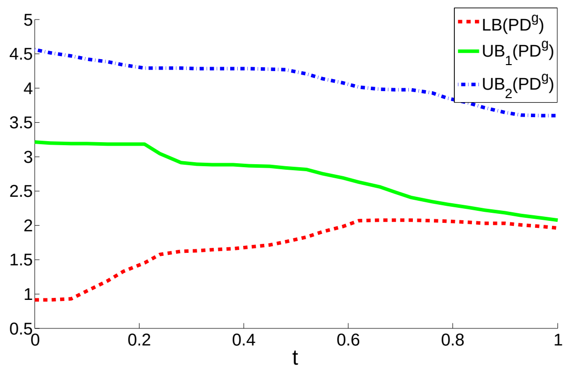
(a) 'hammer'



(b) 'hammer in narrow notch'



(c) 'cup'



(d) 'pawn'

Figure 4.8: Comparison of lower and upper bounds on generalized PD computation. *(a) The lower and upper bounds on generalized PD between the 'hammer' and the 'notch' models are computed over all interpolated configurations. The dash-dot blue curve $LB(\mathrm{PD}^g)$ stands for the lower bound of generalized PD by computing pairwise translational PD. The dashed red curve $UB_2(\mathrm{PD}^g)$ stands for the upper bound of generalized PD computed by the translational PD of their convex hulls. The solid green curve $UB_1(\mathrm{PD}^g)$ highlights the upper bound of PD^g by using our containment optimization, which always lies between $LB(\mathrm{PD}^g)$ and $UB_2(\mathrm{PD}^g)$. In this example, $UB_1(\mathrm{PD}^g)$ is less than $UB_2(\mathrm{PD}^g)$ for most of time t. (b,c,d) we compare the results of the other three examples.*

4.2 A Fast and Practical Generalized PD Algorithm using Constrained Optimization

In this section, we present a fast and practical algorithm to compute generalized PD for rigid, non-convex models (Zhang et al., 2007a). Based on model-dependent distance metrics, we have shown that the optimum answer lies in the contact space (Thm. 1). Therefore, we can pose the generalized PD computation as a constrained optimization problem. We use global approaches to compute an initial guess. We further present efficient techniques to compute a local approximation of the contact space, and incrementally refine the solution in the local contact space along the maximally-decreasing direction of the distance cost. The algorithm makes no assumption about model topology. We highlight the performance of our algorithm on many complex models. In practice, our algorithm takes about a few hundred milli-seconds on non-convex models composed of a few thousand triangles.

4.2.1 Algorithm Overview

As shown in Fig. 4.9, for a colliding configuration $\mathbf{q_o}$ of the robot, the goal is to compute a (locally) closest configuration in the contact space according to a distance metric δ. Our algorithm can optimize the objective function defined by δ by performing incremental refinement in the contact space. The iterative optimization algorithm (Algorithm 2) consists of three major steps:

1. Given an initial guess of contact configuration $\mathbf{q_a}$, the algorithm first computes a local approximation $\mathcal{L}_{\mathbf{q_a}}$ of the contact space around $\mathbf{q_a}$ (Line 3 of Algorithm 2).

2. The algorithm searches over the local approximation to find a new configuration $\mathbf{q_b}$ that minimizes the objective function δ (Line 4).

3. The algorithm assigns $\mathbf{q_b}$ as a starting point for the next iteration (i.e. *walk* from

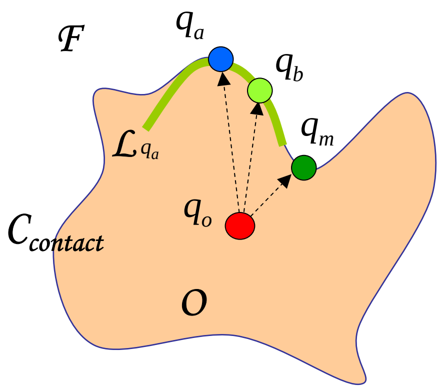


Figure 4.9: Constrained optimization. *We reduce the generalized PD problem to computing a closest configuration in the contact space for* $\mathbf{q_o}$ *by iterative optimization.*

$\mathbf{q_a}$ to $\mathbf{q_b}$) if $\mathbf{q_b}$ is in the contact space and it has a smaller value of the objective function as compared to $\mathbf{q_a}$'s. Otherwise, we compute a new contact configuration $\mathbf{q_b}'$ based on $\mathbf{q_b}$ (Line 5-14).

These steps are iterated until a local minimum configuration $\mathbf{q_m}$ is found or the maximum number of iterations is reached. We address each of these steps in more detail.

4.2.2 Local Contact Space Approximation

Since it is computationally prohibitive to compute a global representation of the contact space $\mathcal{C}_{contact}$, our algorithm computes a local approximation. Given a contact configuration $\mathbf{q_a}$, where A is in contact with B, we enumerate all contact constraints according to the pairs of contact features (Latombe, 1991; Xiao and Ji, 2001). We further decompose each contact constraint into *primitive contact constraints*, i.e. vertex/face $(v-f)$, face/vertex$(f-v)$ or edge/edge $(e-e)$. Conceptually, each primitive contact constraint represents a halfspace, and the set of all primitive constraints are used to characterize the local contact space. Finally, we obtain a local approximation $\mathcal{L}_{\mathbf{q_a}}$ of $\mathcal{C}_{contact}$ around the contact configuration $\mathbf{q_a}$ after concatenating all these primitive constraints $\{C_i\}$ using proper intersection or union operators $\{\circ_i\}$:

Algorithm 2 Generalized PD Algorithm using Constrained Optimization
Input: two overlapping 3D rigid models: A - movable, B - static.
$\mathbf{q_o}$:= a colliding configuration of A, $\mathbf{q_o} \in \mathcal{O}$.
$\mathbf{q_a}$:= an initial guess of contact configuration for A, $\mathbf{q_a} \in \mathcal{C}_{contact}$.
Output: $\mathrm{PD}^g_\delta(A, B)$

```
1:  repeat
2:      i++;
3:      $\mathcal{L}_{\mathbf{q_a}}$ := Local contact space approximation at $\mathbf{q_a}$;
4:      $\mathbf{q_b} := \arg\min\{\delta(\mathbf{q_o}, \mathbf{q}), q \in \mathcal{L}_{\mathbf{q_a}}\}$;
5:      if $\delta(\mathbf{q_o}, \mathbf{q_b}) == \delta(\mathbf{q_o}, \mathbf{q_a})$ then
6:          return $\delta(\mathbf{q_o}, \mathbf{q_a})$;
7:      else if $\mathbf{q_b} \in \mathcal{C}_{contact}$ then
8:          $\mathbf{q_a} := \mathbf{q_b}$;
9:      else if $\mathbf{q_b} \in \mathcal{F}$ then
10:         $\mathbf{q_a}$ := Bisection($\mathbf{q_o}$, $\mathbf{q_b}$);
11:     else
12:         $\mathbf{q_b}'$ := CCD($\mathbf{q_a}$, $\mathbf{q_b}$);
13:         $\mathcal{L}_{\mathbf{q_a}} := \mathcal{L}_{\mathbf{q_a}} \bigcap \mathcal{L}'_{\mathbf{q_b}}$;
14:         goto 4;
15:     end if
16: until $i < MAX\_ITERATION$
```

$$\mathcal{L}_{\mathbf{q_a}} = \{C_1 \circ_1 C_2 \cdots \circ_{n-1} C_n\}. \tag{4.10}$$

It should be noted that we do not explicitly compute a geometric representation of $\mathcal{L}_{\mathbf{q_a}}$. Instead, it is algebraically represented, and each primitive constraint is simply recorded as a pair of IDs, identifying the contact features from A and B, respectively.

When decomposing each constraint into primitive constraints, we need to choose proper Boolean operators to concatenate the resulting primitive constraints. This issue has been addressed in the area of dynamics simulation (Egan et al., 2003) and we address it in a similar manner for generalized PD computation. Fig. 4.10 shows a 2D example with a triangle-shaped robot A touching a notch-shaped obstacle B. When decomposing a $v - v$ contact constraint into two $v - e$ constraints C_1 and C_2, if both of A and B are convex at the contact vertices (Fig. 4.10(a)), we use a union operator, because if either constraint C_1 or C_2 is enforced, there is no local penetration. Otherwise, if either

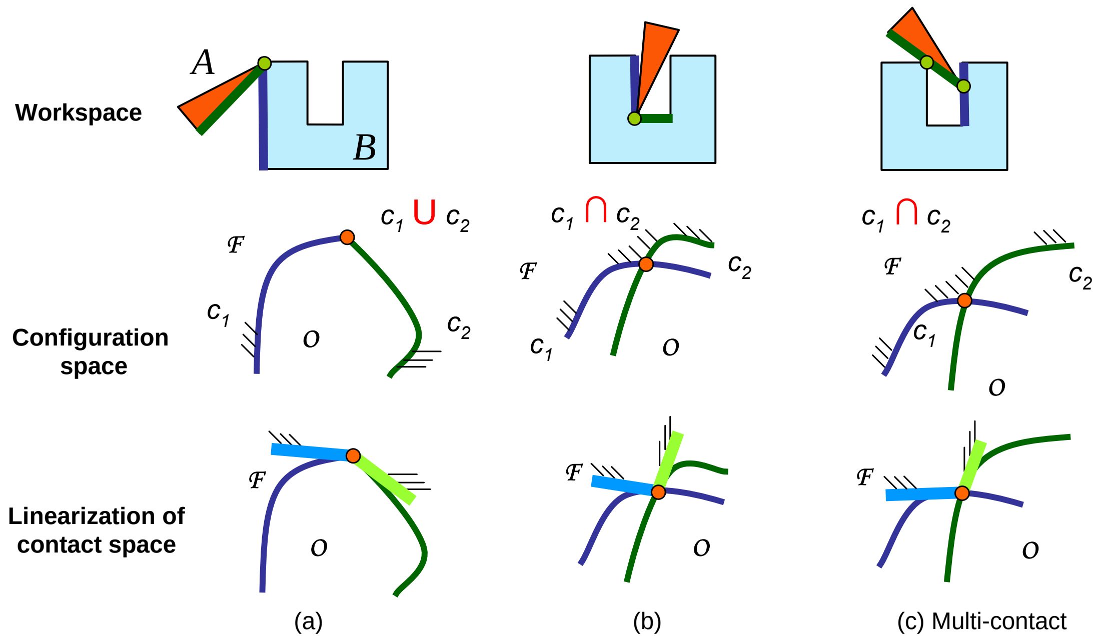


Figure 4.10: Local contact space approximation. *The local contact space is algebraically represented as a set of contact constraints concatenated with intersection or union operators (Eq. 4.10). Columns (a) and (b) explain how to obtain proper operators when decomposing a constraint into primitive contact constraints using 2D examples (Section 4.2.2). Column (c) shows a multiple contact situation. The last row illustrates the corresponding linearization for each local contact space.*

model at the contact vertex is non-convex (Fig. 4.10(b)), the intersection operation is used. For 3D models, a similar analysis is performed by identifying the convexity of edges based on their dihedral angles. In case of multiple contacts, one can first use intersection operations to concatenate all the constraints. Each individual constraint is then further decomposed into primitive constraints properly.

4.2.3 Searching over Local Contact Space

Given a local contact space approximation $\mathcal{L}_{\mathbf{q_a}}$ of the contact configuration $\mathbf{q_a}$, we search over $\mathcal{L}_{\mathbf{q_a}}$ to find $\mathbf{q_b}$ that minimizes the objective function. Since the contact space is a non-linear subspace of $\mathcal{C}$, we use two different search methods: random sampling in $\mathcal{L}$ and optimization over a first-order approximation of $\mathcal{L}$. Each of them can be performed independently.

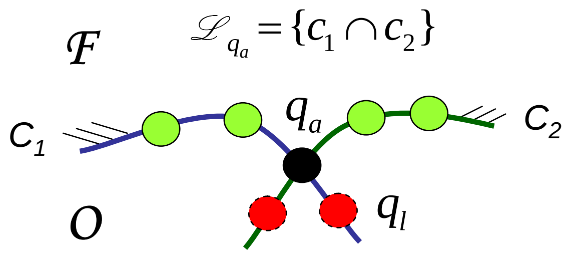


Figure 4.11: Sampling in local contact space. $\mathcal{L}_{\mathbf{q_a}}$ *is a local approximation of contact space around* $\mathbf{q_a}$*, represented by the intersection of its contact constraints* C_1 *and* C_2*. Our algorithm randomly generates samples on* C_1 *and* C_2*. Many potentially infeasible samples, such as* $\mathbf{q}_l$*, can be discarded since they are lying outside the halfspace of* C_2*.*

Sampling in Local Contact Space

Our algorithm randomly generates samples on the local contact approximation $\mathcal{L}_{\mathbf{q_a}}$ around $\mathbf{q_a}$ (Fig. 4.11), by placing samples on each primitive contact constraint C_i as well as on their intersections (Ji and Xiao, 2000). We discard any generated sample $\mathbf{q}$ if it lies outside of the halfspace formulated by $\mathcal{L}_{\mathbf{q_a}}$ by simply checking the sign of $\mathcal{L}_{\mathbf{q_a}}(\mathbf{q})$. Since $\mathcal{L}_{\mathbf{q_a}}$ is a local contact space approximation built from all contact constraints, this checking of $\mathcal{L}$ allows us to cull potentially many infeasible colliding configurations. For the rest of the configuration samples, we evaluate their distances δ to the initial configuration $\mathbf{q_o}$, and compute the minimum.

These samples are efficiently generated for each non-linear contact constraint C_i. First, we generate random values for the rotation parameters. By plugging these values into a non-linear contact constraint, we formulate a linear constraint for the additional translation parameters. Under the formulated linear constraint, random values are generated for these translation parameters.

In practice, an optimal solution for generalized PD may correspond to multiple contacts, suggesting that one needs to generate more samples on the boundary formed by multiple contact constraints. As a result, we set up a system of non-linear equations for each combination of these constraints, generate random values for the rotation parameters in the system (thereby making the system linear), and sample the resulting linear system for the translation parameters.

Linearizing the Local Contact Space

We search for a configuration with smaller distance in the contact space by linearly approximating the contact space. For each basic contact constraint C_i, we compute its Jacobian, which is the normal of the corresponding parameterized configuration space. Using this normal, we obtain a half-plane, which is a linearization of the contact surface (Ruspini and Khatib, 1997; Redon and Lin, 2005). By concatenating the half-planes using Boolean operators $\circ_i$, we generate a non-convex polyhedral cone, which serves as a local linear approximation of $\mathcal{C}_{contact}$. For simplicity, one can only use intersection operators for $\circ_i$ (Redon and Lin, 2005) and obtain a convex cone, which is a subset of the original non-convex cone.

Local Search

The sampling-based local search method is general for any distance metric. Moreover, we can generate samples on each non-linear contact constraint efficiently. Finally, using the local contact space approximation, our method can cull many potentially infeasible samples.

On the other hand, the method of linearizing the contact space is suitable for optimizing generalized PD, if the underlying objective has a closed form. For example, for the OBNO metric, we transform the coordinate in the quadratic function in Eq. (2.5), from an elliptic form to a circular one. Now, the problem of searching over $\mathcal{L}$ reduces to finding the closest point in the Euclidean space from $\mathbf{q_a}$ to the non-convex polyhedral cone, formulated using the linearization of $\mathcal{L}$. Note that the computation of the closest point to the non-convex cone can be difficult. In practice we can use the simplified convex cone.

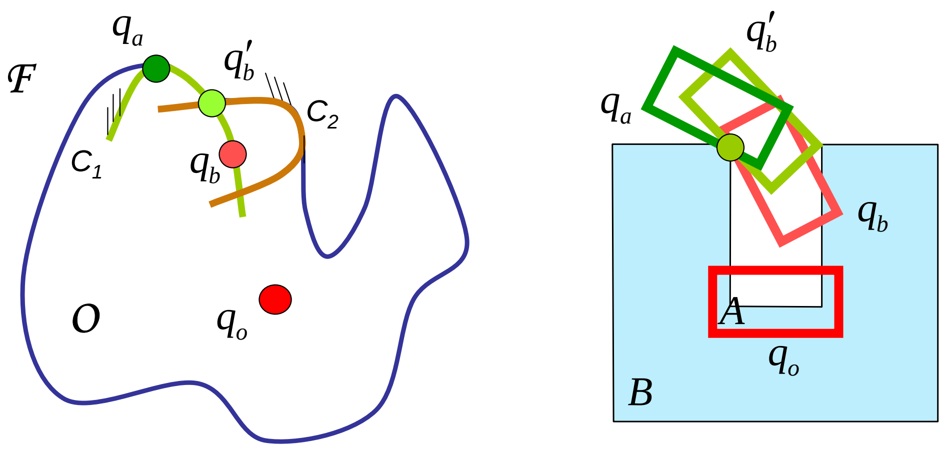


Figure 4.12: Local refinement. *Left: using the local contact space representation of* $\mathbf{q_a}$*, which includes only one constraint* C_1*, we obtain new configuration* $\mathbf{q_b}$*. Though* $\mathbf{q_b}$ *is still on* C_1*, it may not be in the contact space any more, since it will violate other constraint, such as* C_2 *here. The right figure shows a dual example happening in the workspace. When* A *slides on* B*, i.e. from* $\mathbf{q_a}$ *to* $\mathbf{q_b}$*, a collision can be created by other portions of the models. Our algorithm uses CCD to compute a correct, new contact configuration* $\mathbf{q_b}'$*.*

4.2.4 Refinement

Although searching over the local contact space $\mathcal{L}$ around $\mathbf{q_a}$ can yield a new configuration $\mathbf{q_b}$ that improves the optimization objective of $\mathbf{q_a}$, we still need to check whether $\mathbf{q_b}$ is a valid contact configuration before advancing to it because $\mathbf{q_b}$ is computed based upon a local approximation of contact space and $\mathbf{q_b}$ may not be in the contact space.

For instance, the new configuration $\mathbf{q_b}$ may be a collision-free configuration due to the first-order approximation. To handle this case, we project $\mathbf{q_b}$ back to $\mathcal{C}_{contact}$ by computing the intersection $\mathbf{q_b}'$ between the contact space and a curve interpolating from $\mathbf{q_o}$ to $\mathbf{q_b}$ using screw motion. Since $\mathbf{q_o}$ is in $\mathcal{O}$ and $\mathbf{q_b}$ is free, the intersection $\mathbf{q_b}'$ can be efficiently computed by recursive bisection (Line 10 in Algorithm 2). If the middle point $\mathbf{q}_{mid}$ on the screw motion between $\mathbf{q_b}$ and $\mathbf{q_o}$ is an colliding configuration, bisect between $\mathbf{q_b}$ and $\mathbf{q}_{mid}$; otherwise bisect between $\mathbf{q}_{mid}$ and $\mathbf{q_o}$. Once we compute the intersection $\mathbf{q_b}'$, according to the contact space realization theorem (Thm. 1), $\delta(\mathbf{q_o}, \mathbf{q_b}') < \delta(\mathbf{q_o}, \mathbf{q_b})$. Therefore, we are guaranteed to obtain a new configuration $\mathbf{q_b}'$, which is closer to $\mathbf{q_o}$, and thus it can be used for successive iterations.

It is also possible that the new configuration $\mathbf{q_b}$ may be a colliding configuration. As Fig. 4.12 on the left shows, when moving from $\mathbf{q_a}$ to $\mathbf{q_b}$, the contact constraint C_1 is maintained. However, $\mathbf{q_b}$ is a colliding configuration as it does not satisfy the new constraint C_2. The figure on the right highlights this scenario in the workspace. When A moves from $\mathbf{q_a}$ to $\mathbf{q_b}$, the contact is still maintained. In order to handle this case, we use *continuous collision detection* (CCD) to detect the time of first collision when an object continuously moves from one configuration to another using a linearly interpolating motion in $\mathcal{C}$ (Redon, 2004; Zhang et al., 2006). In our case, when A moves from $\mathbf{q_a}$ to $\mathbf{q_b}$, we ignore the sliding contact of $\mathbf{q_a}$, and use CCD to report the first contact $\mathbf{q_b}'$ before the collision (Line 12 in Algorithm 2). The new configuration $\mathbf{q_b}'$ can be used to update the local approximation of $\mathbf{q_a}$ (Line 13 in Algorithm 2). This yields a more accurate contact space approximation and consequently improves the local search, e.g. culling away additional invalid samples.

4.2.5 Initial Guess

The performance of the generalized PD algorithm depends on a good initial guess. For many applications, including dynamic simulation and haptic rendering, the motion coherence can be used to compute a good initial guess. For some other applications in which no motion coherence could be exploited, we propose a heuristic. Our method generates a set of samples in the contact space as a preprocess. At runtime, given a query configuration $\mathbf{q_o}$, our algorithm searches for the K nearest neighbors from the set of precomputed samples imposing the inter-distance between any pair of these K samples should be greater than some threshold. The distance metric used for nearest neighbor search is the same as the one to define generalized PD. The resulting K samples serve as initial guesses for our generalized PD algorithm. To generate samples in the contact space, we randomly sample the configuration space and enumerate all pairs of

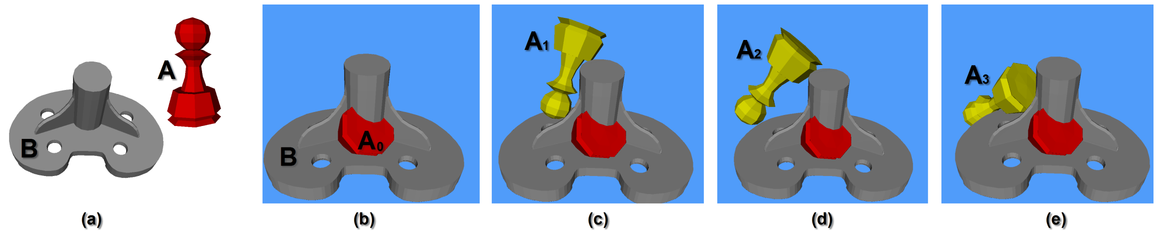


Figure 4.13: The 'CAD part' example. *(a) the models A - 'pawn' and B - 'CAD part' are used in this test. (b) a typical generalized PD query scenario is illustrated where the model A at A_0 overlaps with B. A_1 and A_2 are intermediate configurations of A during the optimization for* $\mathrm{PD}^g_{\mathrm{DISP}}$. *$A_3$ is the solution for an upper bound of* $\mathrm{PD}^g_{\mathrm{DISP}}$. *The sequence of images (c,d,e) illustrates that our algorithm incrementally slides the model 'pawn' on the model 'CAD part' to minimize the* DISP *distance to its original configuration A_0.*

free and collision samples. For each pair, a contact configuration can be computed by a bisection method.

4.2.6 Implementation and Performance

We have implemented our generalized PD algorithm using local contact space sampling for general rigid non-convex models. In this section, we discuss some important implementation issues and highlight the performance of our algorithm on a set of complex models. All the timings reported in this section were taken on a 2.8GHz Pentium IV PC with 2GB of memory.

Implementation

Since our generalized PD formulation is independent of the representation of the configuration space, we use a *quaternion* to represent the rotation because of its simplicity and efficiency. In our generalized PD algorithm, any proximity query package supporting collision detection and contact determination can be employed. In our current implementation, we use SWIFT++ library which is efficient and provides both proximity queries (Ehmann and Lin, 2001). By computing all the contacts between A and

B for a contact configuration $\mathbf{q_a}$, we sample the contact space locally around $\mathbf{q_a}$. For each primitive contact constraint C_i, we derive its implicit equation with respect to the parameters of a rotation component (a quaternion) and a translation component (a 3-vector). In order to sample on a constraint C_i, we first slightly perturb its rotational component by multiplying a random quaternion with a small rotational angle. The resulting rotational component is plugged back into the constraint C_i. This yields a linear constraint with only translational components, and therefore can be used to generate additional samples. To linearize C_i, we compute the Jacobian of its implicit equation for C_i . For other types of contacts, we decompose them into primitive contact constraints. Proper operators to concatenate them are identified by computing the dihedral angle of contacting edges, thereby determining whether the contact features are convex or not.

In the refinement step of the algorithm, we perform collision detection using SWIFT++ library to check whether $\mathbf{q_b}$ from the local search step still lies in the contact space. When $\mathbf{q_b}$ is on contact space, our algorithm proceeds to the next iteration. Otherwise, when $\mathbf{q_b}$ is free, a new contact configuration $\mathbf{q_b}'$ is computed for the next iteration by performing recursive bisections on the screw motion interpolating between $\mathbf{q_o}$ and $\mathbf{q_b}$. Finally, when $\mathbf{q_b}$ is in C-obstacle space, we compute a new contact configuration $\mathbf{q_b}'$ by using CCD for moving from $\mathbf{q_a}$ towards $\mathbf{q_b}$. In our current implementation, we check for collision detection on a set of discrete samples on a screw motion between $\mathbf{q_a}$ and $\mathbf{q_b}$. In order to ignore the old contact during CCD query, the idea of *security distance* is used (Redon, 2004). After computing a new contact configuration $\mathbf{q_b}'$ from the CCD query, our algorithm updates the local approximation around $\mathbf{q_a}$ and resumes a local search again.

Since our generalized PD algorithm only relies on collision detection or contact queries, the algorithm can also be extended for polygonal soup model as shown in Chapter 6.

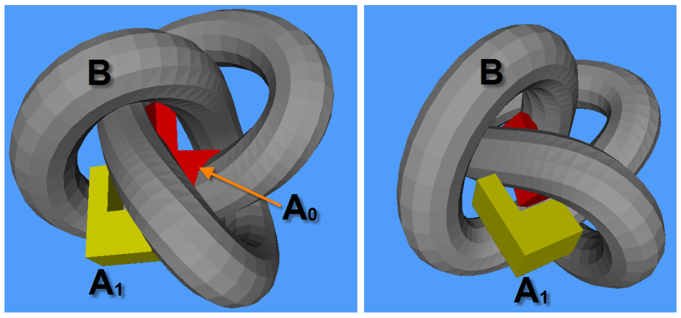


Figure 4.14: The 'torus knot' example. *The left image highlights a generalized PD query for a model 'torus knot' B intersecting with a model 'L-shaped box' at A_0 (red). A_1 is a collision-free placement of the 'L-shaped box' model as a result of* $\mathrm{PD}^g_{\mathrm{OBNO}}$ *computation; the right image shows the same result but from another viewpoint.*

Performance

We use different benchmarks to test the performance of our algorithm. Fig. 4.13(a) shows a typical setup of our experiment including two overlapping models, where A ('Pawn') is movable and B ('CAD part') is stationary. In (b), our algorithm computes $\mathrm{PD}^g_{\mathrm{DISP}}$ or $\mathrm{PD}^g_{\mathrm{OBNO}}$ to separate the model A, initially placed at A_0, from the model B. The three images on the right highlight the intermediate configurations of A_1 and A_2 and a $\mathrm{PD}^g_{\mathrm{DISP}}$ solution A_3 with yellow color. The sequence of images (b,c,d,e) illustrates that our algorithm successfully finds an upper bound of $\mathrm{PD}^g_{\mathrm{DISP}}$ by gradually sliding the 'pawn' model on the 'CAD part' model.

Figs. 4.14 and 4.15 show two more complex benchmarks that we have tested. In Fig. 4.14, the model 'torus knot' has hyperbolic surfaces. This benchmark is difficult for the convexity-based algorithm, as it is difficult to perform the convex decomposition of the complement of the complex model 'torus knot'. On the other hand, the algorithm based on constrained optimization can easily handle this benchmark, and compute an upper bound on generalized PD.

Table 4.2 summarizes the performance of our algorithm on different benchmarks. In our implementation, we set the maximum number of iterations as 30. For the most of the models we have tested, our algorithm can perform $\mathrm{PD}^g_{\mathrm{DISP}}$ query within $300ms$,

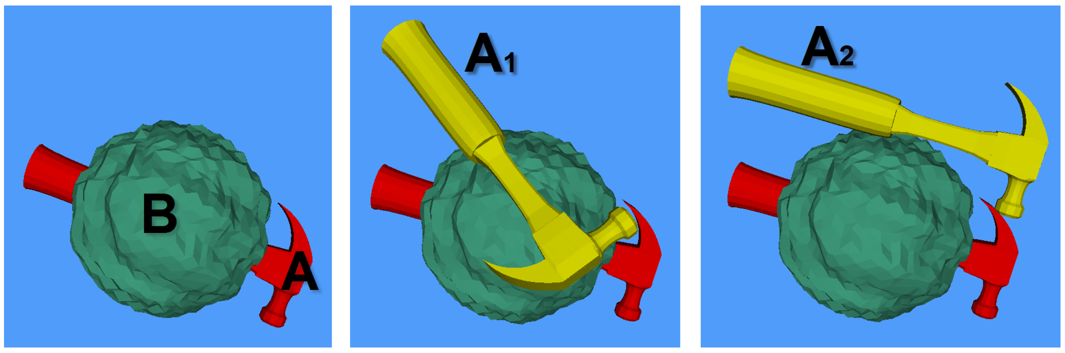


Figure 4.15: The 'hammer' example. *From left to right:* $\mathrm{PD}^g_{\mathrm{DISP}}$ *query between* A *and* B*, an intermediate configuration* A_1*, and the solution* A_2.

	1	2	3
A	L-Shape	Pawn	Hammer
tris #	20	304	1,692
B	Torus knot	CAD part	Bumpy sphere
tris #	2,880	2,442	2,880
Avg $\mathrm{PD}^g_{\mathrm{DISP}}$(ms)	219	297	109
Avg $\mathrm{PD}^g_{\mathrm{OBNO}}$(ms)	156	445	138

Table 4.2: Performance of generalized PD computation using constrained optimization. *This table highlights the geometric complexity of different benchmarks we test, as well as the performance of our algorithm.*

and $\mathrm{PD}^g_{\mathrm{OBNO}}$ query with $450ms$. Our current implementation is not optimized and the timings can be further improved.

4.3 Summary

In this chapter, we have addressed the problem of generalized PD computation between non-convex rigid models, which takes into account translational and rotational motion. We have presented two new algorithms for generalized PD computation. We summarize our algorithms and discuss their limitations.

The convexity-based algorithm makes use of the convexity of the models. Based on the convex decomposition over the input models, the algorithm can compute an lower bound and an upper bound for generalized PD. Our experimental results show that the algorithm is efficient and robust for many 3D models.

The main limitation of the convexity-based method is to perform the convex decomposition during the preprocessing, which is difficult to compute for complex objects or models without connectivity information. Given the complexity of exact generalized PD computation, our algorithm only computes lower and upper bounds. The accuracy of the bounds also depends on the convex decomposition.

The algorithm based on constrained optimization can often handle more complex non-convex models than the convexity-based algorithm. This is because we reduce generalized PD computation to proximity queries such as collision detection and contact determination. Since there are well known efficient algorithms for both queries, this algorithm is relatively easy to implement and can handle complex non-convex models.

The algorithm based on constrained optimization computes an upper bound on generalized PD, since the resulting configuration is guaranteed to be in the contact space. However, its performance depends on the choice of an initial guess and the algorithm can not guarantee a global solution. In theory, the algorithm can converge to a local minimum due to the constrained optimization formulation. One termination condition for the optimization is to check whether the gradient of the distance function has the same direction as the normal of the contact configuration at the contact space after each iteration. Issues can arise in checking this condition in practice. For example, in the case of DISP metric, one can only compute an approximation of the gradient, since no closed form is available for DISP metric. Furthermore, a convergence analysis of the algorithm is difficult, due to the discontinuity in contact space caused by multiple contacts. Our algorithm also shares some similarities with the one (Nawratil et al., 2009). Both algorithms are based on iterative optimization and can only compute a local minimum. However, in (Nawratil et al., 2009) the method and analysis assume that the given models are smooth. Instead, we focus on polyhedra and polygon soup models. Our algorithm is based on efficient collision detection and local contact space search.

Chapter 5

Complete Motion Planning

A complete motion planning approach can either compute a collision-free path if one exists, or report path non-existence otherwise. Although motion planning has been extensively studied for more than three decades, there are no practical approaches for determining path non-existence even for general low 3-4 DOF robots. Earlier exact motion planning approaches are complete. However, these approaches are known to have a high theoretical complexity and are very difficult to implement in practice. Sampling-based approaches such as probabilistic roadmap planners (PRMs) are relatively simple to implement and can be easily applied to general robots with high DOF(Kavraki et al., 1996). However, if there is no collision-free path, these approaches may not terminate. Furthermore, their performance may degrade significantly if the free space of the robot has narrow passages. In practice, when such a planner does not terminate, it is hard to distinguish whether such situation arises due to path non-existence or due to narrow passages and inadequate sampling.

Approximate cell decomposition (ACD) approaches subdivide the configuration space into simple shapes, e.g. rectangloid cells. These approaches are usually complete provided the number of subdivisions is sufficiently high. One of the main computational components in approximate cell decomposition methods is *cell labelling*, i.e. to determine whether a cell lies entirely in free or C-obstacle. Most prior labelling methods are

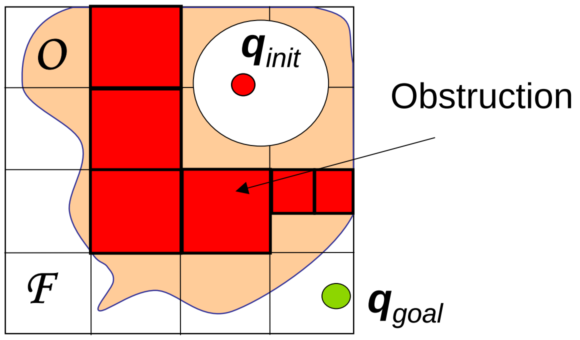


Figure 5.1: Basic idea of path non-existence computation. *To check for path non-existence, we can use C-obstacle query and determine a set of cells lying entirely in C-obstacle. If the union of these cells is 'obstructing' initial and goal configurations, we can conclude the path non-existence.*

based on contact surface computations or explicitly computing a boundary of the free space, which can be rather complicated and prone to degeneracies, especially when the robot has more than 3 DOF or the geometric models of the robot and obstacles are complex (Lozano-Pérez, 1983; Donald, 1987; Zhu and Latombe, 1990).

To be able to compute the path non-existence, the key issues are to efficiently determine whether a volumetric primitive e.g. a cell lies entirely in C-obstacle space and to decide whether the union of those identified cells is 'obstructing' initial and goal configurations (Fig. 5.1).

In this chapter, we present an efficient query algorithm to C-obstacle space, namely C-obstacle query based on generalized penetration depth computation (Zhang et al., 2008c). Using the C-obstacle query, we present a simple and efficient approximate cell decomposition algorithm for path non-existence. Our algorithm searches a path through cells which do not lie in C-obstacle. The non-existence of such a path is a sufficient condition for path non-existence between the initial and the goal configurations of the robot and these computations are performed in a hierarchical manner. Our resulting motion planning algorithm is complete for a rigid robot with translational and rotational DOF. Furthermore, our approach can also be extended to articulated robots. We have implemented our planner and highlight its performance on 3 DOF and 4 DOF robots.

We further present an efficient algorithm for complete motion planning for low DOF robots by combining the completeness of approximate cell decomposition (ACD) with the efficiency of probabilistic roadmaps (PRM) (Zhang et al., 2007b). In practice, we observe up to 10 times improvement in performance over our first complete motion planning algorithm.

We also extend our complete motion planning framework to feedback motion planning (Zhang et al., 2009). Our algorithm computes a global vector field computation in the free space as a feedback plan, which ensures that the robot at any collision-free configuration knows either the direction to move in order to reach the goal, or path non-existence to the goal. We compute a local vector field for each cell in the free space and address the issue of the smooth composition of the local vector fields between the non-uniform adjacent cells. As compared to prior approaches, our algorithm works well on non-convex robots and obstacles. We demonstrate its performance on planar robots with 2 or 3 DOF, articulated robots composed of 3 serial links and multi-robot systems with 6 DOF.

The rest of this chapter is organized as follows. We briefly survey related work on motion planning in Section 5.1. We then introduce path non-existence using approximate cell decomposition in Section 5.2. We present our efficient cell labelling algorithms for complete motion planning in Section 5.3. We present our hybrid planner in Section 5.4 and the global vector field method in Section 5.6. Finally, we discuss a few limitations of our approaches in Section 5.5. All the timings reported in this chapter were taken on a 2.8GHz Pentium IV PC with 2GB of memory.

5.1 Previous Work

Motion planning has been extensively studied for more than three decades. Excellent surveys of this topic are available in (Latombe, 1991; Choset et al., 2005; LaValle, 2006).

In terms of complete motion planning, there are no practical algorithms for determine path non-existence even for general low DOF robots. In this section, we briefly review the related work.

5.1.1 Exact Motion Planning

Some of earlier work on exact motion planning includes criticality-based algorithms such as exact free-space computation for a class of robots (Lozano-Pérez and Wesley, 1979; Kedem and Sharir, 1988; Avnaim and Boissonnat, 1989b; Halperin, 2002), roadmap methods (Canny, 1988), and exact cell decomposition methods (Schwartz and Sharir, 1983). The exact cell decomposition methods require an exact description of the boundary of the free space. The free space are partitioned into a collection of simpler geometric regions and compute a connectivity graph representing the adjacency between the regions.

In theory, these methods are complete and general. However, in practice, most of these methods are challenging to implement. Their implementations have been limited to simple planar robots, convex polytopes or special shapes such as spheres or ladders. No good implementations are known for general robots with higher than three DOF (Avnaim and Boissonnat, 1989b; Banon, 1990). Recently, a star-shaped roadmap method has been proposed (Varadhan and Manocha, 2005), which partition the free space into star-shaped regions. This method is complete as long as there are no tangential contacts in the boundary of the free space.

5.1.2 Approximate Cell Decomposition

Most practical algorithms for complete motion planning of general robots are based on approximate cell decomposition (ACD) (Brooks and Lozano-Pérez, 1985; Latombe, 1991). ACD algorithms are *complete*: provided the number of subdivisions is high. One of the main computational issues in approximate cell decomposition methods is cell labelling (Wise and Bowyer, 2000). In order to label a cell, most prior approaches rely on contact surface computations (Zhu and Latombe, 1990), which could be complicated and prone to degeneracies. Paden et al. (Paden et al., 1989) describe a labelling method based on workspace distance computation. However, due to the lack of rigorous formulation of interpenetration between two models, their method may be overly conservative in practice.

5.1.3 Sampling-based Approaches

The sampling-based approaches such as probabilistic roadmap (PRM) (Kavraki et al., 1996) and rapidly-exploring trees (RRT) (LaValle, 1998) have been widely used for different motion planning applications. These approaches are relatively simple to implement and have been successfully applied to high DOF robots. However, these approaches are only *probabilistically complete*: if a solution exists, the planner finds one in bounded time with high probability; otherwise, the planner may not terminate. Furthermore, since these approaches sample the robot's free space randomly, they may fail to find paths, especially those passing through narrow passages (Amato et al., 1998; Pisula et al., 2000; Simeon et al., 2000; Hsu et al., 2006; Zhang and Manocha, 2008b).

5.1.4 Checking for Path Non-Existence

Exact planning approaches described in Section 5.1.1 can check for path non-existence. However, most of them are not practical due to their implementation difficulty. Approximate cell decomposition approaches can also check for path non-existence. However, one main issue is efficient labelling of the cells. The sampling-based methods such as PRM or RRT are only probabilistically complete and cannot determine the path non-existence. Some effort has been made to address the issue of path non-existence in PRM planners (Basch et al., 2001), though the resulting approach is restricted to very special cases.

5.1.5 Feedback Motion Planning

Feedback motion planning generates a feedback plan over the entire free space for the robot to arrive at the goal. The feedback plan usually is represented as a vector field. One can also use potential field methods to compute navigation functions (Rimon and Koditschek, 1992), and further derive the vector fields by taking the gradients. The more direct approaches are to compute the vector field based on sequential composition (Conner et al., 2003; Belta et al., 2005; Lindemann and LaValle, 2009). By incorporating the non-holonomic and dynamic constraints, these approaches can compute vector fields for car-like robots (Lindemann and LaValle, 2007). Most of these approaches compute feedback plans with the convergence property. A quad-tree based decomposition is used for feedback planning on a simple 2D robot with 2 translational DOF and is further extended for 3D translational robots (Kloetzer and Belta, 2006). A feedback planning algorithm based on randomized sampling is presented in (Yang and LaValle, 2004).

5.2 Path Non-existence Computation using Approximate Cell Decomposition

Our algorithm to check for path non-existence is based on approximate cell decomposition (ACD). The configuration space $\mathcal{C}$ is subdivided into cells at successive levels of the subdivision. A cell C in n-dimensional C-space is defined as a Cartesian product of real intervals:

$$C = [x_1', x_1''] \times [x_2', x_2''] \cdots \times [x_n', x_n''].$$

A cell is labelled as *empty* if it lies entirely in $\mathcal{F}$, as *full* if it lies in $\mathcal{O}$, and mixed otherwise. The *mixed* cells are further subdivided (Fig. 5.2) until the algorithm computes a collision-free path or concludes the path non-existence. In Section 5.3, we present an efficient algorithm for cell labelling.

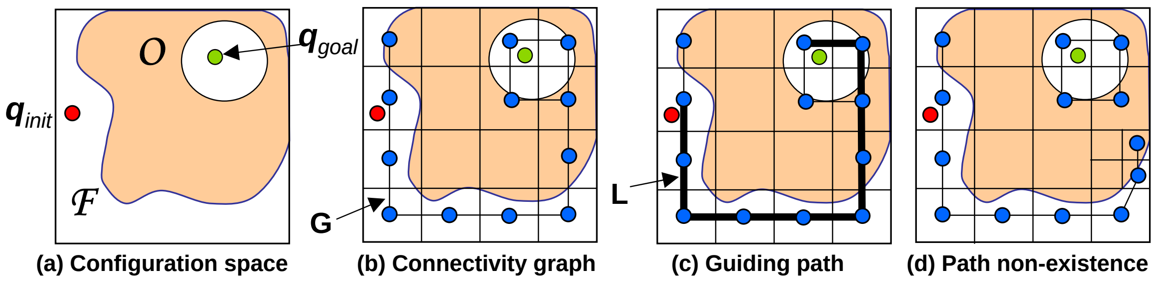


Figure 5.2: Path non-existence computation using approximate cell decomposition. *(a) Path non-existence between the configurations* $\mathbf{q}_{init}$ *and* $\mathbf{q}_{goal}$. *(b) the configuration space is decomposed into cells and a connectivity graph* G *is built for all empty and mixed cells. (c) The guiding path* L*, which connects the cells including* $\mathbf{q}_{init}$ *and* $\mathbf{q}_{goal}$*, is computed from* G*. Any mixed cell along* L *is further subdivided. For simplicity, only one of such mixed cells is highlight in (d). (d) The connectivity graph is updated. Now the cell containing* $\mathbf{q}_{init}$ *and the cell containing* $\mathbf{q}_{goal}$ *are disconnected with respect to the connectivity graph. This concludes that there is no collision-free path between* $\mathbf{q}_{init}$ *and* $\mathbf{q}_{goal}$.

5.2.1 Connectivity Graphs

For each level of subdivision, the connectivity graph G is constructed to represent the adjacency relationship among all *empty* and *mixed* cells. Formally, the connectivity graph (Latombe, 1991) associated with a decomposition of $\mathcal{C}$ is an undirected graph, where:

- The vertices in G are the *empty* and *mixed* cells.
- Two vertices in G are connected by an edge if and only if the corresponding two cells are adjacent to each other.

Intuitively, G covers the classified free space which is the union of the *empty* cells, and the 'uncertain' space which is the union of the *mixed* cells.

In order to check for path non-existence, we first find the cells C_{init} and C_{goal} that contain $\mathbf{q}_{init}$ and $\mathbf{q}_{goal}$, respectively. Next, the algorithm searches the graph G to find a guiding path L which connects the vertices corresponding to the cells C_{init} and C_{goal}. The path L corresponds to a sequence of adjacent *empty* or *mixed* cells connecting

C_{init} and C_{goal} (Fig. 5.2). Therefore, if no such path is found, it is sufficient to claim that there is no collision-free path that connects $\mathbf{q}_{init}$ and $\mathbf{q}_{goal}$, or $\mathbf{q}_{init}$ and $\mathbf{q}_{goal}$ are *disconnected.*

There are various known heuristics to prioritize the search on the connectivity graph G. We use the shortest path algorithm to search for a path in G. We further assign each edge a different weight. The edge associated with two *empty* cells has the lowest weight (0 in our implementation) and the one associated with two *mixed* cells has the highest weight. The rationale is that any two points in two adjacent empty cells can always be connected by a collision-free path. Therefore, edges associated with empty cells are favored over other types of edges.

The algorithm terminates if it can prove path non-existence using the connectivity graph G, or it can find a collision-free path using a subgraph G_f of G. G_f represents the adjacency relationship among all *empty* cells, which lie in the free space. Intuitively, G_f represents a part of the free space that has been classified till the current level of subdivision. If there is a path in G_f, a collision-free path can be easily computed by connecting the centers of the corresponding empty cells of the path. This path can also be further optimized to generate a high quality path (Zhu and Latombe, 1990).

5.2.2 Guided Subdivision

When a path L is computed after searching the connectivity graph G, it is not clear whether $\mathbf{q}_{init}$ and $\mathbf{q}_{goal}$ are disconnected. If so, we need to further explore the 'uncertain' region - the union of *mixed* cells, to acquire more connectivity information within this region. There are various ways to further explore the 'uncertain' region. A straightforward way is to apply another level of subdivision to all the *mixed* cells. However, in this way, the number of cells could increase quickly. Considering that not all 'uncertain' regions contribute to a possible disconnection from $\mathbf{q}_{init}$ to $\mathbf{q}_{goal}$, we employ the first-cut algorithm (Latombe, 1991; Zhu and Latombe, 1990) to first subdivide some of the 'uncertain' regions. Specifically, all the *mixed* cells on the path L are assigned higher priorities for the next level of the subdivision than other *mixed* cells. The algorithm is recursively applied until it finds a collision-free path or concludes path non-existence.

5.3 Cell Labelling for Path Non-existence Computation

Compared to prior approximate cell decomposition approaches, one of the distinct features is that during cell decomposition, we use reliable and efficient algorithms for cell labelling. As a result, our algorithm does not need to explicitly compute the contact surfaces. In this section, we present our *C-obstacle cell query* algorithm to check whether a cell lies entirely in C-obstacle. It is equivalent to checking whether the following predicate P_o is true:

$$P_o(A, B, Q): \ \ \forall \mathbf{q} \in Q, \ A(\mathbf{q}) \cap B \neq \emptyset. \tag{5.1}$$

Here, A is a robot, B represents obstacles and Q is a C-space primitive, e.g. a cell; $A(\mathbf{q})$ represents the placement of A at the configuration $\mathbf{q}$. Q may be a cell generated from approximation cell decomposition or any other volumetric primitive in C-space.

The collision detection algorithms can check whether a single configuration lies in $\mathcal{F}$ or $\mathcal{O}$. In contrast, *C-obstacle cell* query needs to check whether a spatial cell lies entirely in $\mathcal{F}$ or $\mathcal{O}$. This corresponds to checking the collision for all configurations within the cell. Therefore, this query is more general, and it is much harder as compared to collision detection for a single configuration.

In order to perform *C-obstacle cell query*, we place the robot at the configuration $\mathbf{q}_c$ - the center of the cell C and compute the extent of the motion or the bounding motion that the robot can undergo as it moves away from $\mathbf{q}_c$ while still being restricted within the cell C. In order to perform the We then use generalized penetration depth to measure the amount of intersection between the robot at $\mathbf{q}_c$ and the obstacle B, and compare it with the extent of the robot's bounding motion. If the amount of intersection is larger than the extent of the robot's motion, the robot placed at any configuration within the cell will collide with the obstacle. Therefore, the cell C must lie entirely in C-obstacle and it is labelled as a full cell. Our approach is conservative and provides a sufficient condition for the query. In practice, our algorithm is fast and performs the query in a few milliseconds for 2D or 3D rigid robots.

5.3.1 Motion Bound Calculation

Motion bound for a line segment

In order to formulate the motion bound for the robot when its configuration is restricted within a cell, we first introduce a case when a robot moves along a line segment in C-space. Schwarzer *et al.* (Schwarzer et al., 2005) define the motion bound λ when a robot moves along a line segment $\pi_{\mathbf{q}_a,\mathbf{q}_b}$ as an upper bound on $\mu(\mathbf{p}, \pi_{\mathbf{q}_a,\mathbf{q}_b})$ - the maximum length of the trajectory traced by any point $\mathbf{p}$ on the moving robot:

$$\lambda(A, \pi_{\mathbf{q}_a,\mathbf{q}_b}) = \text{Upper Bound}(\mu(\mathbf{p}, \pi_{\mathbf{q}_a,\mathbf{q}_b}) \mid \mathbf{p} \in A).$$

For 2D planar robots with translational and rotational DOF, the motion bound λ can be computed as a weighted sum of the difference between $\mathbf{q}_a$ and $\mathbf{q}_b$ for translational components x, y and the rotational angle ϕ:

$$\lambda(A, \pi_{\mathbf{q}_a,\mathbf{q}_b}) = \Delta_x + \Delta_y + R_\phi \Delta_\phi,$$

where Δ_x (Δ_y) is the absolute value of the difference between the x (y) components of the two configurations $\mathbf{q}_a$ and $\mathbf{q}_b$. Δ_ϕ is for the rotational component. When subtracting angles that "wrap around", we choose the direction consistent with the motion $\pi_{\mathbf{q}_a,\mathbf{q}_b}$.

The weight R_ϕ is defined as the maximum Euclidean distance between every point on A and its rotation center. In this case, we can even achieve a tighter motion bound:

$$\lambda(A, \pi_{\mathbf{q}_a,\mathbf{q}_b}) = \sqrt{\Delta_x^2 + \Delta_y^2} + R_\phi \Delta_\phi. \tag{5.2}$$

We can also extend the motion bound computation for 3D rigid objects:

$$\lambda(A, \pi_{\mathbf{q}_a,\mathbf{q}_b}) = \sqrt{\Delta_x^2 + \Delta_y^2 + \Delta_z^2} + R_\phi \Delta_\phi + R_\theta \Delta_\theta + R_\psi \Delta_\psi. \tag{5.3}$$

where ϕ, θ and ψ are the Euler angles that are used to represent the rotational motion.

Motion Bound for a Cell

Now, we define the motion bound λ of a robot when it is restricted within a cell C, instead of a line segment, as:

$$\lambda(A, C) = \max_{\mathbf{q}_b \in \partial C} \{\lambda(A, \pi_{\mathbf{q}_a,\mathbf{q}_b})\}, \tag{5.4}$$

where $\mathbf{q}_a$ is the center of C, and $\mathbf{q}_b$ is any point on ∂C or the boundary of C.

Among all line segments $\pi_{\mathbf{q}_a,\mathbf{q}_b}$, any of the main diagonal line segments of the cell has the maximum difference Δ on each component between these two configurations. According to Eqs. (5.2, 5.3), we can infer that the maximum of the motion bound $\lambda(A, \pi_{\mathbf{q}_a,\mathbf{q}_b})$ is achieved by any main diagonal line segment of the cell. Therefore, the motion bound for a cell C is equivalent to the motion bound over any main diagonal line segment $\pi_{\mathbf{q}_a,\mathbf{q}_c}$:

$$\lambda(A, C) = \lambda(A, \pi_{\mathbf{q}_a,\mathbf{q}_c}), \tag{5.5}$$

where $\mathbf{q}_a$ is the center of the cell and $\mathbf{q}_c$ is any corner vertex of the cell.

5.3.2 Generalized Penetration Depth Computation

In order to perform the C-obstacle cell query, we measure the extent of intersection between the robot and the obstacle, and compare it with a bound on the robot's motion.

If the robot only has translational DOF, we can use translational PD. However, when the robot is allowed to both translate and rotate, translational PD is not sufficient for C-obstacle cell query.

In order to deal with a robot with translational and rotational DOF, we use our notion of generalized penetration depth presented in Chapters 2 and 4. Our generalized PD takes into account both translational and rotational motion for quantifying the extent of intersection. The exact computation of generalized PD between non-convex objects is a difficult problem (Zhang et al., 2007c). In our C-obstacle cell query algorithm, we use a lower bound generalized algorithm on $\mathrm{PD}^g_{\mathrm{TRAJ}}$ that guarantees the correctness of the query. The detail on lower bound algorithm is presented in Chapter 4.

5.3.3 C-obstacle Cell Query Criterion

We now state a sufficient condition for C-obstacle cell query; i.e., checking whether A and B overlap at every configuration $\mathbf{q}$ within a cell C.

Theorem 4. (C-obstacle Cell Query) *For a cell C with a center at $\mathbf{q}_a$, the predicate $P_o(A, B, C)$ is true if:*

$$\mathrm{PD}^g_{\mathrm{TRAJ}}(A(\mathbf{q}_a), B) > \lambda(A, C).$$

Proof. Our goal is to show that Eq. (4) implies that there is no free configuration along any line segment $\pi_{\mathbf{q}_a,\mathbf{q}_b}$, where $\mathbf{q}_b$ is any configuration on the boundary of a cell C. According to the formulation of $\mathrm{PD}^g_{\mathrm{TRAJ}}$, the maximum trajectory length for every point on a robot A moving along a possible separating path should be greater than or equal to $\mathrm{PD}^g_{\mathrm{TRAJ}}(A(\mathbf{q}_a), B)$. Moreover, according to Eq. (5.4), the trajectory length of the robot when it moves along $\pi_{\mathbf{q}_a,\mathbf{q}_b}$ is less than or equal to $\lambda(A, C)$. Since $\mathrm{PD}^g_{\mathrm{TRAJ}}(A(\mathbf{q}_a), B) > \lambda(A, C)$, the minimum motion required to separate the robot A

from obstacle B is larger than the maximum motion the robot A can undergo. Therefore, there are no free configurations along any line segment $\pi_{\mathbf{q}_a,\mathbf{q}_b}$.

Since there is no free configuration along every line segment between $\mathbf{q}_a$ to any configuration on the boundary of a cell $\mathbf{q}_b$, this implies that every configuration in the cell C lies inside the C-obstacle region, and therefore, the predicate $P_o(A, B, C)$ holds. $\square$

We use Theorem 4 to conservatively decide whether a given cell C lies inside the C-obstacle space. The C-obstacle cell query algorithm consists of two steps:

1. Compute a lower bound on $\mathrm{PD}^g_{\mathrm{TRAJ}}$ for the robot $A(\mathbf{q}_a)$ and the obstacle B by Algorithm 1 presented in Section 4.1.3.

2. Compute an upper bound on motion, $\lambda(A, C)$ by Eq.'s (5.5) and (5.2).

Our C-obstacle cell query algorithm is general for both 2D and 3D rigid models. For the case where the environment consists of more than one obstacle, each obstacle is decomposed into convex pieces. Then the generalized PD algorithm can be used to compute a lower bound between the robot and each of the obstacles.

5.3.4 Extension to Articulated Robots

Our path non-existence algorithm using C-obstacle query can be extended to articulated robots. Given an articulated robot A with n links A_1, A_2, ..., A_n, we treat every link A_i as a rigid robot with translational and rotational DOF. We need to take into account the self-collision among the links of an articulated robot.

To perform C-obstacle query, we check the query criterion described in Section 5.3.3 for every link A_i of the articulated robot. We set the robot's configuration as $\mathbf{q}_a$ - the center of the query cell C. We compute a lower bound on generalized PD between a link A_i and the obstacle B. Next, we compute the motion bound $\lambda(A_i, C)$ for a link A_i when

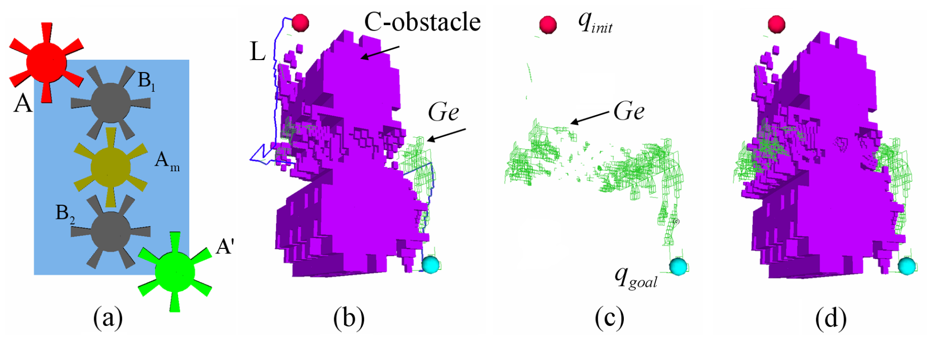


Figure 5.3: Illustration of path non-existence computation for ‘gear’ example. *(a) The goal of this example is to move a gear-shaped robot from A to A' through the two gear-shaped obstacles B_1 and B_2. It is uncertain whether there is a path for these configurations, even though the robot at A_m is collision-free. (b, c) shows the graph G_f built from empty cells, and the region of full cells (shaded volumes). Since no path is found when searching the G_f, we search the graph G for a guiding path L, which indicates the next level of subdivision. (d) After the subdivision is recursively applied, the algorithm finally concludes that no path exists. This is because the initial and the goal configuration are separated by full cells (shaded volumes in (d)).*

the robot’s configuration varies but is restricted within the cell C. Similar to a rigid robot, we can prove that this reduces to the computation of $\lambda(A_i, \pi_{\mathbf{q}_a,\mathbf{q}_c})$ - the bound of a main diagonal line segment of this cell. For an articulated robot composed by a serial of links, a motion bound on a line segment can be computed by using the Jacobian based methods described in (Paden et al., 1989; Schwarzer et al., 2005; LaValle, 2006). Finally, we compare the lower bound of $\text{PD}^g_{\text{TRAJ}}(\text{PD}^g_{\text{TRAJ}}\, A_i, B)$ with $\lambda(A_i, C)$. If the lower bound is larger, the cell C lies in C-obstacle.

If there is self-collision among the links of an articulated robot, the corresponding configuration of this robot will lie in C-obstacle. Therefore, we need to check whether a cell lies in C-obstacle caused by self-collision of the articulated robot. To do so, we check every pair of non-adjacent links A_i and A_j. We compute the lower bound on $\text{PD}^g_{\text{TRAJ}}(A_i, A_j)$. If it is larger than $\lambda(A_i, C) + \lambda(A_j, C)$, the cell C lies in C-obstacle.

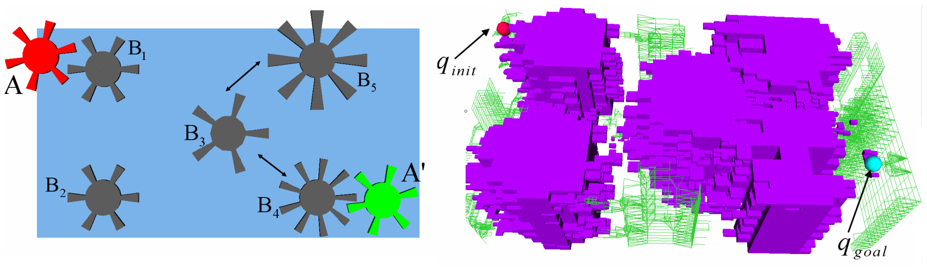


Figure 5.4: ‘Five-gear’ example for path non-existence. *(Left) The goal of this example is to move a gear-shaped robot from A to A′ passing through the 2D environment composed of five, static, gear-shaped obstacles* B_1*, ... and* B_5*. (Right) Our planner can successfully report that no collision-free path exists for this example within 6.317s. The result can be conservatively determined since the initial and the goal configuration are separated by full cells, which are highlighted by the shaded volumes in this subfigure.*

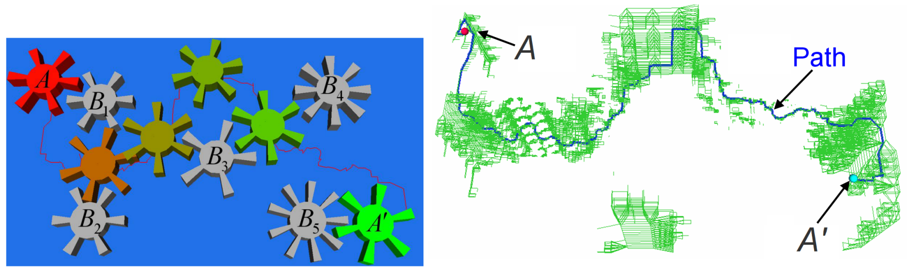


Figure 5.5: ‘Five-gear’ example with narrow passage. *(Left) this planning problem is almost same as Fig. 5.4 except that the obstacle* B_5 *is slightly modified as well as translated. Our method can find a path passing through the narrow passage in the free space. We show the robot’s intermediate configurations for the found path. (Right) The connectivity graph over the empty cells in C-space and the collision-free path are highlighted.*

Free Cell Query

We use *free cell query* to check whether a cell is empty or lies entirely in free space. We can compute the *separation distance* between the robot and the obstacle. This distance describes the ‘clearance’ between the robot and the obstacle. If this ‘clearance’ is greater than the amount of the motion bound that the robot can make, the robot will not collide with the obstacle, and the cell C will be declared as an empty cell.

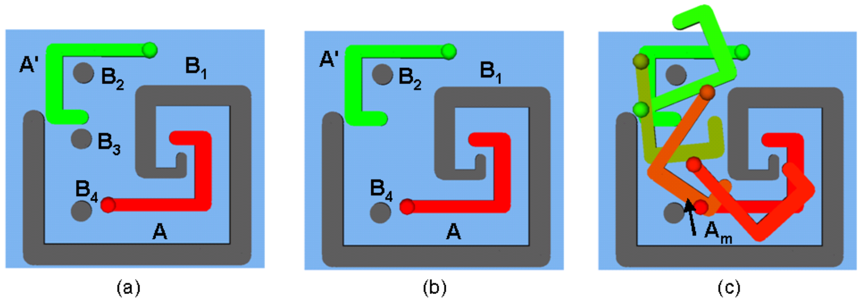


Figure 5.6: ‘2D puzzle’ example. *(a) Our algorithm can report the path non-existence for this planning problem of moving A to A' within* 7.898*s. (b) is a modified version of (a), where the obstacle B_3 is removed. Our algorithm can find a collision-free path through a narrow passage among the obstacles. (c) shows a few intermediate configurations denoted A_m of the robot along the collision-free path.*

5.3.5 Experimental Results

In this section, we describe an implementation of our algorithms for cell labelling and path non-existence computation. We highlight their performance on several benchmarks.

Our implementation of the approximate cell decomposition framework is general for any dimensional configuration space. Currently, we use the implementation for robots with three or four DOF. The main computational component for *C-obstacle query* is to compute generalized PD. We reduce $\mathrm{PD}^g_{\mathrm{TRAJ}}$ to the computation of translational PD between convex pieces, which can be implemented by using the algorithm (van den Bergen, 2001) for 3D models, and the *Minkowski sum* based algorithm for 2D models (Wein, 2008).

We illustrate our algorithm on the ‘two-gear’ example in Fig. 5.3. In order to check whether the gear-shaped robot can pass through the passage among the gear-shaped obstacles, the algorithm performs cell decomposition, and builds the connectivity graph G for *empty* and *mixed* cells as well as its subgraph G_f for the *empty* cells. The cell decomposition, which is performed in the region indicated by the guiding path from the search on the connectivity graph G, is iterated 40 times until the initial and goal configurations are found to be separated by *full* cells. The entire computation takes

3.356s.

We have applied our algorithm to some complex benchmarks including: 'five-gear', 'five-gear with narrow passage', '2D puzzle' and '2D puzzle with narrow passage'. Table 5.1 highlights the performance of our algorithm on these benchmarks. According to Table 5.1, our approach can report path non-existence for these benchmarks within 10s. In particular, for the 'five-gear' example, the total timing is 6.317s with 1.162s and 1.376s for the C-obstacle cell queries and free cell queries, respectively.

Fig. 5.7 shows a four DOF star-shaped robot. With 3 translational DOF and 1 rotational DOF, the star-shaped robot is allowed to translate freely in 3D space as well as to rotate around its local Z axis (indicated by the yellow arrow). For this planning problem, our algorithm can successfully report path non-existence with 161.720s.

Table 5.2 gives details about the performance of our algorithm on different benchmarks. For the 'five-gear' example, the cell decomposition, which is restricted in the region indicated by the guiding path, is iterated 67 times. The final cell-decomposition includes $39,068$ cells, including $3,473$ *empty* cells, $16,172$ *full* cells and $19,424$ *mixed* cells.

Fig. 5.8 shows a benchmark for a 3-DOF 2D articulated robot. The robot is composed by 3 serial links with a fixed base. The articulated robot needs to manipulate from its initial configuration to goal configuration without colliding with any of the ten rectangle-shaped obstacles in the 2D plane. For this problem, our planner can determine path non-existence within $1,513.682$s. The breakdown of the timing and other profiling information of this benchmark are summarized in Tables 5.1 and 5.2.

Since our algorithm performs cell decomposition, it is directly applicable to computing collision-free paths even when the free space has narrow passages. Finding a collision-free path through a narrow passage has been considered as a difficult task for randomized sampling methods, such as PRM. Fig. 5.5 shows the modified 'five-gear' example. Our planner can find a path through the narrow passage within 85.163s.

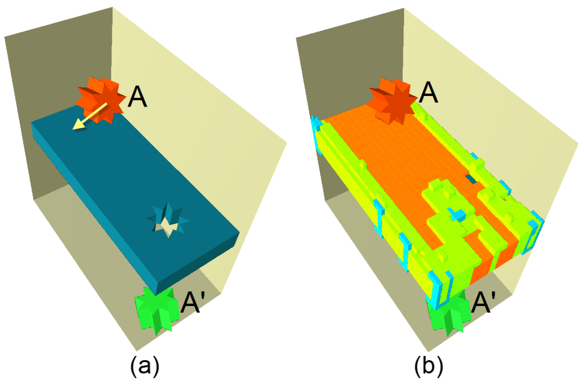


Figure 5.7: 4-DOF 'star' example. *The star-shaped robot is allowed to translate freely in 3D space as well as to rotate around its local Z axis (indicated by the yellow arrow). (a) the goal of this example is to move the star-shaped robot from A to A' by passing through the star-shaped hole of the rectangular bar model. Our algorithm can successfully report path non-existence for this planning example with 161.720s. (b) shows the set of C-obstacle cells, which separate the robot from its initial configuration to goal configuration. We project the configuration space* $R^3 \times SO(1)$ *into* R^3*, and use different colors indicate the different levels of subdivision when cells are generated.*

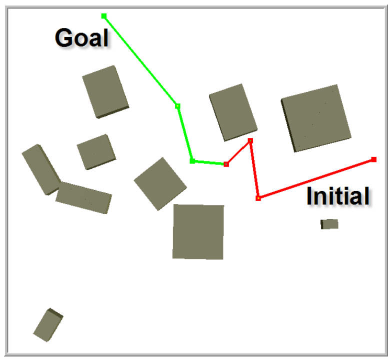


Figure 5.8: A 3-DOF articulated robot. *Our planner determines path non-existence from the robot's initial configuration to goal configuration within* $1,513.682s$.

	two -gear	five -gear	five-gear narrow	puzzle	puzzle narrow	star 4-DOF	serial links
Total timing(s)	3.356	6.317	85.163	7.898	15.751	161.720	1,513.682
Free cell query(s)	0.858	1.376	6.532	2.174	2.993	21.050	42.591
C-obstacle cell query(s)	0.827	1.162	4.675	2.021	2.612	24.492	209.660
G searching(s)	0.389	1.409	30.687	1.991	5.685	61.802	820.976
G_f searching(s)	0.077	0.332	7.169	0.309	1.035	17.643	164.939
Subdivision,Overhead(s)	1.205	2.038	36.100	1.403	3.426	36.733	275.516

Table 5.1: Performance. *This table highlights the performance of our complete motion planning algorithm on different benchmarks.*

	two -gear	five -gear	five-gear narrow	puzzle	puzzle narrow	star 4 DOF	serial links
# of iterations	41	67	237	66	107	294	417
# of free cell queries	32,329	44,649	192,009	59,121	77,297	207,713	2,133,241
# of C-obstacle cell queries	30,069	41,177	176,685	55,683	70,438	174,131	1,962,493
# of cells	28,288	39,068	168,008	51,731	67,635	194,731	1,866,586
# of empty cells	2,260	3,472	15,324	3,438	6,859	33,582	170,748
# of full cells	12,255	16,172	74,713	26,295	3,0351	73,928	670,928
# of mixed cells	13,773	19,424	77,971	21,998	30,425	87,221	1,024,910

Table 5.2: The table summarizes various statistical information of our complete planner.

	Five-gear narrow
Cell Culling Ratio	75.21%
Time Per Cell Culling(ms)	0.12
Time of Original Method(s)	261.4
Time of Accelerated Method(s)	110.4
Speedup	2.4
Time for C-obstacle Cell Query(s)	13.3

Table 5.3: Performance for C-obstacle cell query. *For the Gear example, our query can identify about 75.21% of C-obstacle cells. The average query time is about 0.12ms. Based on C-obstacle query, we improve the performance of the star-shaped roadmap planner by* 2.4 *times in this case.*

Effectiveness of C-obstacle Cell Query

To demonstrate the effectiveness of our C-obstacle cell query, we apply our *C-obstacle query* algorithm to speedup a complete motion planning algorithm - the star-shaped roadmap method (Varadhan and Manocha, 2005) (cf Fig.5.4). We define the *cell culling ratio* as the number of cells in C-obstacle space identified by our query algorithm over the total number of cells in C-obstacle space.

Tab. 5.3 illustrates that our *C-obstacle query* algorithm can achieve 75.21% cell culling ratio in our *Gear* benchmark. Tab. 5.3 also shows that the average time for each *C-obstacle query* in the *Gear* example is about 0.12ms. In this complex 2D scenario, the *C-obstacle query* algorithm improves the performance of the star-shaped roadmap planner by by 2.4 times.

5.4 A Hybrid Approach for Complete Motion Planning

The approach presented in the previous section based on approximate cell decomposition (ACD) and cell labelling is *complete*: it can either find a collision-free path or conclude that no such path exists provided the number of subdivisions is high or small resolution parameters are chosen. However, the algorithm can generate a large number of mixed cells. Moreover, the complexity of the subdivision algorithm increases exponentially with the dimension of $\mathcal{C}$.

On the other side, the practical motion planning algorithms for high-DOF robots are based on sampling-based approaches, including the probabilistic roadmap (PRM) method and its variants. Because of their simplicity and efficiency, these algorithms have been successfully used to solve many high-DOF motion planning problems. However, these algorithms may not terminate when no collision-free path exists in the free space. Their performance can degrade when the configuration space has narrow passages.

In this section, we present an efficient algorithm for complete motion planning that combines approximate cell decomposition (ACD) with probabilistic roadmaps (PRM) (Zhang et al., 2007b). Our approach uses ACD to subdivide the configuration space into cells and computes localized roadmaps by generating samples within these cells. We augment the connectivity graph for adjacent cells in ACD with pseudo-free edges that are computed based on localized roadmaps. These roadmaps are used to capture the connectivity of free space and guide the adaptive subdivision algorithm. At the same time, we use cell decomposition to check for path non-existence and generate samples in narrow passages. Overall, our hybrid algorithm combines the efficiency of PRM methods with the completeness of ACD-based algorithms. We have implemented our algorithm on 3-DOF and 4-DOF robots. We demonstrate its performance on planning scenarios with narrow passages or no collision-free paths. In practice, we observe up to 10 times

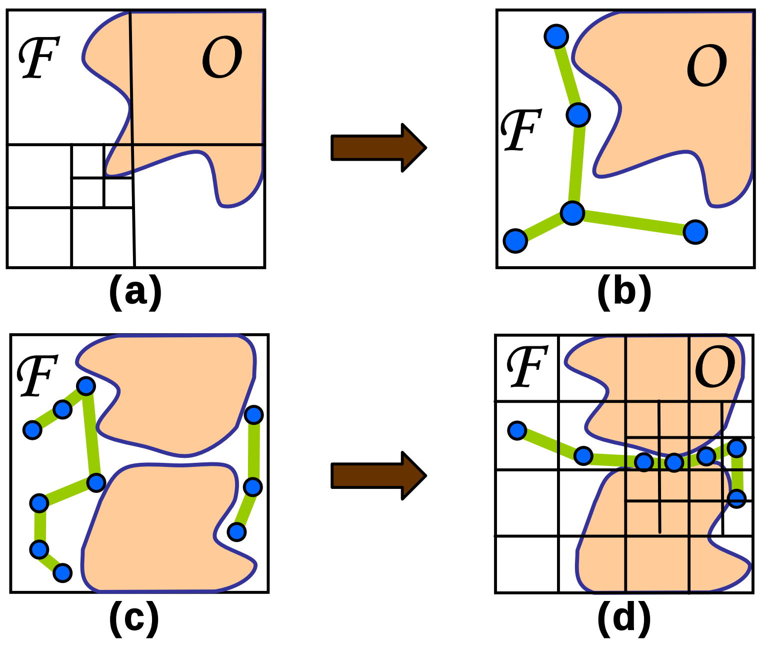


Figure 5.9: Benefits of the hybrid planner. *This example highlights that our hybrid algorithm can combine both benefits of ACD and PRM. First row: (a) In ACD, to capture the connectivity of the free space within this mixed cell, many subdivisions are required; (b) A localized roadmap within this cell can well capture its connectivity by only a few samples, and thereby can improve the overall performance of the planning algorithm of ACD. Second row: (c) It is difficult for PRM methods to sample in the narrow passage; (d) The structure of the cell decomposition can be used to generate more samples in the narrow passage.*

improvement in performance over our first complete motion planning algorithms.

The rest of the section is organized as follows. We first give an overview of our hybrid approach and introduce the key data structures. We then give the details of localized roadmap computation and subdivision algorithms. We describe our implementation of the hybrid planner and highlight its performance on many examples. Finally, we discuss the limitations of our method and compare its performance with prior approaches.

5.4.1 Overview

In this section, we give a broad overview of our hybrid planning algorithm. We also introduce the key data structures used in our algorithm.

At a broad level, our algorithm performs adaptive decomposition of $\mathcal{C}$ into rectangular cells similarly to the previous ACD methods, and uses efficient labelling algorithms presented in the previous section to classify them as empty, full or mixed cells. The main bottleneck in previous ACD methods lies in dealing with a large number of cells. Most of the cells are classified as mixed cells, and they are recursively subdivided till their size is less than a threshold. This is due to three reasons. First, the exact boundary of the free space is complex and not aligned with the cell boundaries (Fig. 5.9). Therefore, many levels of subdivisions are needed to compute a good approximation of the free space. Secondly, most cell labelling algorithms tend to be conservative, i.e. some of the empty or full cells are classified as mixed. Finally, the complexity of the subdivision algorithm increases as an exponential function of the number of DOF.

In order to address these problems, we augment the cells with localized roadmaps, which tend to capture the connectivity of the free space within each mixed cell. Furthermore, we attempt to connect the localized roadmaps of adjacent cells using pseudo-free edges. Within each mixed cell, the roadmap provides a compact representation of its connectivity, while a pseudo-free edge captures the connectivity of the localized roadmaps between two adjacent cells. As a result, there is a high probability that we can compute a path through these mixed cells and assign them a lower priority in terms of adaptive subdivision. Overall, our hybrid algorithm performs fewer subdivisions compared to prior ACD algorithms.

Our hybrid method also improves the performance of PRM algorithms. Since we only generate random samples in the mixed cells at any level in the subdivision, our approach automatically computes more samples near or in narrow passages. Compared to prior PRM approaches, this results in an improved sampling strategy. Moreover, by using ACD for path non-existence queries, our hybrid algorithm is complete.

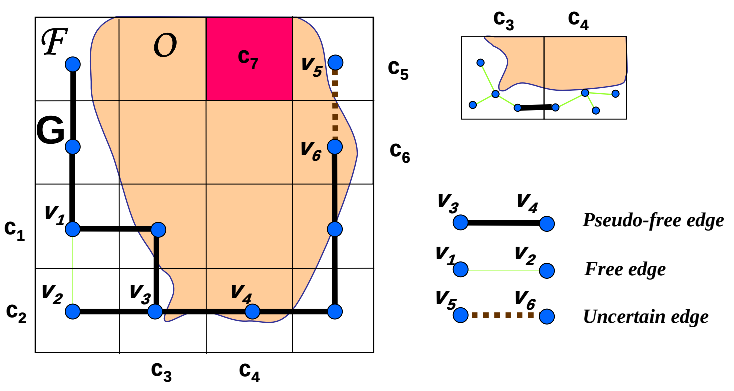


Figure 5.10: Pseudo-free edges and connectivity graph. *ACD subdivides the C-space, and classifies the resulting cells as empty, such as c_1, full such as c_7 or mixed such as c_3. The connectivity graph G is a dual graph to ACD and each empty or mixed cell is mapped to a vertex in G. There are three types of edges in our connectivity graph G. Two adjacent empty cells, such as c_1 and c_2 are connected by a free edge (v_1, v_2). Two non-full cells are connected by a pseudo-free edge such as (v_3, v_4) if their localized roadmaps can be connected as the right figure shows; otherwise, they are connected by an uncertain-edge such as (v_5, v_6).*

Localized Roadmaps

Let us denote the approximate cell decomposition of configuration space as $\mathcal{P}$, and use c_i to represent each cell in $\mathcal{P}$. In our approach, a small fraction of mixed cells are associated with localized roadmaps. For each empty cell, a trivial roadmap with only a single sample in its center is constructed. We also implicitly maintain a global roadmap $\mathcal{M}$ for $\mathcal{P}$, including all the localized roadmaps $\mathcal{M}_c$ associate with each cell c; i.e., $\mathcal{M} \supset \cup\mathcal{M}_c$ where $\mathcal{M}_c \neq \phi$. In addition, for two adjacent cells c_i and c_j, if there is a collision-free path to connect their associated localized roadmaps $\mathcal{M}_{c_i}$ and $\mathcal{M}_{c_j}$, this path is added to $\mathcal{M}$ (Fig. 5.10). Details of this computation are given in Section 5.4.2.

Connectivity Graph

As a dual graph of $\mathcal{P}$, the connectivity graph G represents the connectivity between the cells in $\mathcal{P}$. The graph is defined as follows: each non-full (empty or mixed) cell in $\mathcal{P}$ is mapped to a vertex v in G; if two non-full cells c_i and c_j in $\mathcal{P}$ are adjacent to each other, their corresponding vertices, v_i and v_j, respectively, are connected by an edge

$e(i,j)$ in G. Furthermore, an edge $e(i,j)$ is classified into one of the following three types (Fig.5.10):

- **Free:** If c_i and c_j are both empty, $e(i,j)$ is a *free edge*. This implies that there exits a collision-free path between any configuration $\mathbf{q_0}$ in c_i to any configuration $\mathbf{q_1}$ in c_j.

- **Pseudo-free:** If $e(i,j)$ is not a free edge, but two localized roadmaps M_{c_i} and M_{c_j} associated with c_i and c_j can be connected by a collision-free path, $e(i,j)$ is called a *pseudo-free edge*. The existence of a pseudo-free edge can be checked by any local planner. Its existence indicates that it is *highly likely* that there exists a collision-free path between any free configuration $\mathbf{q_0}$ in c_i and free configuration $\mathbf{q_1}$ in c_j.

- **Uncertain-edge:** If $e(i,j)$ is neither free nor pseudo-free, it is classified as an *uncertain-edge*. Since the localized roadmaps M_{c_i} and M_{c_j}can not be connected by local planning, it is unlikely that there exist a collision-free path between any free configuration $\mathbf{q_0}$ in c_i and any free configuration $\mathbf{q_1}$ in c_j.

We further define some of the subgraphs of G as follows: the *free connectivity graph* G_f is a subgraph of G that only includes all free edges of G. The pseudo-free connectivity graph G_{sf} is a subgraph of G that includes both all the free edges and the pseudo-free edges. The three types of connectivity graphs represent different levels of approximations of the free space $\mathcal{F}$ and are used by the path planning algorithm. More specifically,

- $\mathbf{G}$ represents the adjacency among free or mixed cells, which form a superset of the free space $\mathcal{F}$. Therefore, the graph is useful for deciding path non-existence, because no path found in G implies that there is no collision-free path in $\mathcal{F}$.

- $\mathbf{G}_f$ represents the adjacency among all free cells, which forms a conservative approximation or a subset of $\mathcal{F}$. It is useful for finding a collision-free path for

A.

- $\mathbf{G}_{sf}$ represents the adjacency among all free cells and a portion of mixed cells. They represent a good approximation of the free space for path queries, since a free edge (or a pseudo-free edge) among two adjacent cells indicates there must be (is likely) a collision-free path between any pair of free configurations in the two cells. We compute localized roadmaps to capture the connectivity for this approximation, and use them for path queries.

5.4.2 Hybrid Planning Algorithm

In this section, we describe our hybrid motion planning algorithm in detail, with an emphasis on computation and use of data structures introduced in the previous section. Fig. 5.11 shows a flowchart of our algorithm, which consists of two stages: *finding a collision-free path* and *checking for path non-existence.* These two stages are executed iteratively until a collision-free path is found or the path non-existence is detected.

Starting with an initial, coarse and uniform approximate cell decomposition $\mathcal{P}$ of $\mathcal{C}$, our algorithm proceeds in the following manner.

Stage I. Collision-free Path Computation

1. Locate the cells in $\mathcal{P}$ that contain $\mathbf{q}_{init}$ and $\mathbf{q}_{goal}$; denote their corresponding vertices in G as v_{init} and v_{goal}, respectively.

2. Search G_f to find a path that connects v_{init} and v_{goal}. If a path L_f is found, a collision-free path for the given planning problem can be computed by connecting the initial configuration, the path L_f and the goal configuration, since the space represented by G_f is a conservative approximation of $\mathcal{F}$.

3. If no path is found in G_f, we search the graph G_{sf} for a path to connect v_{init} and v_{goal}. If no path is found in G_{sf}, this means that there is no collision-free

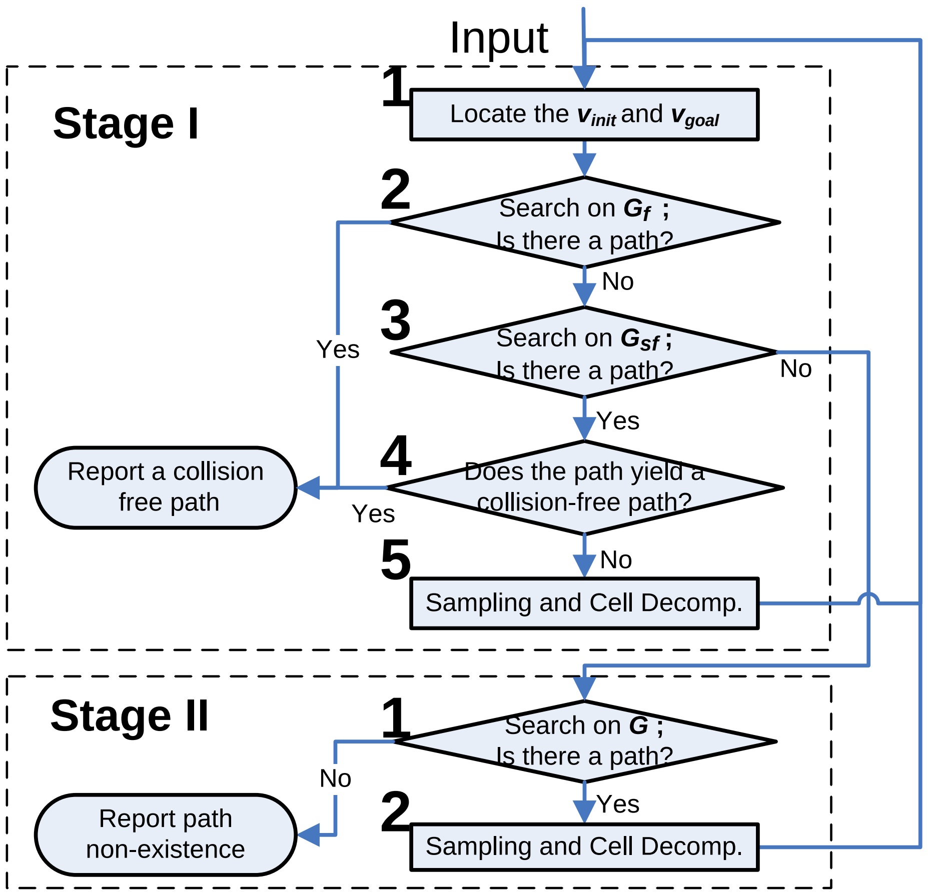

Input
Stage I
1
Locate the v_{init} and v_{goal}
2
Search on G_f ; Is there a path?
No
Yes
3
Search on G_{sf} ; Is there a path?
No
Yes
4
Does the path yield a collision-free path?
Report a collision free path
Yes
5
No
Sampling and Cell Decomp.
Stage II
1
Search on G ; Is there a path?
No
Report path non-existence
2
Yes
Sampling and Cell Decomp.

Figure 5.11: Flowchart of the hybrid planner. *Our algorithm consists of two stages. The algorithm is executed iteratively until a collision-free path is found in stage I, or the path non-existence is detected in stage II.*

path within the current approximation of $\mathcal{F}$ represented by G_{sf}. Therefore, our algorithm proceeds to checking for path non-existence stage.

4. If a path, say L_{sf}, is found in G_{sf}, it suggests that a collision-free path may exist. In order to verify the existence, we search over the union of all localized roadmaps associated with the cells along the path L_{sf}. If a collision-free path is found, our algorithm terminates. More specifically, let $P_{L_{sf}}$ be a sequence of cells in $\mathcal{P}$ corresponding to the vertices in L_{sf}. Let $\mathcal{M}_{L_{sf}}$ be a subgraph of $\mathcal{M}$ that lies within $P_{L_{sf}}$. To verify whether L_{sf} can yield a collision-free path, we first search over $\mathcal{M}_{L_{sf}}$. If no path is found in $\mathcal{M}_{L_{sf}}$, then we search the entire roadmap $\mathcal{M}$.

If no collision-free path is found within $\mathcal{M}$, this implies that the current PRM representation is not fine enough to compute a collision-free path. Therefore, we need a more accurate (or finer) representation of $\mathcal{F}$. For this case, the algorithm proceeds to the next step.

5. If no path can be computed, we identify *critical cells* along the path L_{sf}, which break the reachability between $\mathbf{q}_{init}$ to $\mathbf{q}_{goal}$. Additional samples are generated in the critical cells to improve their localized roadmaps. After that, we perform one level of subdivision on these cells and update the graphs G, G_f and G_{sf}. Next, the algorithm returns to the Path Computation Stage.

Stage II. Checking for Path Non-Existence

1. We perform a graph search on G to find a path connecting v_{init} and v_{goal}. If no path can be found in G, our algorithm can safely conclude that the given planning query has no solution, since the space represented by the graph G is a superset of $\mathcal{F}$.

2. Otherwise, we compute a path L in G to connect v_{init} and v_{goal}, and perform sampling and cell subdivisions on the *critical cells* along L. The algorithm then updates the connectivity graphs and returns to the Path Finding Stage.

Improved Sampling and Cell Subdivision

If the stage of path computation is not able to find a collision-free path in G_f or G_{sf}, we generate additional samples for $\mathcal{M}$ and subdivide the cells in $\mathcal{P}$ (i.e., step 5 of Stage I). A straightforward scheme would generate additional samples for all mixed cells in $P_{L_{sf}}$ and further subdivide them. In order to perform this step more efficiently, we identify the *critical* cells and only generate additional samples and perform cell subdivision on these cells. More specifically, the critical cells are defined as those cells, where the roadmap $\mathcal{M}_{L_{sf}}$ is disconnected with respect to $\mathbf{q}_{init}$ to $\mathbf{q}_{goal}$.

Overall, the use of critical cells results in adaptive sampling and fewer subdivisions. First of all, there may exist cells in C-obstacle that actually separate a part of free space. These types of cells are useful in terms of checking for path non-existence. Therefore, we can concentrate on classifying these cells by performing additional sampling and subdivisions. Moreover, poor sampling in one of these cells can result in a disconnected localized roadmap, and therefore, these cells are good candidates to receive additional samples.

Critical cell computation: In order to identify the critical cells in the set $\mathcal{P}_{L_{sf}}$, we use a propagation algorithm based on depth first search (DFS). The time complexity of this algorithm is linear to the size of $\mathcal{M}_{L_{sf}}$. As Fig. 5.12 shows, we denote the cells along the path L_{sf} as c_1, c_2, ..., c_n (n=5), and their corresponding vertices in L_{sf} are v_{init}, v_2, ..., v_{n-1}, v_{goal}. The algorithm searches $\mathcal{M}_{L_{sf}}$ from $\mathbf{q} = \mathbf{q}_{init}$ using DFS, and set the *reachable* flags of its descent samples as true (initially, the flag for every sample is false). When DFS stops, we check whether the reachable flag of $\mathbf{q}_{goal}$ has been set. If not, the algorithm searches for a cell c_i, which contains at least one reachable sample and has the largest index i. The cell c_i is a critical cell, since the roadmap $\mathcal{M}_{L_{sf}}$ is disconnected in this cell w.r.t. $\mathbf{q_{init}}$ and $\mathbf{q_{goal}}$. If v_i is not equal to v_{goal}, we iterate this process to find more critical cells. Since L_{sf} is computed from G_{sf}, v_i should have a pseudo-free edge with its adjacent vertex v_{i+1} in L_{sf}. Furthermore, this pseudo-free edge is realized by a local path between a sample $\mathbf{q}_m$ in c_i with a sample $\mathbf{q}_n$ in c_{i+1}. Therefore, we can resume DFS search from $\mathbf{q} = \mathbf{q}_m$. This process continues until v_i is equal to v_{goal}.

Critical cell computation for path non-existence: In the stage corresponding to path non-existence computation, if a path L is found in G, we need to refine our representation of $\mathcal{F}$. For this purpose, one known technique is *first graph cut* (Latombe, 1991), which only subdivides the cells along the L, instead of mixed cells in $\mathcal{P}$. In our algorithm, we further reduce the number of subdivisions by identifying the critical cells along L. More specifically, a cell c along L is critical if there exists more than

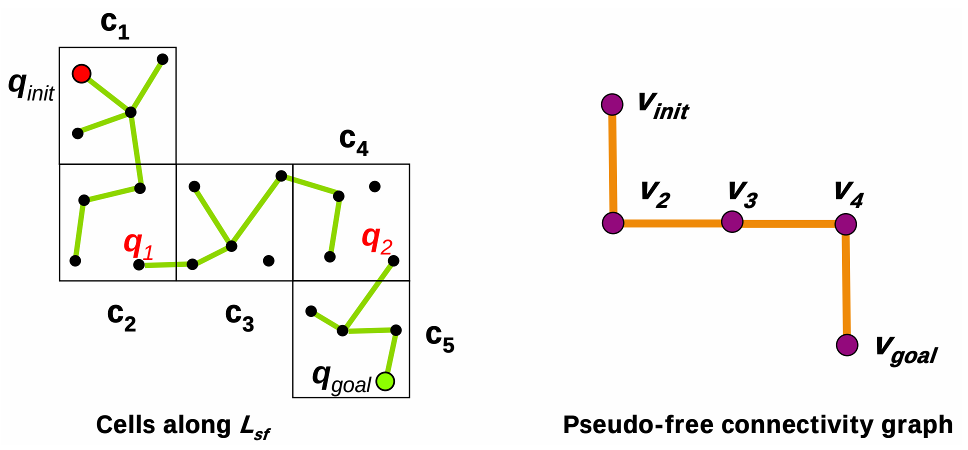


Figure 5.12: Critical cell computation. *The cells c_2 and c_4 are classified as critical cells, since there the roadmap $\mathcal{M}$ is disconnected. We identify such cells using a propagation algorithm based on DFS. Note that since the roadmaps in c_2 and c_3 are connected by an edge, their corresponding vertices v_2 and v_3 are connected by a pseudo-free edge too.*

one connected graph component in $\mathcal{M}_c$; two adjacent cells on the path L are critical, if there is no free edge or pseudo-free between them. Only these critical cells are further subdivided and extra samples are generated to update their localized roadmaps.

5.4.3 Implementation and Performance

We have implemented our hybrid planner and tested its performance on 3-DOF and 4-DOF robots in difficult motion planning scenarios. In this section, we address some implementation issues. We analyze the performance of our planner, and compare it with other complete motion planning algorithms.

Implementation

We perform approximate cell decomposition on the configuration space $\mathcal{C}$, and use C-obstacle and Free-cell query algorithms to label the cells during subdivision. Our formulation of the adaptive subdivision framework is general for arbitrary dimensional $\mathcal{C}$, and we have tested it on 3 and 4 dimensional $\mathcal{C}$.

The two main computational components in our algorithm are graph search and

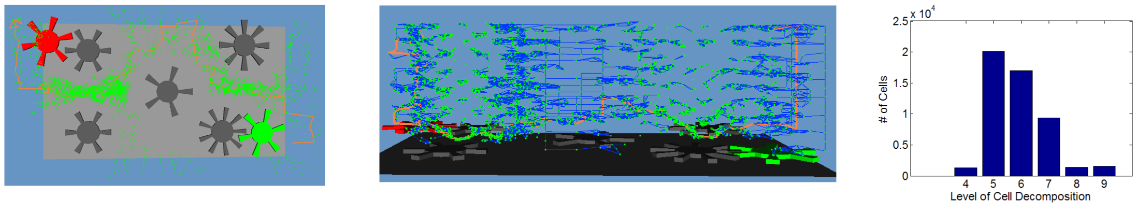


Figure 5.13: ‘Five-gear’ example with narrow passage. *The left figure shows a 3-DOF planning problem with narrow passages. Shown in the left and middle figures, where 3 dimensional C-space is illustrated together with the workspace, our hybrid planner can generate samples in the narrow passage, and the global roadmap constructed can capture the connectivity in the free space well. The middle figure also highlights the roadmap for the free space. The figure in the right shows the histogram of the number of cells in different levels of subdivisions.*

localized roadmap computation. In order to search for a shortest path in the connectivity graph G, we assign different weights to different types of edges. The underlying idea is to assign a higher weight (i.e. a lower priority) to the uncertain edges, so that the search algorithm tends to find a path through the free edges and pseudo-free edges. This results in a path with fewer uncertain edges and results in fewer subdivisions. In our current implementation, the weight of a free edge is set as zero and the weight of a pseudo-free edge is also set as zero. The weight of an uncertain edge $e(i, j)$ is set as the distance between the centers of cells c_i and c_j.

For the localized roadmap computation, more samples are generated for mixed cells than free cells. In our experiments, the maximum number of free samples in each mixed cell, N_m, is set as 5. The maximum trial number of random samples used to generate each free sample, N_{trial}, is 5. For each free cell, we only need to generate a sample at its center.

Results

We have tested our hybrid planner on different examples. Our current implementation is not optimized. We also compare our algorithm with the ACD-based complete planner presented in Section 5.3. The performance and various statistics are summarized in

(sec)	Five-gear	Star	Star(no-path)	Notch
Total timing	33.855	16.197	48.453	102.076
Cell labelling	4.025	9.562	31.793	20.915
Sampling	5.313	0.265	1.096	5.147
Link computation	8.829	4.172	14.345	27.623
G_f, G_{sf} search	1.123	0.462	2.037	3.185
G search	5.472	1.218	6.139	13.574
Subdivision	9.093	0.518	6.130	31.632

Table 5.4: Performance of the hybrid planner. *This table highlights the performance of our algorithm on different examples. We show the breakup of timings among different parts of the algorithm. The five-gear is a 3-DOF example and the rest are 4-DOF examples.*

Tables 5.4 and 5.5.

3-DOF 'five-gear' example with narrow passages This is a difficult 3-DOF motion planning problem. There are narrow passages for this example, and the boundary of C-space for this example is very complex. Our hybrid planner can compute a collision-free path within $33.855s$, which is about three times faster than previous method. The number of cells in the approximate cell decomposition is $50,730$, which is only 30.2% of the number in the previous ACD method. Fig. 5.13 highlights that our approach can generate the samples and construct the probabilistic roadmap effectively near or in narrow passages. The roadmap $\mathcal{M}$ for this example includes $6,488$ samples and $15,298$ edges. Each sample in $\mathcal{M}$ has only 4.7 neighbors on average. This can be observed in Fig. 5.13, where each sample is connected with a few other samples.

Table 5.5 demonstrates that only a subset of mixed cells in ACD are associated with localized roadmaps. This confirms that our approach is able to generate and utilize the samples effectively.

4-DOF 'star' example Figs. 5.14 and 5.15 show a 4-DOF robot, with 3 translational DOF and 1 rotational DOF. The star-shaped robot is allowed to translate freely in 3D space and to rotate around its local Z axis (indicated by the yellow arrow) in its local

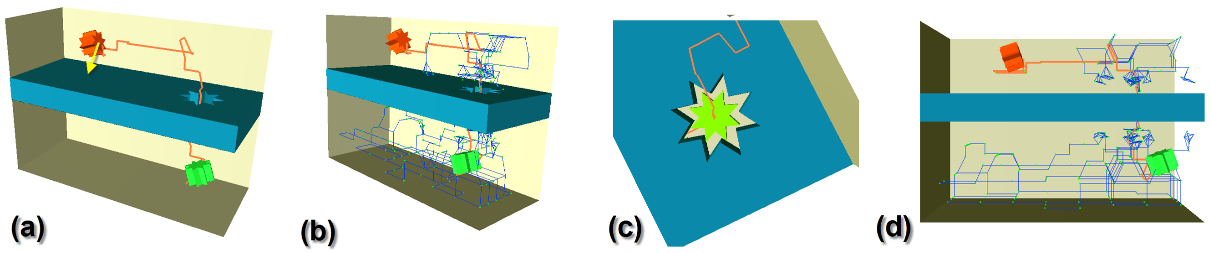


Figure 5.14: 4-DOF ‘star’ example with narrow passage. *The star-shaped robot is allowed to translate freely in 3D space and to rotate around its local Z axis (indicated by the yellow arrow). (a) This planning problem is to move the robot from the red placement (top) to the green placement (bottom) by passing through the star-shaped narrow hole. Our approach can find a collision-free path within* 16.197*s. For the purpose of the visualization, we project the configuration space from* $R^3 \times SO(1)$ *into* R^3*. (a, c) shows the path and the robot's intermediate configurations on the path. (b,d) shows the roadmap from two different viewpoints.*

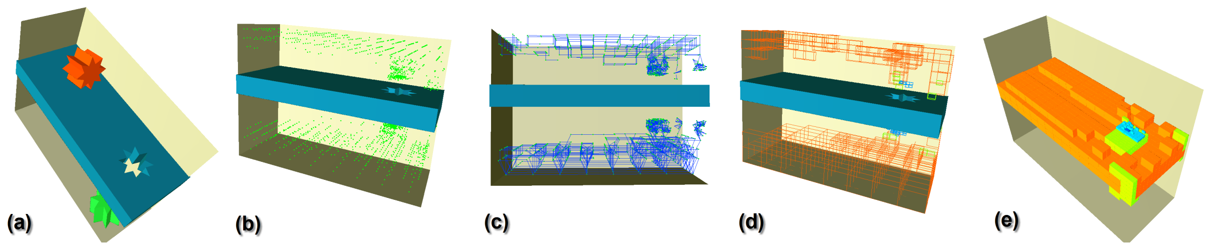


Figure 5.15: 4-DOF ‘star’ example for path non-existence. *We modify the scene in Fig. 5.14 by scaling the robot by* 1.3*. Our planner can report path non-existence for this new example within* 48.453*s. (b, c) shows the samples and the roadmap generated by our approach. (d) shows the subset of mixed cells in ACD, which are associated with localized roadmaps. (e) shows the set of C-obstacle regions, which separate the robot from its initial configuration to goal configuration.*

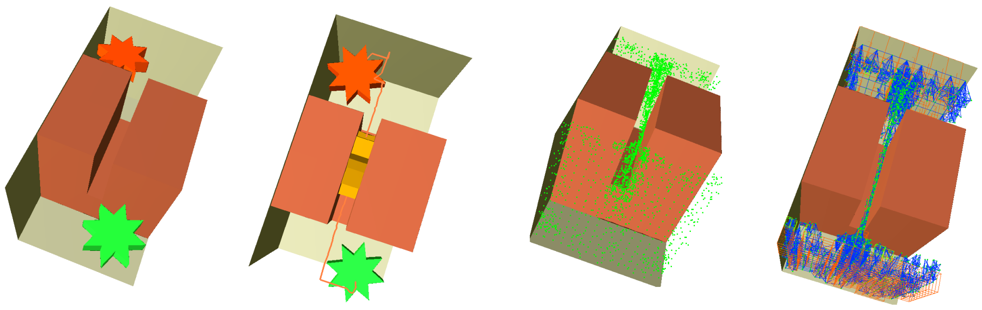


Figure 5.16: 4-DOF ‘notch’ example. *The star-shaped robot needs to pass through the very narrow notch. Our approach can find a collision-free path within* 102.076*s.*

	Five-gear	Star	Star no path	Notch
# of cells	50,730	48,046	82,171	164,446
# of empty cells	1,272	12,159	15,651	7040
# of full cells	20,761	10,063	31,984	108,983
# of mixed cells	28,697	25,824	34,536	48,423
# of samples in $\mathcal{M}$	6,488	465	2,791	5,494
# of edges in $\mathcal{M}$	15,298	732	5,040	12,707
Avg degree of sample	4.72	3.15	3.61	4.63
# of mixed cells associated with $\mathcal{M}$	2,457	69	353	1,584
# of free cells associated with $\mathcal{M}$	568	335	2,078	2,804
Peak memory usage (MB)	67	51	75	130

Table 5.5: This table gives different statistics related to the examples.

	Hybrid Planner	ACD-based Planner	Speedup
Total timing	33.855(s)	85.163(s)	2.52
Total cells	50,730	168,008	3.31

Table 5.6: Comparison. *We achieve up to 3 times speedup over prior ACD method for the 'five-gear' example. For the 4-DOF 'star' example, the ACD version presented in the previous section could not terminate within 10 mins. But the hybrid planner can report the correct result for both scenarios less than 1 min.*

coordinate system. We test this example for two scenarios: to find a collision free path for the original star-shaped robot, and to detect path non-existence when the robot is uniformly scaled by 1.3. The performance and various statistics for this example are summarized in Tables 5.4 and 5.5.

4-DOF 'notch' example Fig. 5.16 shows a 4-DOF example, where the star-shaped robot needs to pass through a very narrow passage, the notch in this figure. Our approach can find a collision-free path for this example within 166.464s, and only generates $5,494$ samples.

5.5 Analysis and Comparison

In this section, we show that both our algorithms based on approximate cell decomposition (ACD) are complete. In our first ACD-based algorithm, we use our C-obstacle query algorithm to label the full cells, which lie entirely in C-obstacle space. Our second hybrid algorithm further improves the overall performance by combining the efficiency of randomized sampling and the completeness of ACD. We also compare our algorithms to other exact or hybrid planners.

5.5.1 Completeness

Both of our algorithms are based on ACD. ACD approaches are traditionally classified as *resolution-complete.* If there are no tangential contacts in the free space, these approaches can either find a collision-free path or report that no such path exists provided small resolution parameters are chosen for the subdivision (Latombe, 1991). We can also classify ACD approaches as *complete* since provided the planners can keep subdividing C-space or the level of subdivisions is sufficiently high, the planners are guaranteed to compute a collision-free path or correctly report path non-existence in finite time.

The key component in our ACD-based algorithms to check for path non-existence is the *C-obstacle cell query*, which provides an efficient scheme to determine whether a volume or a cell lies entirely inside the C-obstacle space. Essentially, this query enables us to annul impossible paths, because any path connecting the initial and goal configurations while passing through a full cell will be infeasible.

Although our generalized PD and C-obstacle query algorithms are conservative, the overall planners are still complete. To prove the completeness of the planners, we need to show the following lemma.

Lemma 2. *Any cell C that lies entirely in C-obstacle space can be correctly classified by performing a finite number of C-obstacle queries using Eq. (4) on the sub-cells from* *the subdivision of C.*

Proof. Since the cell C lies entirely in C-obstacle space, for every configuration $\mathbf{q} \in C$, the robot intersects with the obstacle. In the preprocessing step of the convex decomposition, we assume that convex pieces cover the original models completely, i.e. $\cup A_i = A$ and $\cup B_i = B$. Therefore, at $\mathbf{q}$, there exists at least one pair of intersecting convex pieces of the robot model and obstacle model. So the lower bound on generalized PD computed for every configuration $\mathbf{q} \in C$ is larger than 0. Denote the minimum over these lower bounds as η. Now we can subdivide the cell C uniformly. The level of subdivision is chosen so that the motion bound of the resulting sub-cell becomes less than η. According to the C-obstacle query criterion in Eq. (4), all these sub-cells lie entirely in C-obstacle space. Therefore, the cell C also lies in C-obstacle. $\square$

Since a cell that lies inside C-obstacle space can always be correctly classified within a finite number of subdivision, our ACD-based planners are complete.

An important assumption in this lemma is that the set of convex pieces of the decomposition must completely cover its original model; otherwise, the lower bound of generalized PD for a colliding configuration maybe 0 and the resulting C-obstacle query may not correctly classify the cells in C-obstacle. In another word, the completeness of our ACD-based planners is guaranteed by the condition of complete covering the original models by convex pieces. In practice, considering that the convex decomposition in our algorithm is performed during the preprocessing step, such decomposition is relatively easy to compute. For 2D polygons, there are good convex decomposition schemes (Keil, 2000). For 3D models, even a tetrahedral volumetric decomposition will satisfy.

According to the proof of this lemma, we can also infer that if the lower bound of generalized PD is guaranteed to be larger than 0 for overlapping models, our ACD-based planners will be complete. This can serve as a guideline on developing new lower bound generalized PD algorithms for complete motion planning.

5.5.2 Analysis of C-obstacle Query Algorithm

The complexity of our path non-existence computation is bounded by the number of subdivisions performed and the complexity of the cell labelling algorithms. We only analyze the complexity of the *C-obstacle cell query* algorithm that is bounded by the part of generalized PD computation since the part of motion bound can be computed in constant time. As shown in Section 4.1.5, the computational complexity of our lower bound generalized PD computation for 2D rigid objects A and B is $O(an_2 + bn_1)$, and for 3D rigid objects is $O(ab)$, where n_1 and n_2 are the number of convex pieces of the robot A and the obstacle B; a and b are the geometric complexity of all convex pieces of A and B.

5.5.3 Comparison

We compare our algorithms to some prior exact approaches such as (Avnaim and Boissonnat, 1989b; Banon, 1990), our algorithms are simple and efficient since they does not involve contact surface computation. Furthermore, our algorithms can be easily implemented for 4-DOF robots. In contrast, the prior exact approaches are difficult to implement and are prone to degeneracies due to the enumeration and computation of all the contact surfaces, especially for 4-DOF robots.

We also compare our algorithms for path non-existence with the star-shaped roadmap method (Varadhan and Manocha, 2005), especially because our approaches share some similarities with the star-shaped roadmaps method. Star-shaped roadmap method partitions the free space into star-shaped regions and for each star-shaped region computes a single point called a guard which can see every point in the region. In our approaches, the *empty* cells are a special case of star-shaped regions where any configuration in the cell can be considered as a guard. Moreover, we can label *empty* and *full* cells without relying on contact surface computation, which is simpler as compared to the star-shaped roadmap method.

Many hybrid approaches have been also proposed for efficient motion planning by combining different methods (Foskey et al., 2001; Hirsch and Halperin, 2003; Hsu et al., 2005; Morales et al., 2005). In particular, Hirsch and Halperin (2003) present a hybrid method that combines exact motion planning with probabilistic roadmaps, and apply it to planning the motion of two discs moving among polygonal obstacles. Our approach shares some similarities with this prior approach. Specifically, our method combines ACD with PRM, while their algorithm combines an exact cell decomposition approach with PRM. Conceptually, each of these algorithms computes two explicit representations to approximate the free space $\mathcal{F}$: a subset of $\mathcal{F}$, which is used to compute a collision-free path and a super set of $\mathcal{F}$, which is used to check for path non-existence. However, in our method, the approximate representation of free space can be incrementally refined using spatial subdivision, until a collision free path is found or path non-existence is confirmed. As a result, the strength of ACD approach is fully inherited, i.e. our hybrid method is complete. In Hirsch and Halperin's method, an approximate free space representation is computed as a preprocess and the subset and superset approximations of the free space can not be further refined. As a result, Hirsch and Halperin's method (Hirsch and Halperin, 2003) can decide path non-existence for some cases, but is not a complete motion planning algorithm. In addition, the implementation of their method is limited to disc robots, while our hybrid planner can be applied to robots with arbitrary shapes.

Completeness is another benefit of our hybrid method over probabilistic cell decomposition (Lingelbach, 2004), which is probabilistically complete and can not correctly handle motion planning scenarios where no path exists. On the other hand, based on our C-obstacle query algorithm, our hybrid method can report path non-existence.

5.5.4 Limitations

Our complete motion planning approaches have a few limitations. The C-obstacle cell query algorithm is conservative, which stems from the conservativeness of generalized PD and motion bound computations. Secondly, our path non-existence algorithm assumes that there are no tangential contacts on the boundary of the free space. Otherwise, the algorithm may not terminate. As a result, our algorithm can not deal with compliant motion planning, where a robot cannot pass through obstacles when the robot is not allowed to touch them. Moreover, when we apply our hybrid planner to 4-DOF or higher DOF problems, graph searching becomes one of the major bottlenecks. This is because the size of the connectivity graph increases as a function of the number of the cells in ACD. There is additional overhead of the two-stage hybrid algorithm. If there is a collision-free path, then the work performed in Path Non-existence Stage is unnecessary.

The complexity of each complete planner based on approximate cell decomposition varies as a function of the dimension of the configuration space. In the worst situation, our planners have an exponential complexity with the number of DOF of the robot. However, our experimental results show that our planners can work well on many complex 3-4 DOF problems as compared to the prior approaches.

5.6 Extension: Global Vector Field Computation for Feedback Motion Planning

We have focused on computing collision-free paths for robots. In this section, we show our complete motion planning approaches can also be extended for feedback motion planning, which deals with computing a feedback plan by computing a global vector field over the entire free space. Earlier work on decoupling the feedback and motion planning can be inefficient. Typically, paths computed by planners may not be smooth and can be difficult to track. On the other hand, it can be difficult to design feedback control strategies that take into account non-convex constraints induced by the obstacles in the environment. Therefore, when a feedback controller fails to steer the robot to follow a prescribed path, the robot often has to replan for a new path. In order to overcome these problems, feedback motion planning approaches take into account feedback concerns during collision-free path computation. Rather than planning a single collision-free path between the initial and goal configurations, these approaches compute a feedback plan over the entire free space of the robot that can converge towards the goal (Conner et al., 2003; Lindemann and LaValle, 2009). A feedback plan is often represented as a vector field over the free space. Moreover, the resulting vector field satisfies the convergence property, i.e. the robot is guaranteed to arrive at the goal configuration without colliding with any obstacles in the environment.

Most of the prior work on computing global vector fields for feedback motion planning is based on sequential composition (Conner et al., 2003; Lindemann and LaValle, 2009; Belta et al., 2005; Kloetzer and Belta, 2006). In these approaches, the robot's free space is decomposed into cells. A local vector field is computed within each cell and a global vector field is composed of individual vector fields. However, most prior algorithms are limited to low DOF robots and obstacles with simple shapes due to the difficulty of handling non-convex collision constraints (Kloetzer and Belta, 2006). Other approaches assume that a good cell decomposition of the robot's free space, e.g. the cylindrical algebraic decomposition (CAD), is given a priori (Lindemann and LaValle, 2009). Although CAD is an exact decomposition scheme and be used for general motion planning problems (Schwartz and Sharir, 1983), it is difficult to implement this scheme robustly and efficiently, even for low DOF robots. As a result, most prior practical algorithms for feedback motion planning have been limited to simple robots with three or fewer DOF.

In this section, we present a practical global vector field computation algorithm for smooth feedback motion planning (Zhang et al., 2009). Our algorithm performs approximate cell decomposition method that efficiently decomposes the robot's free space into rectanguloid cells adaptively. We construct a smooth vector field within each cell in the free space and address the issue of smooth composition between the non-uniform adjacent cells. We also show any integral curve over the vector field is guaranteed to asymptotically converge to the goal configuration, to avoid collision with the obstacles, and to be smooth. In practice, our algorithm is relatively simple to implement and we demonstrate its performance on planar robots with $2-3$ DOF, articulated robots composed of 3 serial links, and multi-robot systems with 6 DOF.

The rest of the section is organized as follows. We first formally define our problem. We then present the algorithm. Finally, we analyze some properties of the computed vector field and highlight the performance.

5.6.1 Problem Definition

In this paper, we consider a robot (or a multi-robot system) navigating in a static environment with non-convex obstacles. Given a goal configuration $\mathbf{q}_{goal} \in \mathcal{F}$, we need to compute a vector field V over the free space. For any configuration $\mathbf{q}$ in the same connected component in $\mathcal{F}$ as $\mathbf{q}_{goal}$, the vector field V needs to satisfy the following properties:

1. its integral curve starting from $\mathbf{q}$ over the vector field V should converge to the goal configuration $\mathbf{q}_{goal}$;
2. its integral curve should lie completely in the free space;
3. its integral curve is smooth (e.g. C_∞ differential).

The implication of such a vector field is that for any free configuration in the same component as $\mathbf{q}_{goal}$, there is always a collision-free path induced by the vector field. Therefore, such a vector field is a *feedback plan* for the given robot with the goal configuration $\mathbf{q}_{goal}$. In the next section, we describe our algorithm to compute such a vector field over the free space.

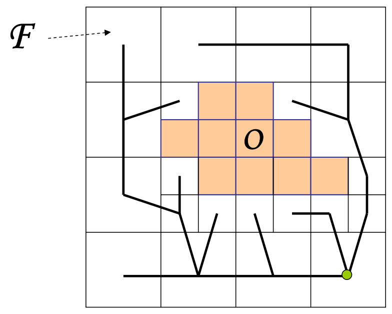


Figure 5.17: A discrete plan over approximate cell decomposition. *The robot's configuration space is subdivided into cells. A discrete plan as a tree is computed over all cells in the free space.*

5.6.2 Vector Field Computation Algorithm

In order to compute a vector field, we first perform *approximate cell decomposition* to subdivide the robot's C-space into cells. Given the decomposition, we compute a discrete plan which captures the global connectivity of the free space. We then construct a local vector field within each cell in $\mathcal{F}$. Finally, a global vector field over the free space is the composition of the local vector field associated with each cell.

Discrete Plan

In order to capture the global connectivity of the free space $\mathcal{F}$, we perform *approximate cell decomposition*. We use our cell labelling algorithms to obtain a set of *empty* cells, which provides an approximate representation of the robot's free space $\mathcal{F}$ (Fig. 5.17). A connectivity graph between the cells is extracted from the decomposition. Specifically, the connectivity graph G is defined where a node corresponds to an *empty* cell, and an edge denotes the adjacency between two *empty* cells in the decomposition.

We compute a discrete plan based on the connectivity graph. For any empty cell, a *discrete plan* specifies its successor cell so that when recursively following the successor cell, the *goal cell* that consists of the goal configuration $\mathbf{q}_{goal}$ will be finally reached. In order to compute such a discrete plan, we first locate the node in the connectivity graph G corresponding to the goal cell. Beginning at this node, we perform the breadth first

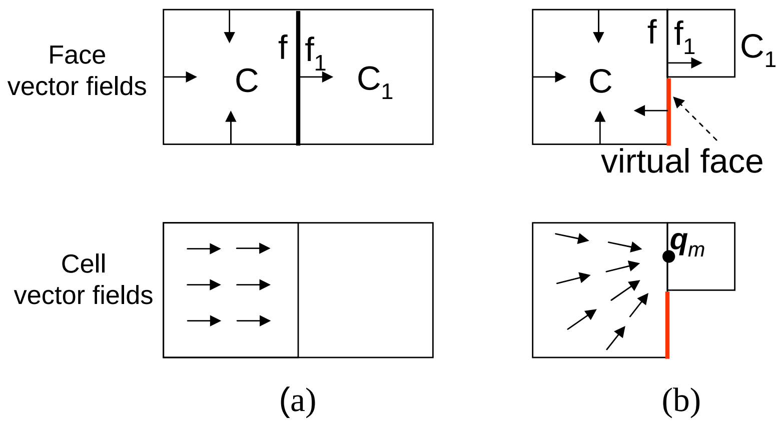


Figure 5.18: Face and cell vector fields for the two cases: uniform and non-uniform adjacent cells. *f is a face in C and f_1 is a face in C_1. We consider two different cases: Case (a), $f \subseteq f_1$ and Case (b), $f \supset f_1$.*

search (BFS) on the graph G and compute a tree that corresponds to that search. The tree represents a discrete plan and the cell for each node has only one successor cell (i.e. the parent node in the tree) except the goal cell, which can have multiple descendant cells (i.e. the children nodes in the tree).

In our algorithm, the decomposition over C-space can be refined incrementally. Initially, the approximate cell decomposition computes a coarse approximation of the free space of the robot. If no path can be computed using the coarse approximation (e.g. the robot's initial configuration lies in a mixed cell), we refine the decomposition. Specifically, we iteratively subdivide a fraction of all mixed cells identified by the first graph cut algorithm until a path connecting a robot's initial configuration and goal configuration is computed or no such path exists for this problem (Latombe, 1991; Zhang et al., 2008c). The discrete plan is computed based on the resulting decomposition.

Vector Field Computation within Cells

Based on the discrete plan described above, we compute a smooth vector field for each *empty* cell. If the cell is an *intermediate cell* in $\mathcal{F}$ (i.e. does not consist the goal configuration), the local vector field in C guides the robot through the cell to its unique successor cell; if it is a *goal cell*, the vector field brings the robot to the goal configuration. We also desire that the vector field computed in each cell be smooth. In order to compute such a local vector field, we follow the general scheme described in (Lindemann and LaValle, 2009). For every face of the cell, we choose a *face vector field* V_f defined over the points on the face. We further choose a *cell vector field* V_c for points within the cell. The overall vector field V defined at any point p in the cell is a smooth interpolation of V_f and V_c.

The different resolutions or sizes between adjacent cells pose a difficulty in terms of choosing the appropriate face or cell vector fields. Consider an intermediate cell C with its successor C_1 (Fig. 5.18). In n dimensional C-space, each cell has $2n$ faces. Suppose the face f in C is sharing a boundary with the face f_1 in C_1. We consider two different cases:

Case (a): Fig. 5.18(a) shows this simpler case where the size of f is smaller than or equal to f_1 (i.e. $f \subseteq f_1$). Here the face f is defined as an *exit face* of the cell C since the vector field of C needs to cross the face to enter its successor C_1. Therefore, the *face vector field* V_f is chosen to be orthogonal to the face and points outwards. For other faces in C, their *face vector fields* are chosen to be orthogonal and point inwards. Finally, the *cell vector field* V_c is chosen to be identical to the face vector field of the exit face (Fig. 5.18(a)).

Case (b): The other case is when $f \supset f_1$ (Fig. 5.18(b)). In this case, the face vector field of f can not always point outwards. Otherwise, the vector field for the region $f - f_1$ will not guide the robot to its successor cell C_1. To order to overcome this problem, a straightforward solution is to further subdivide the cells until the size of each cell becomes the same (i.e. a uniform subdivision of the entire free space). However, this can result in too many cells in the overall decomposition. Other approaches such as splitting the bigger cell so that $f = f_1$ are difficult to be implemented for high dimensional space. Rather, we introduce the notion of a *virtual face*. As shown in Fig. 5.18(b), for the

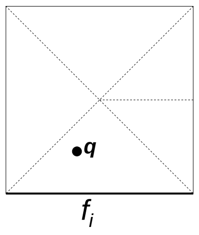


Figure 5.19: GVD over a cell. *A GVD is defined for all faces (including the virtual faces) of a cell. For a configuration* $\mathbf{q}$*, a influencing face* f_i *is defined as the closest face of its cell.*

face f on C, over the region shared by both C and C_1, the *face vector field* is defined as pointing outwards and such a region is also referred to as an *exit face.* The rest of the region in f is covered by multiple virtual faces. Each virtual face has the same size as the exit face, and its face vector field is defined as pointing inwards. Finally, the *cell vector field* V_c at a configuration $\mathbf{q}$ in the cell C is chosen by normalizing the vector $\overrightarrow{\mathbf{q}\mathbf{q}}_m$, where $\mathbf{q}_m$ is the centroid point in the face f_1. It should be noted in our implementation, we don't instantiate the virtual faces. Rather they can be interpreted indirectly from the adjacent cells to C.

Next, we deal with the goal cell. There is no exit face for the goal cell. Therefore, every *face vector field* V_f can be chosen to be orthogonal to the face and point inwards. The *cell vector field* $V_c(\mathbf{q})$ is chosen as the normalization of the vector $\overrightarrow{\mathbf{q}\mathbf{q}}_{goal}$.

With the face vector fields V_f and the cell vector field V_c defined for all cells in the free space, we smoothly interpolate between them as (Lindemann and LaValle, 2009) to compute the overall vector field in each cell. Given a configuration $\mathbf{q}$ in an intermediate cell, we first determine the closest face f_i in this cell and define it as the *influencing face.* Therefore, all the faces of the cell determine a generalized Voronoi diagram of cell (GVD) as shown in Fig. 5.19. Now, $V(\mathbf{q})$, the vector field at configuration $\mathbf{q}$ is defined as:

$$V(\mathbf{q}) = norm((1 - b(\mathbf{q}))V_{f_i}(\mathbf{q}\perp) + b(\mathbf{q})V_c(\mathbf{q})). \tag{5.6}$$

Here $\mathbf{q}_\perp$ is the projection of $\mathbf{q}$ on the face f_i. $b(\mathbf{q})$ is a *weighting function* or *bump*

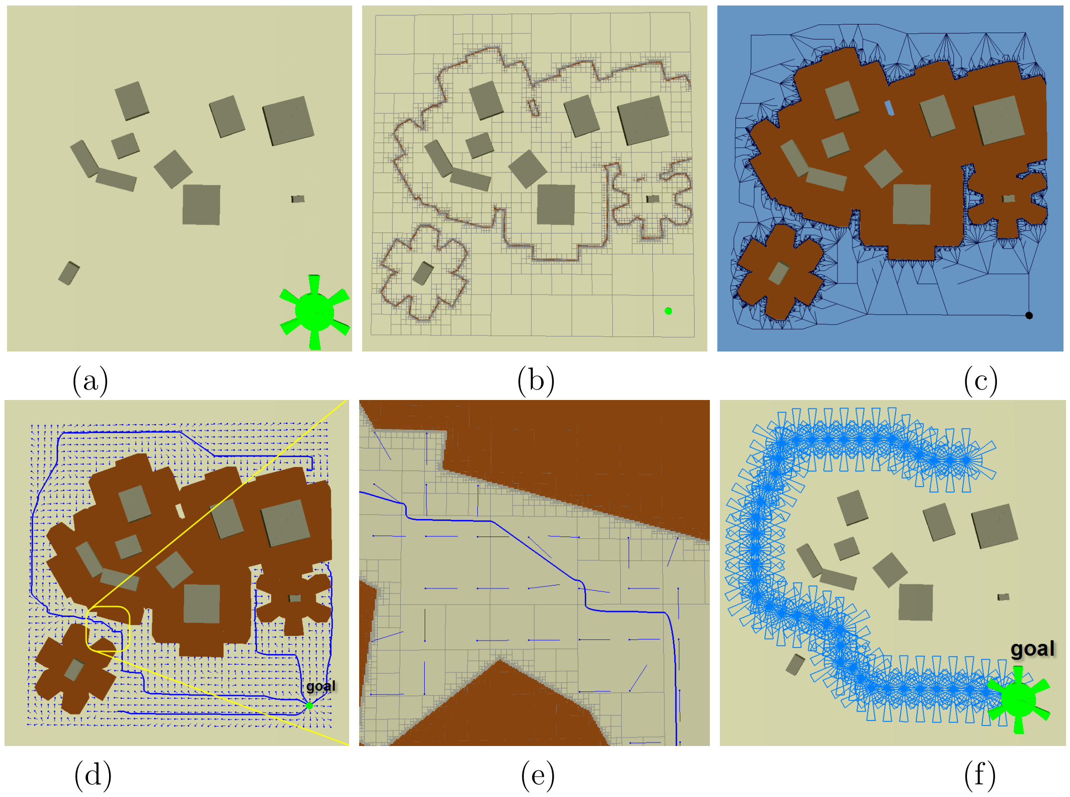


Figure 5.20: ‘Gear’ example. *(a) The problem is to compute a feedback plan for a 2-DOF translating gear-shaped robot in an environment with static obstacles. (b) The C-space is subdivided into cells adaptively. (c) A discrete plan is computed as a tree with the root highlighted as the black dot. (d) The vector field over the free space. The shaded regions denotes C-obstacles. For any collision-free configuration which is in the same component as the goal configuration, its integral curve converges to the goal. Any integral curve is smooth, though it may have high-variation in some region. (e) The zoom-in on the yellow region in (e) shows the integral curve is smooth. (f) The robot moves along an integral curve towards the goal.*

function, whose value is 0 when $\mathbf{q}$ is on any face of the cell, and 1 on the GVD of the cell and interpolates between them according to the ratio of the distance to the influencing face and the other faces of the cell. In theory, any interpolation function that satisfies these conditions can be used. We use a C^∞ function presented in (Lindemann and LaValle, 2009). Moreover, in a goal cell, a subdivision is defined by considering the convex hull of the goal configuration $\mathbf{q}_{goal}$ and every face of the cell. The influence face then is computed based on the convex hull that the configuration $\mathbf{q}$ lies in.

Finally, the global vector field over the entire free space from the decomposition is the composition of the vector field associated within each cell.

5.6.3 Analysis

In this section, we analyze the properties of our vector field computation algorithm.

Theorem 5. (**Convergence**) *For any configuration* $\mathbf{q}$ *in the same connected component in* $\mathcal{F}$ *as* $\mathbf{q}_{goal}$*, the integral curve starting from* $\mathbf{q}$ *over the computed vector field* V *asymptotically converges to* $\mathbf{q}_{goal}$*.*

Building on the formulation shown in (Lindemann and LaValle, 2009), we prove the convergence property by first showing that any integral curve over the vector field in an intermediate cell must reach its exit face and proceed to its successor cell. Specifically, we prove the following two lemmas. Here, a GVD is defined for all faces of an intermediate cell C (Fig. 5.19).

Lemma 3. *Any integral curve cannot cross a GVD face of an intermediate cell* C *more than once.*

Proof. According to Eq. (5.6), V at any point $\mathbf{q}$ on a GVD face is equivalent to $V_c(\mathbf{q})$. Therefore, for a GVD face with normal n, the sign of dot product between n and $V_c(\mathbf{q})$ for any $\mathbf{q}$ on the face is fixed for both cases we have considered. Consequently, a GVD face cannot be crossed more than once. □

Lemma 4. *Any integral curve in an intermediate cell* C *will reach its exit face.*

Proof. Denote the influencing face of $\mathbf{q}$ in C as f_i. We determine the first GVD face f_1 that intersects with the ray from the point $\mathbf{q}$ with the direction $V(\mathbf{q})$. If there is no intersection, $\mathbf{q}$ is in the Voronoi region of the exit face and it is obvious that the integral will continue to the exit face. Otherwise, the signs of the dot products between the normal of f_1 and the vectors $V_c(\mathbf{q})$, and between this normal and V_{f_i} will be the same. The overall vector field at $\mathbf{q}$ is a linear combination of V_c and V_{f_i}. Therefore, on the integral curve, the distance to f_1 is **always decreasing**. Consequently, either f_1

or some other GVD face will be reached by the integral curve. Since there are a finite number of GVD faces in each intermediate cell and the integral curve cannot cross a GVD face more than once (Lemma 3), the integral curve finally will reach the exit face of the cell. □

To complete the proof of the convergence property, we still need to show that the integral curve in a goal cell terminates at the goal configuration. For any point $\mathbf{q}$, the distance to goal configuration $\mathbf{q}_{goal}$ is **always decreasing** since $V_f \cdot \overrightarrow{\mathbf{q}\mathbf{q}}_{goal} > 0$, $V_c \cdot \overrightarrow{\mathbf{q}\mathbf{q}}_{goal} > 0$. Consequently, the dot product for its linear combination of V_f and V_c is larger than 0.

Collision-free Planning: Our integral curve is guaranteed to be collision-free, i.e. fully lie in the free space. This holds since in an intermediate cell, except for the exit face, the face vector field always points inwards and the weighting function for any point on the face is 0. Therefore, the only way an integral curve exits an intermediate cell is to cross through the exit face to its successor cell, which lies in the free space. In the goal cell, all the face vector fields point inwards. Therefore, the integral curve cannot exit from the goal cell. In conclusion, the integral curve fully lies in the free space and is guaranteed to be collision-free.

Smoothness: Same as (Lindemann and LaValle, 2009), our integral curve is also smooth or C^∞ differentiable. Within a cell, the vector field is smooth except on a set of measure zero (the $d-2$ dimensional boundary of a cell; e.g. the vertices of a cell in 2D) since the chosen weighting function b for interpolating V_c and V_f is smooth. Furthermore, the vector field on the face between a cell and its successor is also smooth since all the derivatives of the weighting function on the boundary are 0. Therefore, the integral curve over V is smooth.

Algorithm Complexity: The complexity of the overall algorithm is governed by the approximate cell decomposition step for computing the discrete plan. In the worst case, its complexity can increase exponentially with the number of DOF. The local vector

	Gear	Gear (3-DOF)	L-Shape	3-Link	Multi-Robot
# DOF	2	3	3	3	6
$t_{dec}(s)$	3.01	10.5	9.84	186.2	23.97
level of subdivision	11	6	6	5	3
# Cells	46,510	133,512	182,288	72,262	297,424
Memory (MB)	38	242	304	190	451
Per Point Loc.(μs)	1	2	2	2	2
Per Evaluation(μs)	13	26	24	47	61
Eva. Frequency:kHZ	78.9	38.2	41.1	21.2	16.3

Table 5.7: Performance of our global vector field computation algorithm on different examples.

field computation algorithm within each cell has complexity of $O(n)$ for determining the *influencing face* of a given configuration, where n is the number of faces (including the virtual faces) of the cell.

5.6.4 Experimental Results

We have implemented our vector field computation algorithm. We use an approximate cell decomposition method developed in (Zhang et al., 2008c) to subdivide the C-space. The implementation is general for arbitrary dimensional C-space. In our implementation we parameterize the rotation using Euler angles. During the decomposition, the angles are partitioned into intervals and are allowed to wrap around. The cell labelling to determine whether a cell lies entirely in free space or C-obstacle is performed by computing the separating distance or penetration depth computation between the robot and obstacles (Zhang et al., 2008c). We compute the discrete plan as a tree using the breadth first search. We choose the appropriate face vector field and cell vector field for the two cases as described in Section 5.6.2. In order to compute an integral curve, we use the Runge-Kutta integration method.

We have tested our implementation on planar robots with 2-3 DOF and multi-robot systems up to 6 DOF. Fig. 5.20 shows a gear-shaped robot navigating in a 2D plane with 2 translational DOF. We compute the vector field for the given goal configuration

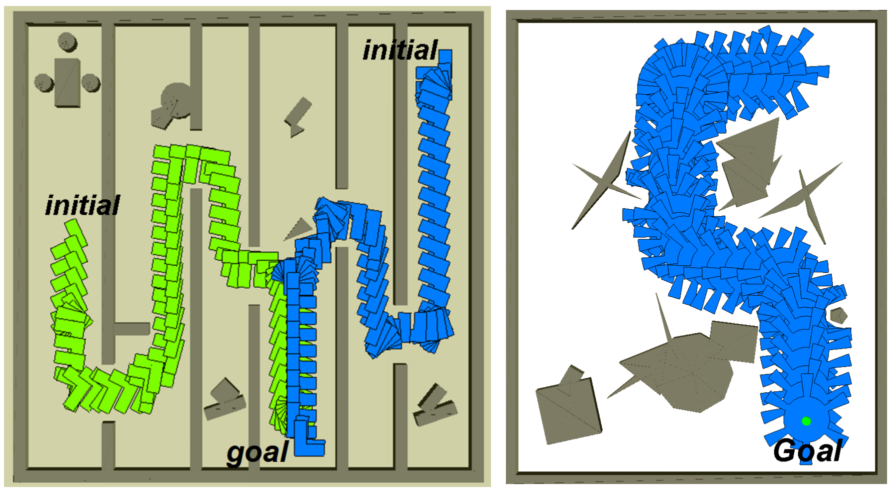
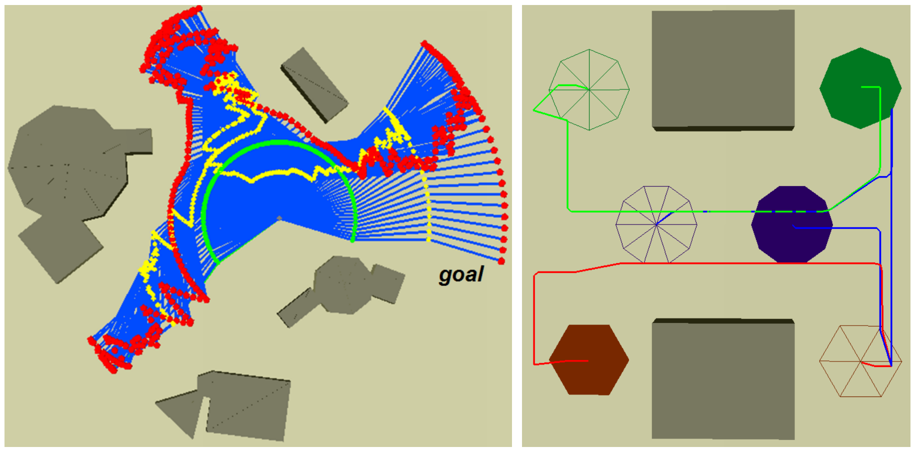


Figure 5.21: *(a) 3-DOF L-shaped robot (b) 3-DOF gear-shaped robot*

and highlight the integral curve for an initial configuration. The integral curve path is collision-free and converges to the goal configuration. It should be noted the integral curve is indeed smooth, though the curve haves high variation in some regions. Fig. 5.21 shows 3-DOF robots navigating in a 2D environment. The robots can translate and rotate in the plane.

We have applied our algorithm to articulated robots. Fig. 5.22(a) shows an example on an articulated robot composed of 3 serial links in a plane. The level of subdivision is 5, i.e in each dimension 2^5 decompositions are applied. Next, the first graph cut algorithm mentioned in Section 5.6.2 is used to incrementally refine the decomposition. Finally, a global vector field is constructed and used to guide the robot to the goal. Due to the underlying axis-aligned decomposition, the robot sometimes moves one link a time.

Fig. 5.22(b) shows an example of feedback planning for a multi-robot system. The system is composed of three robots, and each robot has 2 translational DOF. We compute a feedback plan for this system. This example may be difficult for a decoupled multi-robot planner. When moving towards its goal, each robot is blocked by other robots. Instead, we perform approximate cell decomposition on the composite 6D con-



(a) (b)

Figure 5.22: *(a) The vector field is computed for an articulated robot composed of 3 serial links. It guides the robot moving towards its goal* $\mathbf{q}_{goal}$*. (b) Vector field computation for a multi-robot system with 3 translating planar robots.*

figuration space and compute vector fields on 6D cells in the free space. Given an initial configuration for the multi-robot system, we compute an integral curve over the global vector field. The motion for each robot is computed by projecting the 6D integral curve in a 2D plane. It should be noted that the integral curve in 6D is smooth. However, as shown in the figure, the motion for each robot may exhibit high variation or "cusps" due to the projection to a lower dimensional space.

Table 5.7 highlights the performance of our vector field computation algorithm. In the table, t_{dec} denotes the total timing to perform the approximate cell decomposition, which is the major computation-intensive step in our algorithm. The total number of cells and the memory usage are also reported for each example. Overall, our algorithm can efficiently compute vector fields for smooth feedback planning for these examples.

We have also tested the performance on evaluating the constructed vector filed (shown in Table 5.7). For each example, we randomly generate 1 million samples in the robot's configuration space. Based on the approximate cell decomposition, we can efficiently locate the cell containing each sample in the logarithmic complexity. For any sample lying inside an empty cell, we evaluate its vector. Each evaluation takes only $13 - 61$ microseconds on average; the corresponding frequency is $78.9 - 16.3$ kHZ.

Limitations. We discuss some limitations of our global vector fiele method. Though the integral curves are guaranteed to be C^∞ smooth, the curves can sometimes have sharp turns due to the underlying adaptive decomposition. Furthmore, no smooth vector field exists in a non-contractible free space. However, our vector field has additional non-smoothness on some boundaries of the cells: if the nodes in the discrete plan for two adjacent cells are not connected by an edge, the vector field along their common face is not smooth. To overcome this limitation, one may simplify the discrete plan.

5.7 Summary

We present efficient complete motion planning approaches for low DOF robots. We first present a simple approach to check for path non-existence. Based on approximate cell decomposition, our approach uses the C-obstacle cell query to efficiently check whether a cell in C-space lies entirely inside the C-obstacle region. We describe simple and efficient algorithms to perform this query using *generalized penetration depth* computations. The C-obstacle query algorithm is general for 2D or 3D rigid robots, or articulated robots. The overall planner is complete and we highlight its performance on 3-4 DOF robots.

In order to further improve the efficiency of the complete planner, we present a novel algorithm that combines the completeness of ACD with the efficiency of a probabilistic roadmap approach. The improved planner is also complete. We apply the planner to 4 DOF rigid robots, and observe significant improvement in performance over our first complete planner.

We also extend our complete motion approaches to compute a global vector field in the entire free space of the robot for feedback motion planning. We compute smooth vector field for each cell in the free space and address the issue of smooth composition between non-uniform adjacent cells. As compared to prior work, our algorithm is practical for robot and obstacles with non-convex shapes and is simple to implement.

Chapter 6

A Retraction-based Planner for Cluttered Environments

Sampling-based planning algorithms such as probabilistic roadmaps (PRM) (Kavraki et al., 1996) or rapidly-exploring random trees (RRT) (LaValle, 1998; Kuffner and LaValle, 2000) have been widely used to compute collision-free paths for robots in complex environments. These algorithms generate samples using randomized techniques and attempt to capture the connectivity of the free space by graphs or trees. These algorithms are very simple to implement and have been successfully applied to high-DOF robots.

The performance of sampling-based planning algorithms, however, can degrade significantly if a robot needs to operate in a cluttered environment or the free space of a robot has narrow passages. Due to the small volumes of these passages or regions, it is difficulty to generate an adequate number of samples. Interestingly, the motion planning scenarios that arise in part removal and part disassembly simulations are rather challenging in terms of narrow passages because parts in the simulations usually tightly fits with each other.

In this chapter, we present a retraction-based sampling algorithm to improve the performance of sampling-based planners in narrow passages (Zhang and Manocha, 2008b). We formulate the retraction step as a generalized PD problem which computes the closest boundary configuration for a given colliding configuration (Sections 6.1-6.3). We integrate our retraction-based sampling with a RRT planner in Section 6.4 and analyze the performance of our planner using Voronoi diagram in Section 6.5. Based on our retraction-based sampling, we present D-Plan approach for part disassembly simulation in Section 6.6 (Zhang et al., 2008a). We highlight the efficiency of our approaches in Section 6.7.

6.1 Introduction

The performance of sampling-based planners can degrade if the free space has narrow passages. The narrow passages are classified as regions, whose removal or perturbation can change the connectivity of the free space (Hsu et al., 1998, 2006). Figure 1.4 shows the well-known alpha puzzle benchmark, which is widely regarded as a challenging benchmark for motion planning algorithms. The problem is mainly caused by the small volume and poor visibility of narrow passages in free space. Because the volume is small, it can be difficult to generate an adequate number of samples in these regions of the free space by performing uniform or randomized sampling in C-space. Furthermore, narrow passages may also exhibit the poor visibility, i.e. nearby samples are more difficulty to be connected by straight lines in the free space. Thus, more samples are needed to capture the overall connectivity in these regions. Many techniques have been proposed in the literature to improve the performance of these planners in narrow passages. These include use of workspace information to guide the sampling (Kurniawati and Hsu, 2006), use of filters to reject samples (Boor et al., 1999; Simeon et al., 2000) and retraction-based planning (Amato et al., 1998; Hsu et al., 1998; Saha et al., 2005). In this chapter, we primarily focus on improving the performance of retraction-based planners.

One of the main steps in retraction-based planning is to retract a sample or a config-

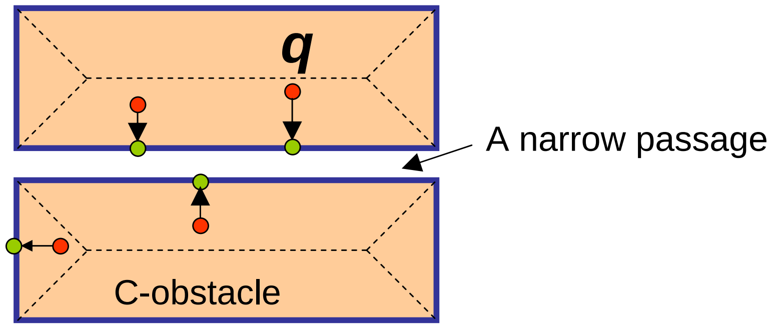


Figure 6.1: Retraction-based sampling for narrow passages. *The basic idea is to retract a randomly generated configuration to a more desirable region in the free space. A desirable location for retraction of a colliding configuration* $\mathbf{q}$ *is the closest boundary point in the free space. Intuitively, the given colliding configuration, e.g.* $\mathbf{q}$ *which is close to the boundary, is retracted into the narrow passage. This increases the number of samples in this narrow passage.*

uration to a more desirable region of the free space. This includes moving samples close to the boundary of the configuration space obstacle (C-obstacle) or near the medial axes of the free space. One specific retraction strategy is to retract any *colliding* configuration (a configuration in C-obstacle) to the closest boundary point of C-obstacle. In practice, this is equivalent to computing the *generalized penetration depth*, which is presented in Chapter 2. However, exact computation of generalized penetration depth has very high complexity. As a result, prior planners use simple heuristics to perform the retraction step, and their performance varies with the shape of the robot and the obstacles, and their relative placement. Other retraction-based approaches for handling narrow passages include dilation-based planners (Saha et al., 2005; Cheng et al., 2006). These algorithms dilate the free space by considering samples that lie inside the C-obstacle space and are close to its boundary. However, most of dilation-based planners can be hard to implement as they need a robust technique to perform the dilation or shrinking operation on general polygonal models.

A Retraction-based Planning: In this chapter, we first present a novel optimization-based retraction algorithm for 3D rigid robots. The retraction step is formulated as an optimization problem for generalized PD computation using an appropriate C-space distance metric. Our algorithm computes samples near the boundary of C-obstacle using local contact analysis and uses those samples to improve the performance of RRT planners in narrow passages. We analyze the performance of our planner using Voronoi diagrams and show that the tree can grow closely towards any randomly generated sample. Our algorithm is general and applicable to all polygonal models. We have implemented our planner and highlight its performance on difficult scenarios with narrow passages. As compared to the basic RRT algorithm, we observe significant improvement in the running time and the number of generated samples.

D-Plan: Efficient Collision-Free Path Computation for Part Removal and Disassembly: We also address the application of part disassembly simulation using our retraction-based planner. The problems of assembly maintainability and mechanical part disassembly frequently arise in design and manufacturing applications. The manual generation of detailed disassembly or maintainability paths can be tedious and time consuming, particularly in environments prone to frequent design changes. The recent trend has been towards developing automated algorithmic solutions for such design problems that can automatically compute a collision-free, global path. These simulation technologies are increasingly used for virtual prototyping and PLM (product lifecycle management), where the goal is to provide efficient software solutions to problems that were traditionally solved using costly physical mockups.

Based on our retraction-based planner, we present an efficient approach - D-Plan for part disassembly simulation and virtual prototyping of part removal. In order to effectively handle the tight-fitting scenarios arisen in virtual prototyping, we use the retraction-based sampling technique to generate samples in narrow passages.

Our D-Plan approach is applicable to general, complex, polygon soup models. Such models are increasingly used in virtual prototyping and PLM, since many CAD systems import models generated from other sources, and sometimes the translators do not maintain the connectivity information. We present techniques to perform efficient contact query among polygon soup models and compute their closest features pairs. We further improve the performance of our planner by performing localized collision detection and exploit the spatial coherence between nearby queries in the configuration space. We highlight the performance on many challenging benchmarks including alpha puzzle benchmark, maintainability of the windscreen wiper motion, and disassembly of a seat from the interior of a car body.

6.2 Related Work

In this section, we give a brief overview of prior work in sampling-based motion planning and retraction-based methods.

6.2.1 Sampling-based Planning

As compared to other motion planning algorithms, sampling-based approaches can deal with high degree-of-freedom motion planning problems, handle complex scenes and are relatively simple to implement and. However, these algorithms may not work well when the free space has narrow passages. The main challenge is to generate sufficient number of samples that can capture the connectivity of free space through narrow passages. Many sampling strategies have been proposed to improve the performance of these planners. See (Hsu et al., 2006) for a recent survey. These include use of workspace information to guide the sampling (van den Berg and Overmars, 2005; Kurniawati and Hsu, 2006), filters to reject samples (Boor et al., 1999; Simeon et al., 2000; Sun et al., 2005), adaption of the sampling distribution based on history (Morales et al., 2005), and retraction-based methods (Amato et al., 1998; Hsu et al., 1998; Wilmarth et al., 1999; Redon and Lin, 2006; Saha et al., 2005; Cheng et al., 2006). Another issue with sampling-based motion planning approaches is that they are probabilistically complete. If there exists a solution path, these algorithms will find one with high probability. However, when there exists no path, these approaches can not decide it.

6.2.2 Retraction-based Motion Planning

The retraction-based approaches have been widely used to improve the performance of sample-based planners in narrow passages (Amato et al., 1998; Hsu et al., 1998; Wilmarth et al., 1999; Foskey et al., 2001; Redon and Lin, 2006; Rodriguez et al., 2006). The main idea is to retract a randomly generated configuration towards a more desirable region, e.g. to the closest point on the boundary of C-obstacle (Fig. 6.1) or the medial axis of the free space.

The main challenge in retraction-based approaches is that the retraction step may involve complicated or non-trivial computation. For example, computing the closest boundary point for a colliding configuration boils down to generalized penetration depth computation based on an appropriate distance metric. The computation of globally optimum penetration depth has high complexity (Zhang et al., 2007c). As a result, most algorithms use heuristics to compute samples near the boundary of C-obstacle or in the contact space (Amato et al., 1998; Wilmarth et al., 1999; Redon and Lin, 2005; Rodriguez et al., 2006). Other approaches include dilation-based planning (Hsu et al., 1998), and current practical solutions for them compute an approximate medial axes of the model (Saha et al., 2005), shrink the boundary using tetrahedral decompositions (Cheng et al., 2006), or estimate the bound of the motion for the moving robot (Ferr and Laumond, 2004). In practice, except (Ferr and Laumond, 2004), most of these techniques are limited to closed models and can be susceptible to robustness issues and degeneracies.

6.2.3 Contact Space Planning

Many retraction-based approaches tend to generate more samples near the *contact space*, the subset of the configuration space (C-space), which consists of the configurations when the robot touches one or more obstacles without any penetration. Contact space planning has been shown useful for handling narrow passages (Redon and Lin, 2006), along with manipulator planning and compliant motion planning. There is considerable work on contact modeling using the geometric or algebraic formulation, sampling, and local compliant planning (Donald, 1987; Hirukawa et al., 1994; Ji and Xiao, 2001; Xiao and Ji, 2001).

6.2.4 Applications to Part Disassembly Simulation

Assembly and disassembly planning is a broad topic that has been extensively studied in CAD/CAM, virtual prototyping and motion planning. It mainly deals with the sequencing of the (dis)assembly operations of multiple parts to (dis)assemble a product (Latombe, 1991; Wilson and Latombe, 1994; Lambert, 2003). In this paper, we only focus on specific problems on part removal or maintainability study, where one needs to compute a collision-free path for a particular part that needs to be removed from an assembly. Part disassembly problem can be reduced to a motion planning problem, where the part to be extracted is treated as a robot and the rest of the assembly parts are treated as static obstacles. Some specialized and efficient motion planning algorithms based on random sampling and diffusion have been proposed for part disassembly and integrated into the KineoCAM commercial software (Ferr and Laumond, 2004).

6.3 Optimization-based Retraction

In this section, we present our optimization-based retraction algorithm for sample generation. We use this algorithm to improve the performance of RRT planners in Section

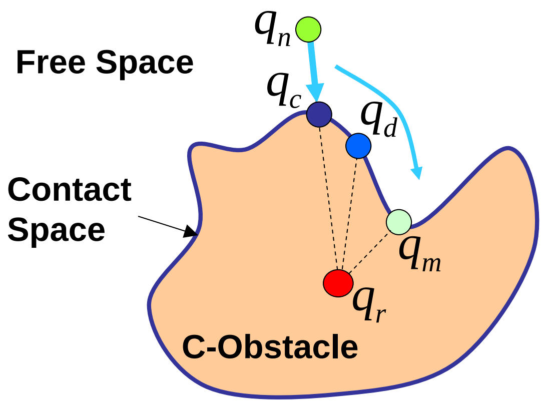


Figure 6.2: Optimization-based retraction. *Given a colliding sample* $\mathbf{q}_r$*, our algorithm retracts it to the locally closest point* $\mathbf{q_m}$ *on the boundary of C-obstacle by iterative optimization. In this case,* $\mathbf{q_n}$ *is the initial guess, while* $\mathbf{q_c}$ *and* $\mathbf{q_d}$ *are intermediate samples during the optimization.*

6.4. Given a colliding sample, our algorithm retracts this sample to a more desirable location, i.e. the closest point on the boundary of C-obstacle or *contact space*. This idea is similar to other retraction-based sampling strategies such as the one used in OBPRM (Amato et al., 1998), which also tend to generate samples near the contact space to improve the performance of the planner in narrow passages. The main difference is that our retraction step is formulated as generalized PD computation and performed using iterative optimization.

6.3.1 The Retraction Step

As shown in Fig. 6.2, given a colliding sample $\mathbf{q}_r$, the *retraction step* is to compute its closest boundary point $\mathbf{q_m}$, which can be formally defined as:

$$\mathbf{q_m} = \arg\min_{\mathbf{q}} \delta(\mathbf{q}, \mathbf{q}_r), \mathbf{q} \in \mathcal{F} \cup \mathcal{C}_{contact}, \tag{6.1}$$

where δ is a distance metric defined in the configuration space of the robot, and the configuration $\mathbf{q}$ lies in the free space $\mathcal{F}$ or the contact space $\mathcal{C}_{contact}$ of the robot. According to Theorem 1, for DISP, OBNO or TRAJ C-space distance metric, the optimal configuration $\mathbf{q_m}$ must lie in the contact space.

The retraction step computation is equivalent to generalized PD computation. Therefore, we can apply techniques developed for generalized PD to perform the retraction computation. More specifically, the generalized PD algorithm using constrained optimization (Section 4.2) is employed since the algorithm can deal with complex non-convex models. Furthermore, this generalized PD algorithm can also benefit from the samples in the planner for choosing a good initial guess. We briefly present our optimization algorithm in the context of retraction computation.

We formulate the retraction computation (Eq. 6.1) as a constrained optimization problem. The objective function is based on the distance metric δ such as DISP, OBNO or TRAJ in C-space, and the constraint is that the resulting configuration needs to lie in $\mathcal{C}_{contact}$. As shown in Fig. 6.2, to retract a given colliding sample $\mathbf{q}_r$, our method starts with a *non-colliding* sample $\mathbf{q_n}$ (either collision-free or in the contact space) as the initial guess. The algorithm then performs the following steps in an iterative manner:

1. Project $\mathbf{q_n}$ into the contact space in order to generate a sample $\mathbf{q}_c$ in the contact space;

2. Perform a *contact query*, i.e. compute the closest feature pairs within a tolerance distance between the robot at $\mathbf{q}_c$ and the obstacles;

3. Searching over the local contact space formed by the closest feature pairs, compute a new non-colliding sample $\mathbf{q_d}$, which locally minimizes the distance to the sample $\mathbf{q}_r$ according to a distance metric δ;

4. Assign $\mathbf{q_n} = \mathbf{q_d}$, and go to Step 1.

These steps are iterated until the distance to $\mathbf{q}_r$ cannot be further reduced, which means a sample $\mathbf{q_m}$ realizing a local minima is found, or the maximum number of iterations has been reached. We use the symbol S to represent the sequence of samples $\mathbf{q_n}$, $\mathbf{q}_c$, and $\mathbf{q_d}$ generated by each iteration, excluding the duplicated or colliding samples. The distance

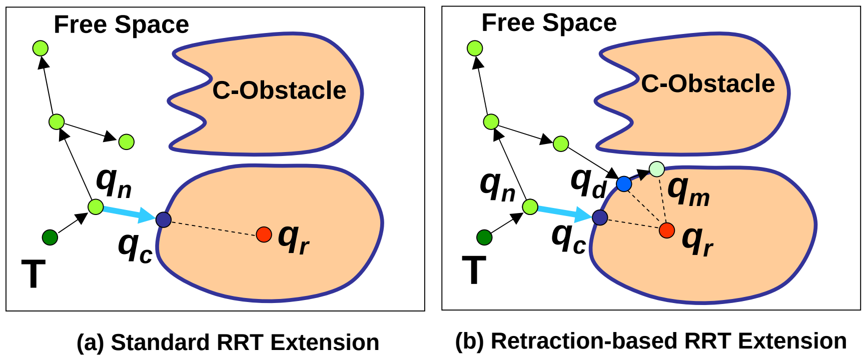


Figure 6.3: Difference of tree extension between RRT and RRRT. *(a) Given a randomly generated configuration* $\mathbf{q_r}$*, the basic RRT extension scheme grows the tree - T from its nearest node* $\mathbf{q_n}$ *towards* $\mathbf{q_r}$*, stopping at the configuration* $\mathbf{q_c}$ *on the boundary. (b) In our retraction-based extension, we retract* $\mathbf{q_r}$ *to the free space by optimization. The retraction step generates a sequence of non-collision configurations* $S = \{\mathbf{q_c}, \mathbf{q_d}, ..., \mathbf{q_m}\}$*, where* $\mathbf{q_m}$ *is a local minima point of the distance* δ *to* $\mathbf{q_r}$*. We then extend the tree to every configuration in S using the basic RRT extension. Therefore, the tree in our algorithm can grow towards* $\mathbf{q}_r$ *closely.*

of every sample in the sequence S to $\mathbf{q}_r$ strictly decreases, i.e. $\delta(\mathbf{q_i}, \mathbf{q}_r) > \delta(\mathbf{q_{i+1}}, \mathbf{q}_r)$, and it monotonically approaches the local minima $\delta(\mathbf{q_m}, \mathbf{q}_r)$. The generated samples in S can be used by any sampling-based planner.

Our algorithm can efficiently optimize over the contact space and compute a local minima. The optimization algorithm only needs to perform collision detection, along with *local contact analysis* to sample and search over the local contact space. Such computation can be implemented for any type of polygonal models, including polygon soup models. As a result, our algorithm is general and applicable to all general polygonal models.

6.4 Retraction-based RRT Planner

In this section, we use the optimization-based retraction algorithm to improve the performance of the rapidly-exploring random tree planners (RRT). Prior retraction-based sampling strategies have mainly been applied to PRM planners and only retract the samples that lie in C-obstacle space. In our case, we retract many of the generated samples including the ones that belong to the free space.

6.4.1 RRT Planner

The RRT algorithm explores the free space by randomly sampling and building a tree (Fig. 6.3-(a)). RRT's are used to search high-dimensional spaces with both algebraic constraints (arising from obstacles) and differential constraints (e.g. the non-holonomic constraints). The basic RRT algorithm is as follows. Starting with a tree T with a root node, the algorithm iteratively adds more nodes to the tree. During each iteration, a configuration $\mathbf{q}_r$ is randomly generated, and the basic RRT algorithm attempts to connect the nearest node $\mathbf{q_n}$ in the tree T to $\mathbf{q}_r$ by a straight line in the configuration space (Fig. 6.3-(a)). If the configuration $\mathbf{q}_r$ and $\mathbf{q_n}$ can be connected via a collision-free path, the tree is extended from $\mathbf{q}_r$ to $\mathbf{q_n}$ and grows. Otherwise, the planner computes $\mathbf{q}_c$, the first *contact* configuration (a configuration in the contact space) on the straight line from $\mathbf{q_n}$ to $\mathbf{q}_r$. The tree then extends to $\mathbf{q}_c$. We refer to this way of growing the tree as the *basic RRT extension*.

One of the challenges for RRT planners is to generate samples in narrow passages of the free space. Moreover, though the basic RRT planner can perform a biased search towards regions not yet visited, such bias does not take into account the obstacles in the environment. Therefore, the basic RRT planner can have difficulty growing out of narrow passages in cluttered environments.

6.4.2 Retraction-based RRT

We use the retraction-based algorithm to improve the performance of RRT planners, especially in narrow passages. Our modified RRT algorithm (Algorithm 3) proceeds as follows:

Algorithm 3 Retraction-based RRT Extension
Input: $T = \{V, E\}$ - an RRT; $\mathbf{q}_r$ - a randomly generated configuration in C-space
Output: T, an extended RRT

```
1: $\mathbf{q_n} \leftarrow$ the nearest neighbor of $\mathbf{q}_r$ in $T$
2: if $\mathbf{q_n}\mathbf{q}_r$ is a collision-free path then
3:     $T.AddVertex(\mathbf{q}_r), T.AddEdge(\mathbf{q_n}, \mathbf{q}_r)$
4: else
5:     // To get a set of non-colliding configurations by retracting $\mathbf{q}_r$
6:     // Using $\mathbf{q_n}$ as the initial guess
7:     $S \leftarrow ConstrainedOptimization(\mathbf{q}_r, \mathbf{q_n})$
8:     for $\mathbf{q_i} \in S$ do
9:         $Basic\ RRT\ Extension(T, \mathbf{q_i})$
10:    end for
11: end if
12: return $T$
```

1. Given the randomly generated configuration $\mathbf{q}_r$, free or colliding, we compute the nearest node $\mathbf{q_n}$ in the tree;

2. Check whether $\mathbf{q}_r$ and $\mathbf{q_n}$ can be connected via a collision-free path. If there is such a path, grow the tree from $\mathbf{q_n}$ to $\mathbf{q}_r$.

3. Otherwise, we retract the sample $\mathbf{q}_r$, as shown in Fig. 6.3-(b):

   (a) Using $\mathbf{q_n}$ as the initial guess, we apply the optimization-based retraction step presented in Section 6.3. The retraction step generates a sequence S of non-collision configurations, approaching the closest boundary point of $\mathbf{q}_r$;

   (b) For every configuration in S, our algorithm performs the basic RRT extension.

We refer to our scheme of growing the tree as *retraction-based extension*. There are several benefits of our enhanced RRT planner using this new extension scheme. First, the sampled configurations that are close to the narrow passages are likely to be retracted into the narrow passages. Consequently, our extended planner generates more samples in narrow passages. Secondly, the tree grows closely towards any randomly generated configuration, significantly improving the bias of the growth of the tree towards regions not yet visited. Finally, we perform the retraction step on free as well as colliding configurations. Overall, our retraction-based RRT can explore or capture the connectivity of narrow passages quickly. We further analyze the behavior of our retraction-based RRT algorithm in Section 6.5 and demonstrate these benefits on many challenging benchmarks in Section 6.7.

6.5 Analysis

In this section, we analyze the behavior of our retraction-based RRT planner. We use the Voronoi diagram defined over the nodes of the RRT in the configuration space to analyze the performance of our enhanced RRT planner. Based on this analysis, we identify the planning scenarios with narrow passages where our retraction-based RRT planner can be quite effective.

6.5.1 Voronoi Diagrams

The behavior of RRT algorithms can be understood using Voronoi diagrams (Kuffner and LaValle, 2000; LaValle, 2006; Yershova et al., 2005). Specifically, the Voronoi diagram for a set of points S in a metric space is the partition of this space which associates a region $V(\mathbf{q})$ with each point $\mathbf{q}$ from S in such a way that all points in $V(\mathbf{q})$ are closer to $\mathbf{q}$ than to any other point in S. Given a tree built by an RRT algorithm, we consider the Voronoi diagram over the set of nodes of the tree (Fig. 6.4) in the configuration space associated with a distance metric δ.

Given a randomly generated configuration $\mathbf{q}_r$, the step of computing the nearest node in the RRT algorithm is equivalent to locating the Voronoi region that contains $\mathbf{q}_r$ (Fig. 6.4). Therefore, the probability of a node $\mathbf{q}$ in the tree being chosen for extension is proportional to the ratio ρ of the volume of its Voronoi region $V(\mathbf{q})$ to the volume of

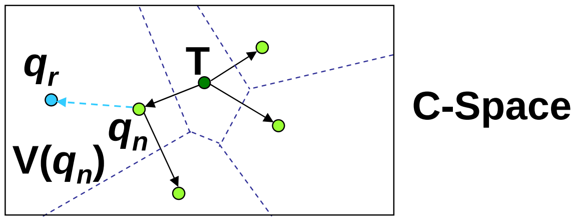


Figure 6.4: Analyzing RRT using Voronoi diagrams. *Given a randomly generated configuration* $\mathbf{q}_r$*, the step of finding its nearest node* $\mathbf{q_n}$ *in the tree* T *for extension is equivalent to locating the Voronoi region that contains* $\mathbf{q}_r$ *(LaValle, 2006).*

the sampling space or the entire configuration space $\mathcal{C}$:

$$\rho(\mathbf{q}) = \frac{Volume(V(\mathbf{q}))}{Volume(\mathcal{C})}. \tag{6.2}$$

We refer to this ratio as *extension ratio* of a node. If a node has a higher value of extension ratio ρ, this node has a higher likelihood of being chosen for extension as compared to other nodes in the tree. Therefore, RRT planners can bias the growth of the tree towards regions not yet visited.

6.5.2 Analysis of Retraction-based RRT

We analyze the behavior of our retraction-based RRT planner (Fig. 6.5). The tree in our planner is biased towards the contact space, and many nodes of the tree are either close to it or in the contact space (e.g. $\mathbf{q_{n1}}$ in Fig. 6.5-a). This is due to the retraction algorithm, which iteratively optimizes over the contact space. We classify the nodes in the tree, near or in the contact space, according to whether a node is:

- Type 1: far away from any narrow passage (e.g. $\mathbf{q_{n0}}$ in Fig. 6.5-(a)),
- Type 2: lying in a narrow passage, or
- Type 3: close to the entrance of a narrow passage (e.g. q_{n1} in Fig. 6.5-(a)).

Among these nodes, the nodes of Type 2 or Type 3 are important for planning as they are associated with narrow passages. Our retraction algorithm utilizes them in a manner

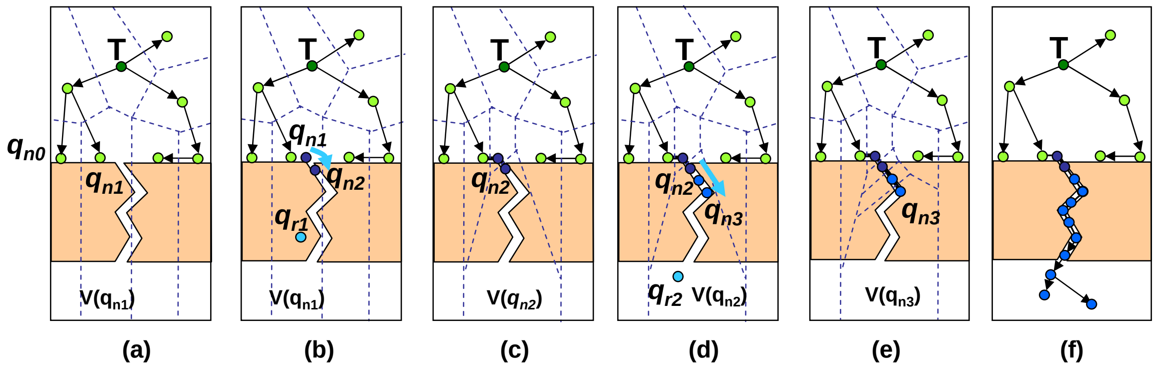


Figure 6.5: Analysis of the retraction-based RRT planner *(a) A tree - T needs to grow through a zigzag narrow passage. The Voronoi diagram is defined over the nodes of the tree in the configuration space. The Voronoi region of the node* $\mathbf{q}_{n_1}$ *is denoted as* $V(\mathbf{q}_{n_1})$. *(b) For a randomly generated colliding configuration* $\mathbf{q}_{r_1}$, *our RRT planner uses its nearest neighbor* $\mathbf{q}_{n_1}$ *as the initial guess and performs the optimization-based retraction. Since* $\mathbf{q}_{n_1}$ *is close to the entrance of the narrow passage, it is very likely that the optimization algorithm generates configurations, e.g.* $\mathbf{q}_{n_2}$, *in the narrow passage. (c) Though the node* $\mathbf{q}_{n_2}$ *in the tree T lies in the narrow passage, it has a high value of extension ratio* ρ *due to the large volume of its Voronoi region* $V(\mathbf{q}_{n_2})$. *Therefore, the node* $\mathbf{q}_{n_2}$ *has a high likelihood of being chosen for extension. (d) Given a randomly generated collision-free configuration* $\mathbf{q}_{r_2}$, *the node* $\mathbf{q}_{n_2}$ *is chosen for extension and more samples in the narrow passage are generated. (e) With a high value of the extension ratio, the node* $\mathbf{q}_{n_3}$ *in the narrow passage has a high likelihood of being chosen for extension. Therefore, more nodes in the narrow passage can be generated. (f) Finally, the tree grows through the narrow passage.*

such that many samples are generated in close proximity of Type 2 and Type 3 nodes or in the associated narrow passages. This is due to our retraction computation, and Fig. 6.5-(b) shows such an example. For a randomly generated sample $\mathbf{q}_{r_1}$, its nearest neighbor $\mathbf{q}_{n_1}$ with Type 3 is chosen for extension. Stating from $\mathbf{q}_{n_1}$, our retraction algorithm iteratively optimizes over the contact space and generates more samples towards or along narrow passages.

A key factor that governs the effectiveness of our retraction-based RRT planner is the probability with which the nodes of Type 2 and Type 3 are chosen for RRT extension. In general, the performance of sample-based approaches degrades in narrow passages since the ratios of the volumes of narrow passages to the volume of the sampling space are typically small. As a result, prior randomized sampling methods may not compute sufficient number of samples in narrow passages to find collision-free paths. However, there are many planning scenarios with narrow passages where the extension ratios, ρ, for the tree nodes of Type 2 and Type 3 are much larger as compared to the ratios of the volumes of their associated narrow passages to the volume of the sampling space. Fig. 6.5 illustrates such cases. Our retraction algorithm can generate more samples in the narrow passages and thereby improve the performance of the planner.

6.6 D-Plan: Efficient Collision-Free Path Computation for Part Removal and Disassembly

The simulation of assembly maintainability in virtual prototyping attempts to remove a particular part from an assembly. Similarly, part disassembly simulation boils down to computing collision-free trajectories for objects through tight spaces or narrow passages. Many of these problems reduce to motion planning of robots, where collision-free paths need to be computed for rigid objects with six degrees of freedom (DOF) among stationary obstacles (Chang and Li, 1995; Latombe, 1991; Garber and Lin, 2002; Tesic and Banerjee, 2002; Ferr and Laumond, 2004). However, in terms of narrow passages, these motion planning scenarios are rather challenging since the parts are often tight-fit with each other. Furthermore, the underlying models are complex and may be represented using thousands of polygons. Many times the models are given as polygon soup models with no connectivity or topology information. Such models are increasingly used in virtual prototyping and PLM (product lifecycle management), since many CAD systems import models generated from other sources, and sometimes the translators do not maintain the connectivity information. It is important that motion planners in PLM applications can handle such datasets automatically.

In this section, we present a general and fast motion planning algorithm, D-Plan,

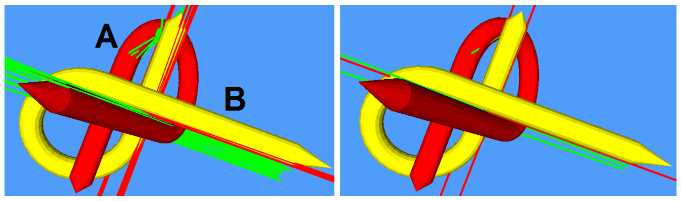


Figure 6.6: Contact query for polygon soup models. *Left: The basic algorithm for contact query is to determine all feature pairs between alpha-shaped models A and B, whose distances are less than a given tolerance. Many feature pairs are computed, while some of them are either duplicate or nearby. Right: We filter the duplicate or nearby pairs and compute the representative ones, i.e. the locally closest feature pairs of between the models. In this example, we obtain three representative pairs of features.*

for part disassembly simulation (Zhang et al., 2008a). D-Plan uses retraction-based sampling to improve the performance in narrow passages. We apply D-Plan to general, complex, polygon soup models. We present techniques to perform efficient contact query among polygon soup models and compute their closest features pairs. We further improve the performance of our planner by performing localized collision detection and exploit the spatial coherence between nearby queries in the configuration space.

6.6.1 Contact Query on Polygon Soup Models

The basic algorithm for performing the contact query can be described as follows. We assume that each object corresponding to the robot or any obstacle is represented as a collection of triangles. We may or may not have any connectivity information among the triangles. Given two polygon soup models A and B, we build a bounding volume hierarchy (BVH) for each of them. By traversing both the BVHs, we can efficiently determine all the pairs of triangles between the two models, whose distance is less than some given tolerance κ_c. For each such triangle pair, we can further obtain the feature pair realizing the closest distance. In general, there are three types of feature pairs: (V,F), a vertex of A and a face of B, (F,V), a face of A and a vertex of B and an edge of A and an edge of B (E,E). We collect every such feature pair whose distance is less than κ_c, as Σ.

There could be many duplicate or nearby feature pairs in Σ, computed by the above basic algorithm, as shown in Fig. 6.6(left). The duplicate feature pairs are generated because a feature in each model may be incident to more than one triangle. For example, for an (E, E) feature pair in Σ, if there are two triangles sharing an edge for each model, this feature pair is reported four times. Furthermore, the hierarchical algorithm described above will report many nearby feature pairs as long as their distances are smaller than the given tolerance κ_c.

The duplicate or nearby feature pairs need to be filtered from Σ. Otherwise, many unnecessary contact constraints are constructed, which can result in many extra samples during our sampling stage. In general, it is difficult to filter out duplicate or nearby feature pairs, since we do not have the connectivity information of the models. We use a simple heuristic for the purpose of filtering and computing the representative ones, i.e. the locally closest feature pairs between A and B. For each feature pair computed by the basic algorithm, we compute their witnessing points p_a and p_b in those features, i.e. the points which witness the closest distance between the two features. In order to determine whether a feature pair is duplicate or representative, we compare its witnessing points p_a and p_b against witnessing points p'_a and p'_b of every known representative pair. If the distance between p_a and p'_a and the distance between p_b and p'_b are both greater than the tolerance κ_c, the pair is declared as a new representative pair.

6.6.2 Localized Collision Detection

The main issue with our retraction-based sampling method is how to efficiently search over the local contact space. Our method randomly generates many samples on the local contact space and performs the optimization step. In this case, the collision detection routine is invoked for every generated sample to check whether it is in free space. Given a highly cluttered environment with narrow passages, typically many generated samples

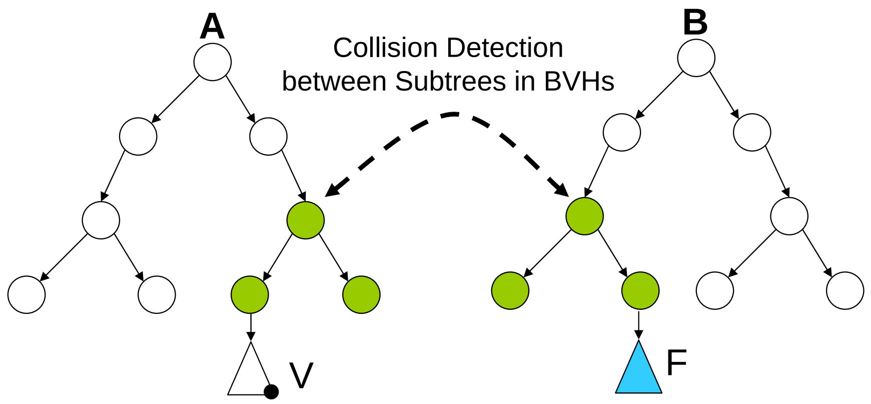


Figure 6.7: Localized collision detection. *Our algorithm can quickly cull colliding samples generated near the contact space and in narrow passages. The culling is achieved by checking the collision among subtrees of BVHs. We determine such subtrees using the closest features from the latest contact query (a (V,F) feature pair in this example).*

would be colliding. When a sample is likely to lie in the C-obstacle space rather than free space, BVH-based collision detection algorithms often perform poorly as they traverse the hierarchy all the way to the leaf nodes. As a result, the resulting sampling method spends a significant fraction of the overall running in collision checking and traversing the BVHs.

We use a simple localized approach to accelerate the sample generation and collision checking. Our algorithm can conservatively cull away the samples lying in C-obstacle space by efficiently exploiting spatial coherence between nearby queries. During each retraction step, our localized collision detection algorithm performs as follows (Fig. 6.7):

1. Use the feature pairs reported by the latest contact query, and locate their corresponding triangles as well as the corresponding bounding volumes (BVs) in BVHs of the robot A and the obstacles B;

2. Compute the subtree within each BVH, which contains the located BVs;

3. Perform collision detection among the subtrees of BVHs for A and B;

4. If a collision is reported, we can quickly declare that the sample is colliding;

5. Otherwise, detect collisions using the entire BVHs.

Our localized approach can considerably accelerate the collision detection when sampling near the contact space and in narrow passages. The size of each subtree is much smaller than the entire BVH. However, our localized approach is conservative. If a sample could not be identified as colliding in Step 3, the algorithm moves to Step 5, and performs collision detection by traversing from the root of the BVH from scratch.

Essentially, our localized collision detection approach exploits spatial coherence that exists within our retraction-based sampling algorithm. When planning near the contact space or in narrow passages, our sampling algorithm performs collision queries for nearby samples in the configuration space. Our localized approach exploits this spatial coherence by making use of the subtrees of BVHs for direct traversal. Therefore, our approach is able to quickly cull away many colliding samples. Finally, our localized approach can also improve the performance of local planning methods, where collision detection is performed on a finite number of samples on the interpolated path.

6.7 Implementation and Results

In this section, we present experimental results of our retraction-based RRT planner (RRRT) on 3D rigid robots and our D-Plan approach for part disassembly simulation. We first address some implementation issues. Next, we highlight the performance of our RRRT planner on a set of examples. In each example, a rigid robot needs to plan through some narrow passages in the 3D environment. We also test the performance of D-Plan on difficult part removal and disassembly problems. These include the well-known Alpha puzzle benchmark (Amato et al., 1998), maintainability of the windscreen wiper motion with $15K$ triangles for the robot and $11K$ triangles for the obstacle, and disassembly of a seat ($30K$ triangles) outside a car body ($214K$) described in (Ferr and Laumond, 2004).

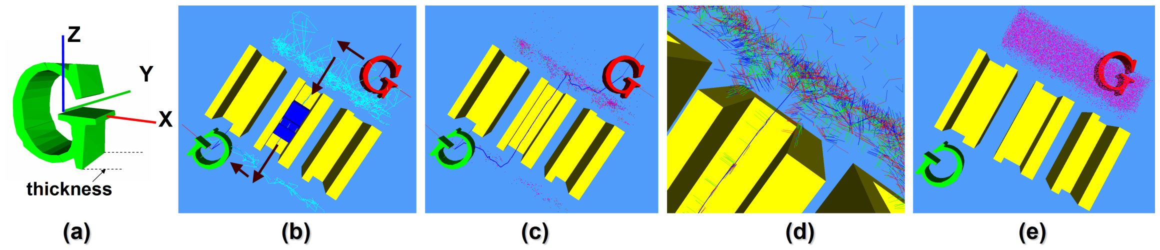


Figure 6.8: ‘Notch’ example. *(a) A G-shaped robot. (b) The environment is consisted of three notch-shaped obstacles, and the robot needs to move from one side to the other. Due to the narrow passages, the basic RRT planner was unable to find a solution after* $500,000$ *iterations within* $1,232.2s$*. On the other hand, RRRT can find a collision-free path within* $25.4s$*. In (b), (c), the collision-free path, the tree and its nodes are highlighted after being projected from the 6D C-space into 3D Euclidean. Many nodes are biased towards the contact space as well as the narrow passage. (d) is a magnified version of the result. Here, a node in the tree is visualized by a 3D point to indicate the position of the robot, together with a 3-dimensional orthogonal frame to indicate the orientation. (e) shows the nodes generated by the basic RRT planner, where no node lies in the narrow passages.*

6.7.1 Implementation

We have implemented RRRT on 3D rigid robots. Our implementation consists of two parts: the implementation of the retraction step and the integration with an RRT planner. In the first part, we use the DISP distance metric defined in SE(3) and extend the generalized PD algorithm using constrained optimization in in Section 4.2 to perform the retraction step. We set the maximum iteration in each retraction step as 5. We use PQP (Larsen et al., 1999) for collision detection. We implement a basic RRT planner. For simplicity, we perform the local planning by using a linear interpolation motion and checking for collisions on a finite number of intermediate configurations of the motion.

We integrate our retraction algorithm into the basic RRT planner. During each retraction step, a sequence of configurations, close to or in the contact space, are generated. Our planner RRRT then attempts to extend the tree to each configuration using the basic RRT extension scheme. In our implementation, we observe the difficulty of connecting two nearby samples when both of them are close to the contact space. Currently we use an enhanced local planning scheme - *vertex enhancement* that can generate

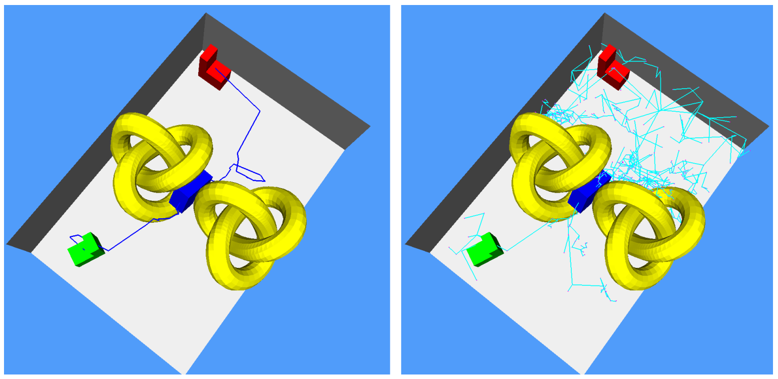


Figure 6.9: 'Torus' example. *Left: A L-shaped robot needs to move from one side (red) in the environment, consisting of two torus knot shaped obstacles, to the other (green). Our planner can compute a collision-free path within* 44.9*s. Right: the computed tree is highlighted.*

additional samples around them (Kavraki et al., 1996). To deal with this issue, other local planning schemes can also be employed (Zhang and Manocha, 2008a).

Our D-Plan approach is based on retraction-based RRT planner. We implement the contact query on general polygon soup models. This enables our D-Plan to automatically handle any type of CAD models. By extending PQP (Larsen et al., 1999), our contact query uses hierarchies of swept sphere volumes of given models to efficiently determine those feature pairs (V,F), (F,V), or (E/E), whose distances are less than the tolerance κ_c. In our experiment, this tolerance is simply set as the radius of the smallest enclosing sphere of the robot multiplying by 0.01. We remove the duplicate pairs and further identify locally closest pairs, using the tolerance $\kappa_r = 10\kappa_c$. Though both parameters are chosen heuristically, they work well on our benchmarks. Fig. 6.6 depicts the result of contact query between two alpha-shaped models.

We use localized collision detection to improve the performance of D-Plan. Based on PQP, we find the leaf nodes in BVHs, which correspond to the latest contact query. We determine the subtrees containing the leaf nodes in each BVH. Currently, the depth of each subtree is chosen as 6, though it may be worthwhile choosing the depth according to the complexity of the model. We use the subtrees to localize the computation of collision checking.

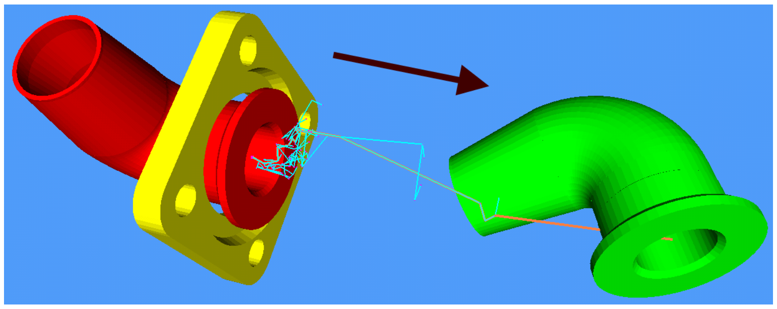


Figure 6.10: 'Flange' example. *The CAD model - 'elbow' needs to slide out of the hole of the CAD model - 'flange'. Our planner takes* 25.0*s for this task, while a variant of RRT presented in (Rodriguez et al., 2006) takes* 227.1*s on a similar PC.*

6.7.2 Results

We test our retraction-based RRT planner (RRRT) on a set of benchmarks. In our experiment, we run every benchmark 10 times and compute the average running time. The timing is summarized in Table 6.1. The geometric complexity of the benchmarks is highlighted in Table 6.2. RRRT can handle general polygonal models, including polygon soup models. The timings reported for *RRRT* planner were taken on a 2.8GHz Pentium IV PC with 2GB of memory.

In the 'notch' example (Fig. 6.8), there are three narrow passages since the width of the corridor within each notch-shaped model is 1 and it is slightly larger than the 'thickness' of the g-shaped robot, 0.95. The environment also possesses an interesting property. The widths of the two gaps formed by the three notch-shaped models are 0.9, resulting in two potentially false passages. Therefore, dilation-based planners may not work well on this benchmark. In our experiment, the basic RRT planner was unable to find a solution within $1,232.2$s. On the other hand, our planner can find a collision-free path within 25.4s. Figs. 6.9 and 6.10 show two additional examples where the models are more complex. Our planner can compute a collision-free path through a narrow passage within 44.9s and 25.0s, respectively.

Table 6.3 highlights two ways to break down the running time for RRRT. One way is to measure the $t_{retraction}$, the time on the retraction step and $t_{linking}$, the time on

	Notch	Torus	Flange
RRRT: t_{all} (s)	25.4	44.9	25.0
RRRT: nodes	1,401	1,471	119
Basic RRT: t_{all} (s)	$>$ 1,232.2$\star$	4,920.9	680.1
Basic RRT: nodes	$>$ 105,987$\star$	43,512	95

Table 6.1: Performance of our retraction-based RRT. *The table compares the performance of our planner - RRRT with the basic RRT planner on different benchmarks. The table includes the planning time t_{all} and the number of nodes in the resulting tree. $\star$: the basic RRT planner cannot find a path within a large mount of time.*

	Notch	Torus	Flange
# Tri of robot	28	20	3,525
# Tri of obstacles	756	5,760	5,306
# of obstacles	3	2	1

Table 6.2: Model complexity. *The table summarizes the geometric complexity of each benchmark.*

	Notch	Torus	Flange
t_{all}(s)	25.4	44.9	25.0
$t_{retraction}$(s)	11.0	22.1	16.1
$\#_{retraction}$	625	1,203	232
t_{per_retra}(ms)	17.600	18.371	69.397
$t_{retraction}/t_{all}$	43.3%	49.2%	64.4%
$t_{linking}$ (s)	14.2	18.9	6.6
$\#_{linking}$	1,587	2,920	303
$t_{per\_linking}$ (ms)	8.948	6.473	21.782
$t_{linking}/t_{all}$	55.9%	42.1%	26.4%
t_{cd}(s)	18.5	32.1	19.3
$\#_{cd}$	91,958	133,580	20,794
t_{per_cd}(ms)	0.201	0.240	0.928
t_{cd}/t_{all}	72.8%	71.5%	77.0%
$t_{contact}$(s)	0.9	2.0	2.9
$\#_{contact}$	1,548	2,831	252
$t_{per\_contact}$(ms)	0.581	0.706	11.508
$t_{contact}/t_{all}$	3.4%	4.4%	11.5%

Table 6.3: Breakdown of running time. *The table summarizes two ways to break down the running time for main functions in RRRT. One way is to measure $t_{retraction}$, the time on the retraction step and $t_{linking}$, the time on connecting the generated samples. Another way is measure t_{cd}, the time on collision detection and $t_{contact}$, the time on contact query. $\#_{retraction}$ and t_{per_retra} denote the number of retraction steps and the average time of each retraction step, respectively.*

connecting the samples. The other way is to measure t_{cd} the time on collision detection and $t_{contact}$, the time on contact query. Overall, the function for collision detection takes around 70% to 80% of the total time.

We apply D-Plan to four challenging benchmarks. The timing of these benchmarks was taken on these benchmarks was on a PC with a 4-core Xeon 3GHz CPU with 4GB of memory. The geometric complexity of them and the performance of our D-Plan approach are summarized in Tables 6.5 and 6.6.

- Alpha Puzzle - Fig. 6.11: A well known benchmark with narrow passages for testing the performance of motion planning approaches (Amato et al., 1998);
- Pipe - Fig. 6.12: Maintainability test in the CAD model of a complex machinery room. We check how to remove a pipe-shaped robot from the machinery room without any collision;
- Wiper - Fig. 6.13: Maintainability test of the windscreen wiper motion (Ferr and Laumond, 2004). This is an industrial benchmark with narrow passages;
- Car Seat - Fig. 6.14: Disassembly of a seat outside a car body (Ferr and Laumond, 2004). This is an industrial benchmark with complex geometric representation. For simplicity, in our local planning algorithm, we perform collision detection by using a finite number of samples on the interpolating motion, e.g. 15, 50, 20, and 30 samples for alpha puzzle, pipe, wiper, and car seat benchmark, respectively.

6.7.3 Comparison

We compare the performance of our retraction-based RRT planner with other RRT planners. We first compare with the basic RRT planner, which uses the *basic RRT extension*. Table 6.1 shows for all four benchmarks, RRRT is much more efficient than the basic RRT planner. We also test both planners on different versions of the 'notch'

Robot Scale	0.80	0.85	0.90	0.91	0.92	0.93	0.94	0.95
Basic RRT: t_{all}(s)	11.6	31.2	120.8	143.4	184.7	292.7	306.3	$>$ 1,232.2 $\star$
RRRT: t_{all}(s)	6.6	7.0	9.4	14.0	8.1	26.1	23.6	25.4
Basic RRT: nodes	4,226	8,843	22,755	23,085	28,417	43,200	42,530	$>$ 105,987 $\star$
RRRT: nodes	398	467	524	786	510	1,519	1,460	1,401

Table 6.4: Performance comparison of RRRT and RRT. *The table compares the performance of retraction-based RRT with the basic RRT planner on the 'notch' example. The robot is scaled from* 0.8 *to* 0.95. *Our planner significantly improves the performance on each version of the problem.* $\star$ *denotes the most difficult version -* 0.95. *The basic RRT planner cannot solve it after running* 500,000 *iterations within* 1.232.2*s, while our RRRT can compute a path within* 25.4*s.*

example, i.e. scaling the G-shaped robot from 0.8 to 0.95. Table 6.4 shows that RRRT significantly improves the performance for each version of the problem. Even for relatively less open scenarios (e.g. scale = 0.8), our planner is still more efficient than basic RRT planner. Figs. 6.8-(c),(e) compare the distribution of the nodes generated by RRRT and the basic RRT planners for the 0.95 version. The RRRT planner generates more samples in the contact space and narrow passages, while the basic RRT cannot generate samples in narrow passages.

There are variants of RRT-based planners such as (Strandberg, 2004; Rodriguez et al., 2006) to improve the performance on narrow passages. We quantitatively compare our planner with the RRT-based planner presented in (Rodriguez et al., 2006) by using the 'flange' example(Fig. 6.10). Our planner takes 25.0s for this task and is much more efficient than the planner in (Rodriguez et al., 2006), which takes 227.1s on a similar PC. Compared with another RRT planner in (Strandberg, 2004), one difference is that we perform the retraction on both colliding as well as collision-free configurations, while their method can only bias the growth of the tree using the collision-free configurations. Finally, another RRT variant (Yershova et al., 2005) also takes into account C-obstacle into the RRT bias as ours. However, their work mainly characterizes the issue when the sampling domain is not well adapted to the problem, while our method focuses on improving the performance in narrow passages.

Our planner is efficient as compared to other retraction-based methods (Ferr and

Benchmarks	Alpha Puzzle	Pipe	Wiper	Car Seat
A	Alpha	Pipe	Wiper	Seat
B	Alpha	Machine Room	Windscreen	Car Body
# Tri: A	1,044	10,352	15,197	30,790
# Tri: B	1,044	38,146	11,569	214,337

Table 6.5: Model complexity of part disassembly benchmarks. *Many benchmarks we use have no connectivity information. The moving objects correspond to robots (row A) and the static obstacles are shown in row B.*

Benchmarks	Alpha Puzzle	Pipe	Wiper	Car Seat
RRRT: timing (s)	1043.0	124.7	1197.8	181.2
# samples	53,535	2,539	8,890	1,352
Basic RRT: timing (s)	>119,668.4	1,444.0	>12,011.3	311.0
# samples	>84,847	4,345	>39,962	3,230

Table 6.6: Performance of D-Plan. *For all these difficult benchmarks, our planner is able to compute a collision-free motion in less than 20 minutes. The basic RRT planner, however, can not solve the alpha puzzle benchmark within 100 times of our planning running time. For the rest of benchmarks, D-Plan is also significantly faster than the basic RRT planner.*

Laumond, 2004; Saha et al., 2005; Cheng et al., 2006). Our D-Plan approach takes less than 20 minutes to compute a collision-free path for all these difficulty benchmarks. Furthermore, for the alpha puzzle benchmark, D-Plan can find a collision-free path within $1,043.0$ seconds, while the recent retraction-based planner takes $5,850$ seconds on a relative slower machine (Cheng et al., 2006), and the basic RRT planner based on our implementation can not find a path within $119,668.4$ seconds. Except for the wiper benchmark, the performance of our D-Plan approach is competitive as compared to the algorithm in KineoCAM software (Ferr and Laumond, 2004).

Also, our planner has many distinct features. The small-step retraction-based methods (Saha et al., 2005; Cheng et al., 2006) identify the colliding configurations near the free space by shrinking the models of the robot and the obstacles. These methods are only applicable to closed models. Moreover, it is difficult to perform the shrinking step

		Alpha Puzzle	Pipe	Wiper	Seat
Contact Query	$t_{contact}$ (s)	128.2	7.1	344.8	21.4
	$\#_{contact}$	82,309	4,628	21,583	1,597
	$t_{per\_contact}$ (ms)	1.6	1.5	16.0	13.4
	$t_{contact}/t_{all}$	12.3%	5.6%	28.8%	11.8%
Localized Collision Detection	t_{lcd} (s)	181.6	4.7	33.2	2.4
	$\#_{lcd}$	5,854,345	304,774	1,445,944	86,785
	t_{per_lcd} (ms)	0.031	0.016	0.063	0.028
	t_{lcd}/t_{all}	17.4%	3.7%	2.8%	1.3%
	Culling Ratio	48.8%	18.82%	36.5%	32.6%
	t_{per_lcd}/t_{per_cd}	13.3%	3.6%	9.5%	1.3%
Collision Detection	t_{cd} (s)	698.1	111.2	608.9	126.0
	$\#_{cd}$	2,995,802	247,419	917,818	58,497
	t_{per_cd} (ms)	0.233	0.449	0.663	2.1
	t_{cd}/t_{all}	66.9%	88.3%	50.1%	69.5%
Planner	t_{all} (s)	1043.0	125.9	1197.8	181.2

Table 6.7: Breakdown of running time of D-Plan. *The table highlights the timing breakdown for D-Plan. tcon, #con and tper_con are the total timing, the number, and the timing on average for the contact query; tlcd and tcd are the total timing for localized collision detection and collision detection, respectively. Overall, the module for collision detection takes around* 50.1% *to* 88.3% *of the total timing (tcd/tall), the module for contact query takes around* 5.6% *to* 28.8%, *and the module for localized collision detection accounts for* 1.3% *to* 17.4% *of total running time. In order to evaluate the effectiveness of our localized collision detection, we measure the culling ratio #lcd/#cd, the number of global collision detection queries over the number of the localized collision detection queries. According to the table, our method can achieve* 18.8% *to* 48.8% *culling ratios on the tested benchmarks. Furthermore, tper_lcd/tper_cd, the ratio of the timing for localized collision detection over the average time for collision detection is around* 1.3% *to* 13.3%. *These two ratios indicate that our localized collision detection algorithm effectively exploits the spatial coherence and considerably improves the planner's performance.*

on complex models, and the topology of the dilated free space may be different. On the other hand, our algorithm is directly applied to general polygonal models. Furthermore, based on the shrinking of geometric models, these methods implicitly use the formulation of *growth distance* for quantifying the amount of intersection among the models (Ong and Gilbert, 1996). This formulation is not as rigorous as our underlying formulation of generalized penetration depth computation, which is based on a proper distance metric that meaningfully combines the translational and rotational motion of the robot as we discuss in Chapter 2.

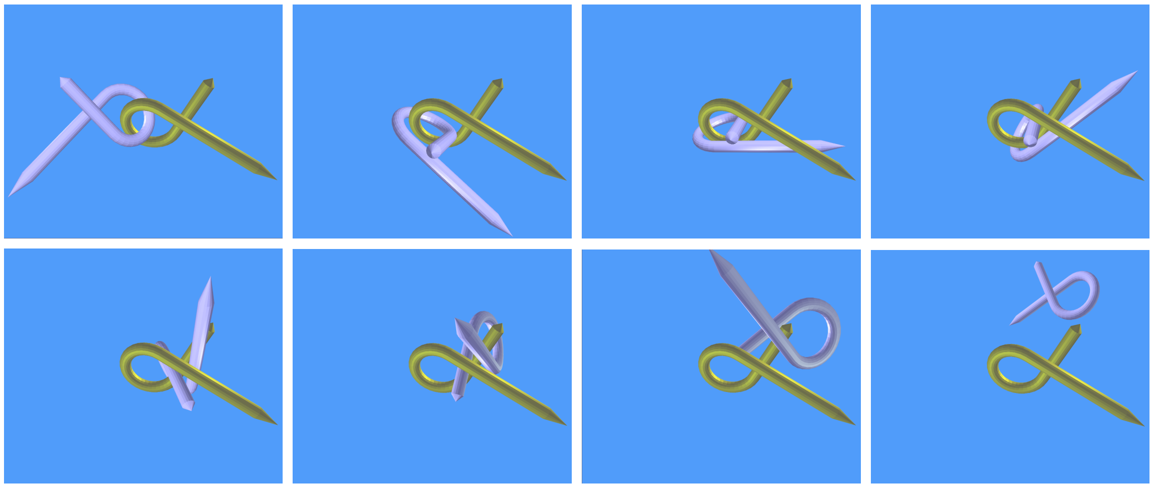


Figure 6.11: Alpha puzzle benchmark. *the sequence of images shows a collision-free path computed by D-Plan. For this challenge benchmark, D-Plan takes* $1,043.0s$.

6.8 Limitations

There are several limitations of our retraction-based planner. Our optimization-based retraction searches over the contact space and computes a local minima. As a result, it can generate many configurations that lie in the contact space but not in the narrow passages. This can affect the overall performance of the planner. Furthermore, the optimization-based retraction step has additional overhead. If the configuration has no narrow passages, our enhanced planner may take longer time as compared to the basic planner. However, in the notch benchmark where the robot is scaled down to 0.8 so that the narrow passages become wider, our planner still performs as well as the basic RRT. Finally, our algorithm is restricted to rigid models, and performing the retraction step on articulated models can be more expensive.

In terms of our D-Plan approach for part disassembly simulation, there are also a few limitations. Like most previous motion planning approaches, our local planning algorithm performs discrete collision detection along a finite number of samples on the interpolating motion. If the chosen resolution is not high enough, a local path reported as collision-free may not be correct. Furthermore, our method needs to perform the

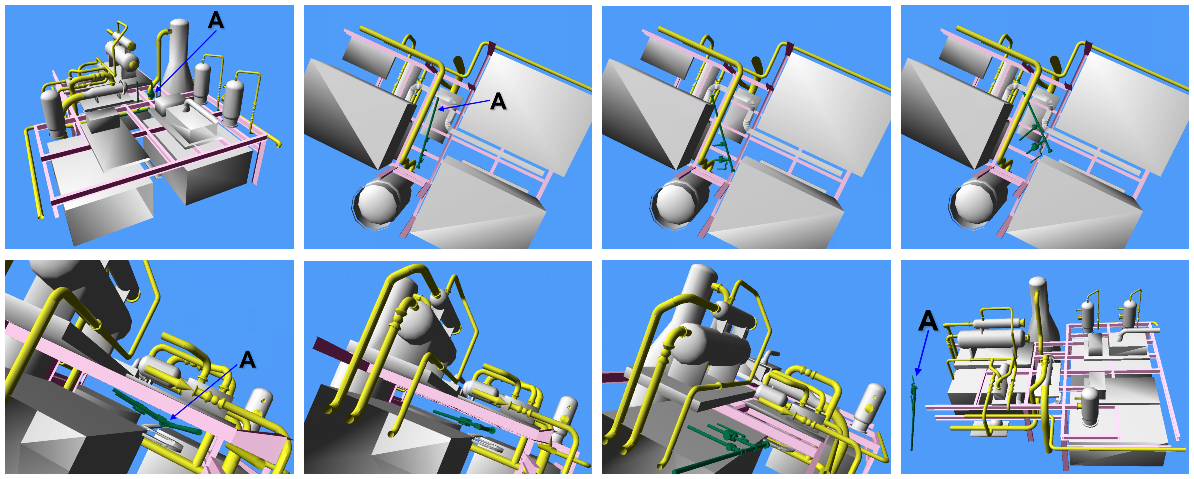


Figure 6.12: Maintainability test of a pipe motion. *A pipe model (highlighted as A) needs to be taken out of the machinery room. The sequence of images shows a path automatically generated by D-Plan. It only takes around 2 minutes to compute the collision-free path.*

contact query repeatedly and use it for sample generation as part of the retraction step. This may impact the overall performance of the planner on complex models. Finally, D-Plan is only able to handle rigid objects.

6.9 Summary

In this chapter, we present an optimization-based retraction algorithm to improve the performance of RRT planners by retracting the samples so that they can be more likely to be connected to the tree. The resulting tree can grow closely towards every randomly generated sample, including colliding as well as free configurations. We analyze the behavior of our planner using Voronoi diagrams of the configurations in the tree and highlight the scenarios where our planner can handle narrow passages well. We have implemented this algorithm and applied it for rigid robots in challenging planning scenarios. Our experimental results show that our algorithm generates more samples near the contact space or in the narrow passages than prior RRT planners and is able to explore more difficult regions in the configuration space. In practice, we observe

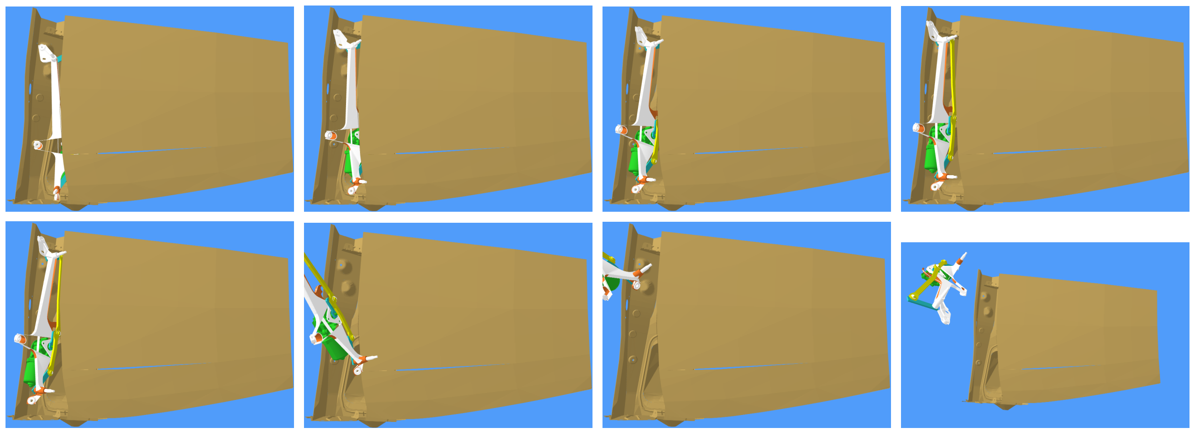


Figure 6.13: Maintainability of the windscreen wiper motion. *The first image shows the input configuration for the benchmark, where the wiper is inside the windscreen model with very tight space. The wiper needs to be taken outside with the final configuration shown in the last image. The intermediate sequence of images shows a path automatically computed by D-Plan.*

significant speedups over prior RRT planners.

We also present a general and efficient motion planning approach, D-Plan, for part removal and disassembly simulation. Our algorithm is based on sample-based motion planning and we use new optimization-based technique to generate samples near the boundary of C-obstacle space and narrow passages. Furthermore, we utilize a constrained interpolation scheme for local planning that connects nearby samples with a higher probability of computing collision-free paths. We also present efficient techniques for performing contact query among general polygon soup models and accelerate the performance based on localized collision detection. D-Plan is able to handle all CAD models with no assumptions on their connectivity or topology. We demonstrate its application to many challenging CAD scenarios.

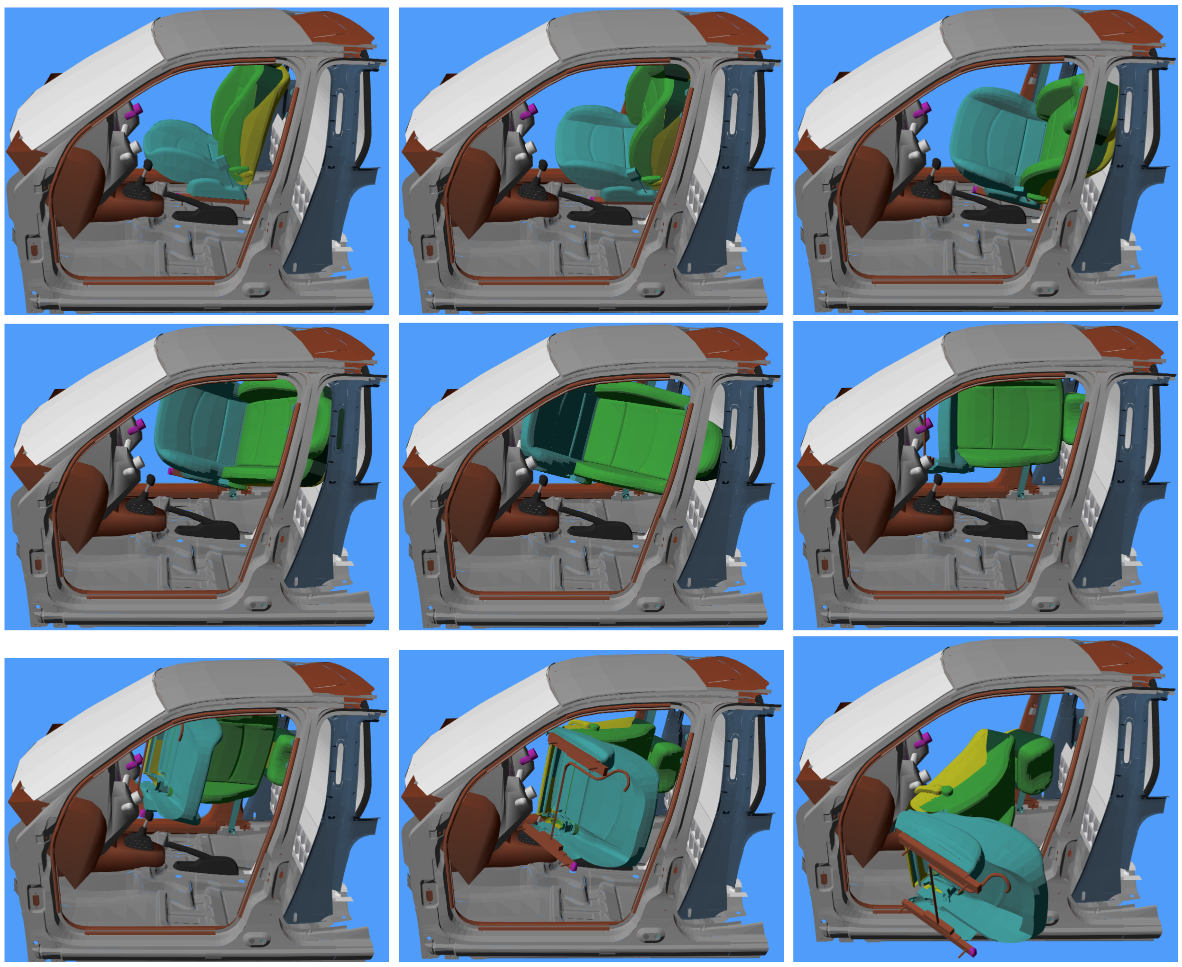


Figure 6.14: Disassembly of a seat outside a car body. *This benchmark of disassembly of a seat outside of a car body also arises from an industrial application (Ferr and Laumond, 2004). It is a difficult scenario for motion planning approaches due to the cluttered environment and complex geometric representation of the models, i.e.* $30K$ *triangles for the seat model and* $214K$ *triangles for the car body model. The sequence of images shows a path automatically generated by D-Plan in about* 3 *minutes.*

Chapter 7

Conclusion and Future Works

In this thesis, our main focus has been on design of efficient and practical motion planning approaches. We mainly address two important problems: (1) develop practical approaches to check for path non-existence for low (up to 4) DOF robots; (2) improve the performance of sampling-based planners when the free space of a robot has narrow passages.

We show that the generalized penetration depth is an important proximity query and can be useful for both the problems. In practice, collision detection algorithms can only determines whether a configuration lies in the robot's free space or not. By quantifying the extent of intersection between the overlapping models (e.g. a robot and an obstacle), penetration depth can be used as a distance measure about the C-obstacle space of the robot. In order to handle 6-DOF free-flying rigid robots, we extend the prior notion of translational PD to generalized PD, which takes into account both the translational and rotational motion. We formulate the generalized PD computation using model-dependent C-space distance metrics and present practical algorithms to compute generalized PD.

We use our generalized penetration depth computation algorithms to design practical motion planning approaches: which are either complete for general low DOF robots, or are efficient in terms of solving difficult planning problems for rigid robots with narrow passages. We use our approach for part removal or disassembly problems in virtual prototyping and CAD.

In the following sections, we summarize our algorithms, discuss some of their limitations and suggest a few directions for future investigation.

7.1 Generalized PD Computation

We address the generalized PD problem between rigid models by taking into account translational as well as rotational motion. We formulate the problem by choosing an appropriate C-space distance metric, such as DISP, OBNO or TRAJ. We present an efficient algorithm to compute the DISP metric for rigid or articulated robots.

We present two new algorithms to compute generalized PD. We first present a convexity-based algorithm using convex decomposition and containment optimization. For two overlapping convex polytopes, we show that generalized PD is equivalent to translational PD. when the complement of one of the objects is convex, we pose the generalized PD computation as a variant of the convex containment problem and compute an upper bound using optimization techniques. When both of the objects are non-convex, we treat them as a combination of the two cases described above and present algorithms that compute a lower bound and an upper bound on generalized PD. Our empirical results show that we can efficiently compute generalized PD for these cases.

We also present a practical algorithm to compute generalized PD for non-convex models based on constrained optimization. Our algorithm performs iterative optimization in the contact space. The experimental results show that we can efficiently compute an upper bound of generalized PD for complex non-convex models composed of a few thousands of trainless. As compared to the convexity-based algorithm, the constrained optimization algorithm offers the following benefits:

- **Generality:** Our algorithm is general and applicable to both convex and non-convex rigid models. The approach makes no assumption about object topology and doesn't need to perform convex decomposition. The approach can also be extended for polygon soup models.

- **Practicality:** Our algorithm is relatively simple to implement and useful for many applications that require both translation and rotation measures for the intersection.

- **Efficiency:** Our algorithm use a local optimization algorithm and reduce the problem of generalized PD computation to performing multiple collision detection and contact queries.

We demonstrate the efficiency and the robustness of both generalized PD algorithms on many complex 3D models.

7.1.1 Limitations

One limitation of our convexity-based generalized PD algorithm is that it requires convex decomposition as a preprocess. Such computation is inefficient for complex objects or difficult on models without the connectivity information. Given the complexity of exact generalized PD computation for non-convex polyhedra, we only compute lower and upper bounds. These bounds depend on the convex decomposition of the models and may not tight. However, in most practical cases, the extent of penetration is small and we expect that our algorithm would compute a good approximation.

The main limitation of our algorithm based on constrained optimization is that the algorithm only computes a local minimum and can not guarantee a global solution for generalized PD computation. Its performance depends on the choice of an initial guess.

7.2 Efficient Motion Planning

We present efficient complete motion planning approaches using generalized PD computation. In order to check for non-existence, we use the C-obstacle query to efficiently check whether a cell in C-space lies entirely inside the C-obstacle region. We describe simple and efficient algorithms for C-obstacle using generalized PD computation. Our query algorithm is applicable to 2D or 3D rigid robots, or articulated robots. We further present an efficient hybrid algorithm that combines the efficiency of randomized sampling methods with the completeness of approximate cell decomposition algorithms. We also extend our complete motion planning approach to feedback motion planning by computing a global vector field in the entire free space of the robot. As compared to prior approaches, our planner and its variants are complete and relatively simple to implement. The resulting planners also work robustly on non-convex robots and obstacles. We demonstrate their performance on 3-4 DOF robots.

We present an optimization-based retraction algorithm to improve the performance of sampling-based planners in narrow passages for 3D rigid robots. Based on generalized PD computation, our algorithm retracts samples to the nearby boundary of the C-obstacle space. Our algorithm is general and makes no assumption about model connectivity or object topology. Based on the retraction-based sampling, we present D-Plan approach for part disassembly tasks, which frequently arise in CAD/CAM and virtual prototyping and have tight-fitting configurations. We highlight the performance of D-Plan on many challenging benchmarks with narrow passages (e.g. the alpha puzzle benchmark).

7.2.1 Limitations

Our complete motion planning approaches have a few limitations. The C-obstacle cell query algorithm is conservative, which stems from the conservativeness of generalized

PD and bounding motion computations. Secondly, our path non-existence algorithm assumes that there are no tangential contacts on the boundary of the free space. Otherwise, the algorithm may not terminate. As a result, our algorithm can not deal with compliant motion planning, where a robot cannot pass through obstacles without touching the obstacles. The complexity of our algorithms based on approximate cell decomposition varies as a function of the dimension of the configuration space. However, our experimental results show that our algorithms works well on many path non-existence scenarios. Finally, when we apply our planners to 4-DOF or higher DOF problems, graph searching along with the cell decomposition becomes one of the major bottlenecks. This is because the size of the connectivity graph G increases as a function of the number of the cells in ACD.

For global vector field computation method, the resulting integral curves are guaranteed to be C^{∞} smooth. However, the curves can have sharp turns due to the underlying adaptive decomposition.

7.3 Future Work

We have presented new algorithms for efficient motion planning using generalized PD computation. In practice, we have been able to solve some of the planning benchmarks that were considered as challenging for prior techniques. But, there are a number of remaining challenges in the areas. This includes improving generalized PD computation for rigid robots and articulated robots, extending our complete motion planning approaches for higher DOF problems, such as 6-DOF rigid robots and incorporating differential constraints into our geometric path planning approaches.

Generalize PD computation: It would be useful to derive tight bounds on the approximations (i.e. the lower and upper bounds) and analyze the convergence properties of our generalized PD algorithms. A tighter lower bound can improve the accuracy of C-obstacle cell query and therefore improve the overall performance of path non-existence computation. For retraction-based planners, a tighter upper bound on generalized PD can result in more effective retraction computation.

Articulated robots such as manipulators have been widely used. In order to improve the efficiency of motion planning for an articulated robot in a constrained environment, we need to reason about its C-obstacle space more effectively. Generalized PD computation can provide a distance measure to C-obstacle space. However, the straightforward way by treating each link of the robot as a rigid body can be overly conservative for computing generalized PD. In order to compute tight bounds, we may make use of the robot's kinematic structure and design efficient schemes for local contact space search.

Complete motion planning for 6-DOF rigid robots: Our C-obstacle query algorithm is directly applicable to 3D rigid robots or high DOF robot. The implementation of ACD is also general for any dimensional C-space. The main challenge is to reduce the number of cells from spatial decomposition for higher dimensional problems. Currently, one of the performance bottlenecks in our implementation is the graph searching step for each level of guided subdivision. This may be improved by using incremental graph search algorithms that can exploit the spatial coherence between different levels.

It is also interesting to investigate sampling-based path non-existence approaches. For any colliding configuration of a n-DOF robot, based on the lower bound on its generalized PD, we can compute a closed n-ball, which lies entirely in the robot's C-obstacle. One sufficient condition for checking for path non-existence is whether the robot's initial and goal configurations are path disconnected with respect to the union of all such balls from randomized sampling. The major difficulty to address is to determine the disconnection while avoiding the expensive step of explicitly computing the union of balls in high dimensional C-space.

Feedback motion planning and dynamic constraints: We are interested in combining our feedback planning algorithm with randomized sampling techniques for higher DOF robots. We are also interested in improving the quality of integral curves over the global vector field. For instance, in order to reduce the high-variation of curvature on the paths, we may compute an optimal vector field over the approximate cell decomposition. Also, we would like to extend our algorithm to robots with non-holonomic and dynamic constraints so that the planner can not only satisfy collision avoidance constraints, but also guarantee the safety of the system in terms of dynamic constraints.

Finally, it may be worth testing the performance of our retraction-based planner on more complex benchmarks and integrating our D-Plan approach into commercial CAD and virtual prototyping systems.

Bibliography

Agarwal, P., Amenta, N., and Sharir, M. (1998). Largest placement of one convex polygon inside another. In *Discrete Comput. Geom*, volume 19, pages 95–104. 24

Agarwal, P., Guibas, L., Har-Peled, S., Rabinovitch, A., and Sharir, M. (2000). Penetration depth of two convex polytopes in 3d. *Nordic J. Computing*, 7:227–240. 23

Amato, N., Bayazit, O., Dale, L., Jones, C., and Vallejo, D. (1998). OBPRM: An obstacle-based prm for 3d workspaces. *Proceedings of WAFR*, pages 197–204. 9, 95, 143, 146, 147, 149, 161, 166

Amato, N., Bayazit, O., Dale, L., Jones, C., and Vallejo, D. (2000). Choosing good distance metrics and local planners for probabilistic roadmap methods. *IEEE Trans. Robot. and Autom.*, 16(4):442–447. 27

Avnaim, F. and Boissonnat, J.-D. (1989a). Polygon placement under translation and rotation. *RAIRO Inform. Theor.*, 23:5–28. 24, 59

Avnaim, F. and Boissonnat, J.-D. (1989b). Practical exact motion planning of a class of robots with three degrees of freedom. In *Proc. of Canadian Conference on Computational Geometry*, page 19. 4, 5, 8, 94, 126

Ball, R. (1876). *The Theory of Screws.* Hodges and Foster, Dublin. 42

Banon, J. (1990). Implementation and extension of the ladder algorithm. In *Proceedings of International Conference on Robotics and Automation*, pages 1548–1553. 94, 126

Basch, J., Guibas, L. J., Hsu, D., and Nguyen, A. T. (2001). Disconnection proofs for motion planning. In *Proc. IEEE International Conference on Robotics and Automation*, pages 1765–1772. 95

Belta, C., Isler, V., and Pappas, G. J. (2005). Discrete abstractions for robot motion planning and control in polygonal environments. *IEEE Transactions on Robotics*, 21(5):864–874. 96, 129

Boor, V., Overmars, M. H., and van der Stappen, A. F. (1999). The gaussian sampling strategy for probabilistic roadmap planners. In *Proceedings of International Conference on Robotics and Automation*, pages 1018–1023. 10, 143, 146

Brooks, R. A. and Lozano-Pérez, T. (1985). A subdivision algorithm in configuration space for findpath with rotation. *IEEE Trans. Syst*, SMC-15:224–233. 5, 8, 94

Cameron, S. (1997). Enhancing GJK: Computing minimum and penetration distance between convex polyhedra. *IEEE International Conference on Robotics and Automation*, pages 3112–3117. 65, 66

Cameron, S. A. and Culley, R. K. (1986). Determining the minimum translational distance between two convex polyhedra. In *Proc. of IEEE Inter. Conf. on Robotics and Automation*, pages 591–596. 20

Canny, J. (1988). *The Complexity of Robot Motion Planning*. ACM Doctoral Dissertation Award. MIT Press. 5, 94

Chang, H. and Li, T. (1995). Assembly maintainability study with motion planning. In *Proceedings of International Conference on Robotics and Automation*. 157

Chazelle, B. (1983). The polygon containment problem. *Advances in Computing Research*, 1:1–33. 24, 59

Cheng, H.-L., Hsu, D., Latombe, J.-C., and Sánchez-Ante, G. (2006). Multi-level free-space dilation for sampling narrow passages in PRM planning. In *Proc. IEEE Int. Conf. on Robotics & Automation*, pages 1255–1260. 144, 146, 147, 168

Choset, H., Lynch, K., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, L., and Thrun, S. (2005). *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press. 2, 93

Conner, D. C., Rizzi, A. A., and Choset, H. (2003). Composition of local potential functions for global robot control and navigation. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3546–3551. 96, 129

de Berg, M., van Kreveld, M., Overmars, M. H., and Schwarzkopf, O. (1997). *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin. 46

Dobkin, D., Hershberger, J., Kirkpatrick, D., and Suri, S. (1993). Computing the intersection-depth of polyhedra. *Algorithmica*, 9:518–533. 20, 23

Donald, B. R. (1987). A search algorithm for motion planning with six degrees of freedom. *Artif. Intell.*, 31(3):295–353. 4, 92, 148

Egan, K., Berard, S., and Trinkle, J. (2003). Modeling nonconvex constraints using linear complementarity. Technical Report 03-13, Department of Computer Science, Rensselaer Polytechnic Institute (RPI). 79

Ehmann, S. and Lin, M. C. (2001). Accurate and fast proximity queries between polyhedra using convex surface decomposition. *Computer Graphics Forum (Proc. of Eurographics'2001)*, 20(3):500–510. 69, 70, 71, 85

Ericson, C. (2004). *Real-Time Collision Detection*. Morgan Kaufmann. 22

Ferr, E. and Laumond, J.-P. (2004). An iterative diffusion algorithm for part disassembly. In *Proceedings of International Conference on Robotics and Automation*, pages 3149–3154. 6, 147, 148, 157, 161, 166, 167, 168, 173

Foskey, M., Garber, M., Lin, M., and Manocha, D. (2001). A voronoi-based hybrid planner. *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*. 127, 147

Garber, M. and Lin, M. (2002). Constraint-based motion planning for virtual prototyping. In *Proc. ACM Symposium on Solid Model and Applications*. 157

Gottschalk, S., Lin, M., and Manocha, D. (1996). OBB-Tree: A hierarchical structure for rapid interference detection. *Proc. of ACM Siggraph'96*, pages 171–180. 66, 71

Grinde, R. and Cavalier, T. (1996). Containment of a single polygon using mathematical programming. *European Journal of Operational Research*, 92(2):368–386. 24, 60

Guibas, L., Holleman, C., and Kavraki, L. (1999). A probabilistic roadmap planner for flexible objects with a workspace medial-axis-based sampling approach. In *Proc. of IROS*, pages 254 – 259. 10

Halperin, D. (2002). Robust geometric computing in motion. *International Journal of Robotics Research*, 21(3):219–232. 4, 5, 94

Halperin, D. (2004). Arrangements. In Goodman, J. E. and O'Rourke, J., editors, *Handbook of Discrete and Computational Geometry*, chapter 24, pages 529–562. CRC Press LLC, Boca Raton, FL. 4, 38

Hirsch, S. and Halperin, D. (2003). Hybrid motion planning: Coordinating two discs moving among polygonal obstacles in the plane. In *Springer Tracts in Advanced Robotics*, pages 239 – 256. 127

Hirukawa, H., Papegay, Y., and Tsukune, H. (1994). A motion planning algorithm of polyhedra in contact for mechanicalassembly. In *20th International Conference on Industrial Electronics, Control and Instrumentation*, volume 2, pages 924–929. 148

Hofer, M. and Pottmann, H. (2004). Energy-minimizing splines in manifolds. In *SIGGRAPH 2004 Conference Proceedings*, pages 284–293. 29

Hsu, D., Kavraki, L., Latombe, J.-C., Motwani, R., and Sorkin, S. (1998). On finding narrow passages with probabilistic roadmap planners. *Proc. of 3rd Workshop on Algorithmic Foundations of Robotics*, pages 25–32. 9, 10, 13, 143, 146, 147

Hsu, D., Latombe, J., and Kurniawati, H. (2006). On the probabilistic foundations of probabilistic roadmap planning. *Int. J. Robotics Research*, 25(7):627–643. 95, 143, 146

Hsu, D., Latombe, J.-C., and Motwani, R. (1999). Path planning in expansive configuration spaces. *International Journal of Computational Geometry and Applications*, 9((4 & 5)):495–512. 29

Hsu, D., Sánchez-Ante, G., and Sun, Z. (2005). Hybrid PRM sampling with a cost-sensitive adaptive strategy. In *Proc. IEEE Int. Conf. on Robotics & Automation*. 127

Ji, X. and Xiao, J. (2000). On random sampling in contact configuration space. *Proc. of Workshop on Algorithmic Foundation of Robotics*. 81

Ji, X. and Xiao, J. (2001). Planning motion compliant to complex contact states. *International Journal of Robotics Research*, 20(6):446–465. 148

Joskowicz, L. and Sacks, E. (1999). Computer-aided mechanical design using configuration spaces. *Computing in Science and Engineering*, 1(6):14–21. 40

Kavraki, L., Svestka, P., Latombe, J. C., and Overmars, M. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. Automat.*, pages 12(4):566–580. 5, 27, 91, 95, 142, 163

Kazerounian, K. and Rastegar, J. (1992). Object norms: A class of coordinate and metric independent norms for displacement. In et al, G. K., editor, *Flexible Mechanism, Dynamics and Analysis: ASME Design Technical Conference, 22nd Biennial Mechanisms Conference*, volume 47, pages 271–275. 28, 29, 30

Kedem, K. and Sharir, M. (1988). An automatic motion planning system for a convex polygonal mobile robot in 2-d polygonal space. In *ACM Symposium on Computational Geometry*, pages 329–340. 4, 94

Keil, J. M. (2000). Polygon decomposition. In Sack, J.-R. and Urrutia, J., editors, *Handbook of Computational Geometry*, pages 491–518. Elsevier Science Publishers B.V. North-Holland, Amsterdam. 125

Kim, Y. J., Lin, M. C., and Manocha, D. (2002a). DEEP: an incremental algorithm for penetration depth computation between convex polytopes. *Proc. of IEEE Conference on Robotics and Automation*, pages 921–926. 20, 23, 56, 65

Kim, Y. J., Lin, M. C., and Manocha, D. (2002b). Fast penetration depth computation using rasterization hardware and hierarchical refinement. *Proc. of Workshop on Algorithmic Foundations of Robotics.* 21, 23, 66

Kloetzer, M. and Belta, C. (2006). A framework for automatic deployment of robots in 2d and 3d environments. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 953–958. 96, 129

Kuffner, J. (2004). Effective sampling and distance metrics for 3d rigid body path planning. In *IEEE Int'l Conf. on Robotics and Automation.* 27

Kuffner, J. and LaValle, S. (2000). RRT-connect: An efficient approach to single-query path planning. In *Proc. IEEE International Conference on Robotics and Automation.* 27, 142, 154

Kurniawati, H. and Hsu, D. (2006). Workspace-based connectivity oracle: An adaptive sampling strategy for prm planning. In *Proc. of 7th International Workshop on the Algorithmic Foundations of Robotics.* 10, 143, 146

Lambert, A. J. D. (2003). Disassembly sequencing: a survey. *International Journal of Production Research*, 41:3721–3759(39). 148

Larsen, E., Gottschalk, S., Lin, M., and Manocha, D. (1999). Fast proximity queries with swept sphere volumes. Technical Report TR99-018, Department of Computer Science, University of North Carolina. 12, 48, 162, 163

Latombe, J.-C. (1991). *Robot Motion Planning*. Kluwer Academic Publishers. 2, 4, 28, 29, 30, 38, 78, 93, 94, 97, 98, 118, 124, 132, 148, 157

LaValle, S. M. (1998). Rapidly-exploring random trees: A new tool for path planning. Technical Report TR 98-11, Computer Science Dept., Iowa State University. 5, 95, 142

LaValle, S. M. (2006). *Planning Algorithms*. Cambridge University Press (also available at http://msl.cs.uiuc.edu/planning/). 2, 27, 28, 29, 32, 33, 93, 104, 154, 155

Lin, M. and Manocha, D. (2003). Collision and proximity queries. In *Handbook of Discrete and Computational Geometry*. 11, 22

Lin, Q. and Burdick, J. (2000). Objective and frame-invariant kinematic metric functions for rigid bodies. *The International Journal of Robotics Research*, 19(6):612–625. 28, 34

Lindemann, S. R. and LaValle, S. M. (2007). Smooth feedback for car-like vehicles in polygonal environments. In *IEEE International Conference on Robotics & Automation*, pages 3104–3109. 96

Lindemann, S. R. and LaValle, S. M. (2009). Simple and efficient algorithms for computing smooth, collision-free feedback laws over given cell decompositions. *The International Journal of Robotics Research*, 28(5):600–621. 96, 129, 133, 134, 135, 136, 137

Lingelbach, F. (2004). Path planning using probabilistic cell decomposition. In *Proc. IEEE International Conference on Robotics and Automation*. 127

Loncaric, J. (1985). *Geometrical analysis of compliant mechanisms in robotics*. PhD thesis, Harvard University. 28

Loncaric, J. (1987). Normal forms of stiffness and compliance matrices. *IEEE Journal of Robotics and Automation*, 3(6):567–572. 28

Lozano-Pérez, T. (1983). Spatial planning: A configuration space approach. *IEEE Trans. Comput.*, C-32:108–120. 2, 92

Lozano-Pérez, T. and Wesley, M. (1979). An algorithm for planning collision-free paths among polyhedral obstacles. *Comm. ACM*, 22(10):560–570. 4, 94

McNeely, W., Puterbaugh, K., and Troy, J. (1999). Six degree-of-freedom haptic rendering using voxel sampling. *Proc. of ACM SIGGRAPH*, pages 401–408. 22

Milenkovic, V. (1998). Rotational polygon overlap minimization and compaction. *Computational Geometry*, 10(4):305–318. 24, 25

Milenkovic, V. (1999). Rotational polygon containment and minimum enclosure using only robust 2d constructions. *Computational Geometry*, 13(1):3–19. 24, 61

Milenkovic, V. and Schmidl, H. (2001). Optimization based animation. In *ACM SIGGRAPH 2001*. 24, 25, 61

Morales, M., Tapia, L., Pearce, R., Rodriguez, S., and Amato, N. M. (2005). C-space subdivision and integration in feature-sensitive motion planning. In *Proc. IEEE International Conference on Robotics and Automation*. 127, 146

Mount, D. (1992). Intersection detection and separators for simple polygons. In *Proc. 8th Annual ACM Sympos. Comput. Geom*, pages 303–311. 67

Murray, R., Li, Z., and Sastry, S. (1994). *A Mathematical Introduction to Robotic Manipulation*. CRC Press. 37, 42

Nawratil, G., Pottmann, H., and Ravani, B. (2009). Generalized penetration depth computation based on kinematical geometry. *Computer Aided Geometric Design*, 26(4):425–443. 89

Ong, C. (1997). On the quantification of penetration between general objects. *International Journal of Robotics Research*, 16(3):400–409. 13, 24, 56

Ong, C. and Gilbert, E. (1996). Growth distances: New measures for object separation and penetration. *IEEE Transactions on Robotics and Automation*, 12(6). 20, 24, 169

Paden, B., Mess, A., and Fisher, M. (1989). Path planning using a jacobian-based freespace generation algorithm. In *Proceedings of International Conference on Robotics and Automation*. 5, 8, 95, 104

Park, F. (1995). Distance metrics on the rigid-body motions with applications to mechanism design. *ASME J. Mechanical Design*, 117(1):48–54. 28

Pisula, C., Hoff, K., Lin, M., and Manocha, D. (2000). Randomized path planning for a rigid body based on hardware accelerated voronoi sampling. In *Proc. of 4th International Workshop on Algorithmic Foundations of Robotics*. 95

Plaku, E. and Kavraki, L. E. (2006). Quantitative analysis of nearest-neighbors search in high-dimensional sampling-based motion planning. In *Workshop on Algorithmic Foundations of Robotics (WAFR)*, New York, NY. 28

Raab, S. (1999). Controlled perturbation for arrangements of polyhedral surfaces with application to swept volumes. In *Proc. 15th ACM Symposium on Computational Geometry*, pages 163–172. 4, 39

Redon, S. (2004). Fast continuous collision detection and handling for desktop virtual prototyping. *Virtual Reality*, 8(1):63–70. 84, 86

Redon, S. and Lin, M. (2005). Practical local planning in the contact space. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4200–4205. 24, 82, 147

Redon, S. and Lin, M. (2006). A fast method for local penetration depth computation. *Journal of Graphics Tools*, 11(2):37–50. 146, 147, 148

Reif, J. (1979). Complexity of the mover's problem and generalizations. In *20th Annual IEEE Symposium on Foundations of Computer Science*, pages 421–427, San Juan, Puerto Rico. 5

Rimon, E. and Koditschek, D. E. (1992). Exact robot navigation using artificial potential fields. *IEEE Transactions on Robotics and Automation*, 8(5):501–518. 96

Rodriguez, S., Tang, X., Lien, J., and Amato, N. (2006). An obstacle-based rapidly-exploring random tree. In *Proceedings of International Conference on Robotics and Automation*, pages 895–900. 147, 164, 167

Ruspini, D. and Khatib, O. (1997). Collision/contact models for the dynamic simulation of complex environments. *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*. 82

Ruspini, D. and Khatib, O. (2000). A framework for multi-contact multi-body dynamic simulation and haptic display. In *Proceedings of International Conference on Robotics and Automation*. 40

Saha, M., Latombe, J., Chang, Y., Lin, and Prinz, F. (2005). Finding narrow passages with probabilistic roadmaps: the small step retraction method. *Intelligent Robots and Systems*, 19(3):301–319. 143, 144, 146, 147, 168

Schwartz, J. T. and Sharir, M. (1983). On the piano movers probelem ii, general techniques for computing topological properties of real algebraic manifolds. *Advances of Applied Maths*, 4:298–351. 5, 94, 129

Schwarzer, F., Saha, M., and Latombe, J. (2005). Adaptive dynamic collision checking for single and multiple articulated robots in complex environments. *IEEE Tr. on Robotics*, 21(3):338–353. 32, 100, 104

Simeon, T., Laumond, J. P., and Nissoux, C. (2000). Visibility based probabilistic roadmaps for motion planning. *Advanced Robotics Journal*, 14(6). 10, 95, 143, 146

Spivak, M. (1999). *A Comprehensive Introduction to Differential Geometry*. Publish or Perish Press, 3 edition. 47

Strandberg, M. (2004). Augmenting RRT-planners with local trees. In *Proceedings of International Conference on Robotics and Automation*, volume 4, pages 3258–3262. 167

Sud, A., Govindaraju, N., Gayle, R., Kabul, I., and Manocha, D. (2006). Fast proximity computation among deformable models using discrete voronoi diagrams. *Proc. of ACM SIGGRAPH*, pages 1144–1153. 24

Sun, Z., Hsu, D., Jiang, T., Kurniawati, H., and Reif, J. (2005). Narrow passage sampling for probabilistic roadmap planners. *IEEE Trans. on Robotics*, 21(6):1105–1115. 146

Tchon, K. and Duleba, I. (1994). Definition of a kinematic metric for robot manipulators. *Journal of Robotic Systems*, 11(3):211–221. 28

Tesic, R. and Banerjee, P. (2002). Motion modeling concepts in virtual manufacturing simulator. *Journal of Advanced Manufacturing*, 1:37–49. 157

van den Berg, J. and Overmars, M. (2005). Using workspace information as a guide to non-uniform sampling in probabilistic roadmap planners. *The International Journal of Robotics Research*, 24(12):1055–1071. 146

van den Bergen, G. (2001). Proximity queries and penetration depth computation on 3d game objects. *Game Developers Conference*. 20, 21, 23, 56, 65, 66, 71, 106

Varadhan, G., Kim, Y., Krishnan, S., and Manocha, D. (2006). Topology preserving approximation of free configuration space. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3041–3048. 39

Varadhan, G. and Manocha, D. (2005). Star-shaped roadmaps - a deterministic sampling approach for complete motion planning. In *Proceedings of Robotics: Science and Systems*, Cambridge, USA. 5, 94, 109, 126

Wein, R. (2008). 2d Minkowski sums. In Board, C. E., editor, *CGAL User and Reference Manual*. 3.4 edition. 106

Wilmarth, S. A., Amato, N. M., and Stiller, P. F. (1999). Motion planning for a rigid body using random networks on the medial axis of the free space. In *Symposium on Computational Geometry*, pages 173–180. 146, 147

Wilson, R. H. and Latombe, J.-C. (1994). Geometric reasoning about mechanical assembly. *Artif. Intell.*, 71:371–396. 148

Wise, K. D. and Bowyer, A. (2000). A survey of global configuration-space mapping techniques for a single robot in a static environment. *The International Journal of Robotics Research*, 19(8):762–779. 95

Xiao, J. and Ji, X. (2001). On automatic generation of high-level contact state space. *International Journal of Robotics Research*, 20(7):584–606. 78, 148

Yang, L. and LaValle, S. M. (2004). The sampling-based neighborhood graph: A framework for planning and executing feedback motion strategies. *IEEE Transactions on Robotics and Automation*, 20(3):419–432. 96

Yershova, A., Jaillet, L., Simeon, T., and LaValle, S. M. (2005). Dynamic-domain RRTs: Efficient exploration by controlling the sampling domain. In *Proceedings of International Conference on Robotics and Automation*, pages 3856–3861. 154, 167

Zhang, L., Huang, X., Kim, Y., and Manocha, D. (2008a). D-Plan: Efficient collision-free path computation for part removal and disassembly. *Journal of Computer-Aided Design and Applications*, 5(6):774–786. 17, 143, 158

Zhang, L., Kim, Y., and D.Manocha (2008b). Efficient distance computation in configuration space. *Computer Aided Geometric Design*, 25(7):489–502. 14, 27, 28, 29, 35, 40

Zhang, L., Kim, Y., and Manocha, D. (2007a). A fast and practical algorithm for generalized penetration depth computation. In *Proceedings of Robotics: Science and Systems*, Atlanta, GA, USA. 14, 15, 25, 55, 77

Zhang, L., Kim, Y., and Manocha, D. (2007b). A hybrid approach for complete motion planning. In *IEEE/RSJ International Conference On Intelligent Robots and Systems (IROS)*, pages 7–14, San Diego, CA, USA. 16, 93, 110

Zhang, L., Kim, Y., and Manocha, D. (2008c). Efficient cell labelling and path non-existence computation using c-obstacle query. *The International Journal of Robotics Research*, 27(11-12):1246–1257. 12, 16, 92, 132, 138

Zhang, L., Kim, Y., Varadhan, G., and D.Manocha (2007c). Generalized penetration depth computation. *Computer-Aided Design*, 39(8):625–638. 14, 25, 28, 29, 55, 56, 102, 147

Zhang, L., LaValle, S., and Manocha, D. (2009). Global vector field computation for feedback motion planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 477–482. 16, 93, 129

Zhang, L. and Manocha, D. (2008a). Constrained motion interpolation with distance constraints. In *International Workshop on the Algorithmic Foundations of Robotics (WAFR)*, pages 269–284. 163

Zhang, L. and Manocha, D. (2008b). A retraction-based RRT planner. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3743–3750. 17, 95, 142

Zhang, X., Lee, M., and Kim, Y. (2006). Interactive continuous collision detection for non-convex polyhedra. In *Pacific Graphics 2006 (Visual Computer)*. 84

Zhu, D. and Latombe, J. (1990). Constraint reformulation in a hierarchical path planner. *Proceedings of International Conference on Robotics and Automation*, pages 1918–1923. 5, 8, 92, 95, 98