RANDOMIZED SINGLE-QUERY MOTION PLANNING IN EXPANSIVE SPACES

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

David Hsu

May 2000

Abstract

Random sampling is a fundamental technique for motion planning of objects with many degrees of freedom (dof). This thesis presents efficient randomized algorithms for *single-query* motion planning of objects with many dofs and under complex motion constraints. Unlike most other probabilistic roadmap planners, our algorithms perform no preprocessing of the environment. They sample collision-free configurations incrementally in the connected components of the space that contain the query configurations, thus avoiding the high cost of pre-computing a roadmap for the entire space. Two specific planners are discussed. One addresses the simpler problem of path planning. The other extends the basic idea and takes into account kinematic and dynamic constraints on motion as well. A control system is used to represent both types of constraints in a unified framework. Our algorithms have been tested extensively on both synthesized examples and real-life CAD data from the industry; they have shown strong performance on rigid-body and articulated objects with up to 18 dofs. We also demonstrate their generality and effectiveness in three practical applications: assembly maintainability checking, motion synthesis for animated characters, and kinodynamic motion planning for an integrated real-time robot system in environments with moving obstacles.

The lack of theoretical explanation for the randomized motion planners' success in experiments has motivated us to introduce the notion of *expansive spaces* as a new way to characterize the complexity of input environments. It provides us a conceptual framework to understand why randomized motion planners work well and under what conditions. We prove that in an expansive space, our algorithms find a solution trajectory with probability that converges to 1 at an exponential rate, if a solution exists.

An efficient motion planner is also useful as a primitive for accomplishing more complex tasks. An example of this is the robot placement problem, an important application from the manufacturing industry. By combining a randomized path planner with local iterative optimization, our placement algorithm computes simultaneously a base location and a corresponding collision-free path for a fixed-base robot manipulator to execute specified tasks as efficiently as possible.

Acknowledgments

First and foremost, I thank my advisor Jean-Claude Latombe for being an exemplary researcher to me. He has given me much guidance and support throughout my time at Stanford. His insight and vision have been instrumental in completing this work.

I also thank Rajeev Motwani and Leonidas Guibas for reading my thesis. I was fortunate to have worked with both of them. They have always kept their doors open to me and provided me invaluable technical advice.

Several people gave me input which improved the quality of this work. Paul LeBlanc at General Motors R & D Center made lots of effort in winning the approval for me to use the GM CAD data for my research. Charles Wampler, also at General Motors R & D Center, suggested to me the robot placement problem, which has extended the scope of the work reported here. I spent a summer in 1997 with Dr. Michael Cohen at Microsoft Research, working on computer animation. This experience greatly broadened my perspective on the potential application of my research. This work also benefited from collaboration with Stanford Aerospace Robotics Laboratory and GRASP Laboratory at The University of Pennsylvania.

Financial support for this work was provided by ARO MURI grant DAAH-04-96-1-007, a Microsoft Graduate Fellowship, and a grant from General Motors through Stanford Integrated Manufacturing Association (SIMA).

At Stanford, I had the opportunity to work among a group of excellent people: Héctor González-Baños, Lydia Kavraki, Robert Kindel, James Kuffner, Steve LaValle, David Lin, Li Zhang, and many others. I have learned much from them.

I am grateful to many people whose friendship has made my time at Stanford so memorable. In particular, I would like to thank Héctor González-Baños, Jeffrey Su, Lyman and J.-Mae Taylor, Tina Tian, and William and Elaine Zee. Jeff and Tina generously helped me when I arrived at Stanford. They also enriched my spiritual life. Héctor has always made himself available and offered me abundant good advice whenever I have to make important decisions. Lyman and J.-Mae Taylor showed me the American way of life from an interesting angle, which I would have missed otherwise. William and Elaine treated me like their own child and gave me a family away from home.

As an undergraduate, I studied at The University of British Columbia in the beautiful city of Vancouver, Canada. There I met two most wonderful mentors, George Bluman and Maria Klawe, who pointed me to the right direction at a crucial stage in my life. I am indebted to them for their guidance, constant encouragement, and trust in my abilities. I hope that I have lived up to their expectations. I also thank David Kirkpatrick, Jack Snoeyink, and Robert Woodham for instilling in me the excitement of research.

Above all, I thank my family: my grandparents Jia-Feng and Lan Xu, my parents Tian-Biao and Fu-Di Xu, and many uncles and aunts. My grandparents brought me up. They implanted in me the value of honesty and diligence. Their love and care, I can never repay. When I was younger, my parents provided me with much-needed guidance and support to survive an extremely competitive school system and ultimately gave me the freedom to explore on my own. My aunt and uncle, Yi-Hua Xu and Lin Zhou, cared for me in a very difficult time in my life. Their love and sense of humor kept me going in even the most miserable days.

I owe much to my wife Diana for her unconditional love, understanding, and support and for sharing much fond memory together. Without her “distractions”, this work would probably have been finished a little earlier, but with much less fun. Her presence is a constant source of inspiration in my life.

Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

Geometry and motion are two ubiquitous phenomena of the physical world. To operate in this world or to simulate it, we need proper modeling of the environment, compact data structures to represent the models, and efficient algorithms to generate motion for different types of moving objects. Motion planning for objects with many degrees of freedom (dof) and under complex motion constraints is the subject of this thesis.

Motion planning is concerned with computing collision-free motion for objects in an environment populated with obstacles. It has its origin in robotics, where planning collision-free motion to achieve a specified goal is a fundamental characteristic of autonomous robots. Motion planning has since found applications outside traditional robotics, in domains as disparate as design for manufacturing, computer animation, medical surgery simulation, and computational biology. For example, in computer animation, motion planning techniques enable animated characters to respond to task-level commands such as “go to the table and pick up the apple” [JBN94, KKKL94, KL99], thus making them more interactive and lively. In computational biology, motion planning helps discovering new drugs by identifying paths with desirable properties for small drug molecules to dock in the cavities of large protein structures [SLB99]. These new applications are rising as the driving forces of motion planning research.

In its simplest form, motion planning is a purely geometric problem: given a description of the geometry of an object and a static environment, our goal is to find a collision-free path for the object to move from an initial configuration to a goal configuration. This is called the *basic motion planning problem* [Lat91b]. A key difficulty in developing a general motion planning algorithm (or a motion planner, as it is often called) lies in the large number of dofs that an object may have. A free-flying rigid body in three dimensions has six dofs. An articulated robot manipulator can have arbitrary large dofs depending on the number of joints that it has. Unfortunately the fastest *complete* algorithm (an algorithm that finds a path if one exists and reports that none exists otherwise) is exponential in the number of dofs of a moving object [Can88a]. In addition, the motion of objects is often subject to physical constraints. For example, a car cannot move sidewise; the motion of a bouncing ball must obey the laws of physics. These constraints compound the difficulty of motion planning.

We believe that random sampling is a fundamental technique for attacking the exponential dependency of motion planning algorithms on the number of dofs. By sacrificing a limited amount of completeness in a well-understood manner, random sampling significantly improves the efficiency of motion planners, thus making them practical in a wide range of applications. Intuitively the success of random sampling in motion planning problems is due to the "abundance of witnesses" [Kar91]: while feasible trajectories may lie in a space difficult to search efficiently by a deterministic algorithm, many solutions actually exist, and thus it is relatively easy to obtain one by sampling the space at random.

In this thesis, we present a novel randomized motion planner for rigid-body and articulated objects with many dofs and extend the algorithm to deal with objects whose motion is subject to kinematic and dynamic constraints. We also provide probabilistic analyses of these techniques under a suitable geometric characterization of the environment and demonstrate their effectiveness in several practical applications.

1.1 Overview of Motion Planning Problems

Motion planning is not a single problem, but a collection of problems sharing some common characteristics. The goal of a motion planner is to transform the world from an initial state to a goal state by computing a sequence of admissible motions for the moving objects in the presence of obstacles. The basic path planning problem stated at the beginning of the chapter captures this essential feature, though it only deals with the geometric relationship between the moving objects and the obstacles. Various extensions of the basic problem have been studied in the literature. We group them below into four categories. Since motion planning is a vast field, it is clearly impossible to cite all the important work. The references listed here are only representative ones from a limited number of viewpoints of the author.

More complex moving objects Extensions in this category are driven by more and more realistic modeling of complex moving objects. Motion planning under kinematic and dynamic constraints [Lau86, BL93, DXCR93, BDG85] has received much attention. Coordinating the motion of multiple moving objects [SS83b, ELP87, OLP89] is another interesting problem. In robotics, the most common moving objects are, of course, robots. Researchers have developed modular robots, which consist of hundreds of individual pieces, each operating independently, but coordinating with one another in order to achieve a goal. Transforming a modular robot from an initial configuration to a goal configuration is a challenging new problem [PEUC97, KRVM98, CY99, NGY00]. More recently, motion planning for deformable objects [KLH98] has gained importance, especially in the context of virtual prototyping, whose goal is to reason about the motion of objects during manufacturing or maintenance.

Dynamic Environments In the basic motion planning problem, all obstacles are assumed to be static. However, many environments in the physical world are dynamic. An environment may contain moving obstacles, whose motion is either completely known [RS85, KZ86, Fuj95] or only partially known [QK93, LS95]. It may also contain movable objects [ALS95, KL94b], which can be manipulated in order to accomplish a task.

Incomplete Information Most of the above problems assume that the environment containing the moving objects is completely known *a priori*, but the assumption does not always hold in practice. For example, a robot may have very limited information about the environment in advance, but is equipped with sensors (*e.g.*, cameras and laser rangers) to obtain new information from the environment. Sensor data provide the information that the robot needs to explore the environment and perform specified tasks. At the same time, they restrict the amount of information that the robot can use to decide its action: a robot can only "see" a subset of the environment detected by its sensors. Sensor-based motion planning [LS87, QK93, CB97, Rim97, GBGL$^+$98] aims at generating efficient motion strategies under these constraints.

Uncertainty The use of motion planning in physical environments—as opposed to simulated ones—poses additional challenges. In a physical environment, robots must deal with imprecision in control and sensing. Preimage backchaining [LPMT84, Bri89] is an elegant framework for addressing this issue. It has been applied to mobile robot navigation, but its impact so far remains limited [Lat00]. In practice, most robotic systems use some form of reactive execution to deal with the imprecision in control and sensing.

This work addresses the basic motion planning problem as well as the extension that takes into account kinematic and dynamic constraints. The techniques proposed here are applicable to environments that are either static or contain moving obstacles with known motion. In the following, we will use the term *path planning* for the basic motion planning problem, and use the term *motion planning* for various extensions of the basic problem.

1.2 Randomized Motion Planning: Algorithms, Analyses, and Applications

Random sampling has been applied successfully in recent years to solve path planning problems for objects with many dofs. Previous work deals mainly with the multi-query problem, *i.e.*, to process many queries in a known static environment (see, *e.g.*, [KL94a, ŠO95b, HST94, AW96, BOvdS99]). These algorithms typically preprocess the environment for a relatively long time by random sampling so that path planning queries for the given environment can be answered efficiently later. However, in many practical applications, environments change frequently, and it is wasteful of computation time to perform expensive preprocessing. The focus of this work is to develop efficient random-sampling techniques for the single-query problem, in which we solve the path planning problem for only one query. Like the previous randomized path planners, our algorithm builds a graph, whose vertices correspond to collision-free configurations of an object and whose edges correspond to simple collision-free paths between the vertices; this graph encodes the connectivity information among the collision-free configurations. Unlike the previous work, our algorithm builds the graph by first sampling in the neighborhoods of the query configurations and then iteratively in the neighborhoods of the newly-sampled configurations. It stops as soon as a path is found between the initial and the goal configuration. The

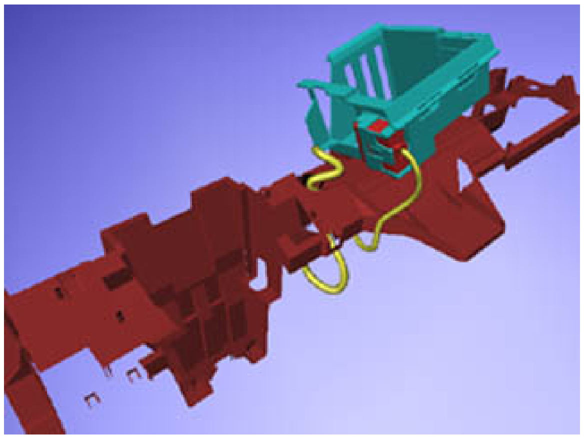

Figure 1.1. Assembly maintainability checking, whose goal is to remove a specified component from an assembly of mechanical parts without collision.

new sampling strategy offers two main advantages. First, it samples only the connected components of the space containing the initial and the goal configurations, thus avoiding the high cost of pre-computing a graph for the entire space. Second, it is more efficient in obtaining collision-free configurations, because it samples only the neighborhoods of configurations known to be collision-free. Our algorithm has been applied to difficult assembly maintainability problems from the automotive industry (using General Motors CAD data describing environments with up to 200,000 triangles), and have demonstrated strong performance. See Figure 1.1 for an example.

Path planning is the simplest type of motion planning problems. When an object's motion is constrained and it cannot move with arbitrary velocity and acceleration, our problem becomes more complex, because the final computed motion not only has to be collision-free, but also satisfies all the constraints. Here we consider two important classes of constraints, non-holonomic (kinematic) constraints and dynamic constraints. The former imposes a relationship between the configuration of an object and its velocity; the latter involves the acceleration as well. Despite the apparent difference, both types of constraints can be cast into the same mathematical form and represented by a *control system*, which is a system of differential equations that describes all the possible movements of an object locally. To sample new configurations, we first pick at random a point in the space of allowable control functions and then transform the sampled point into a valid configuration

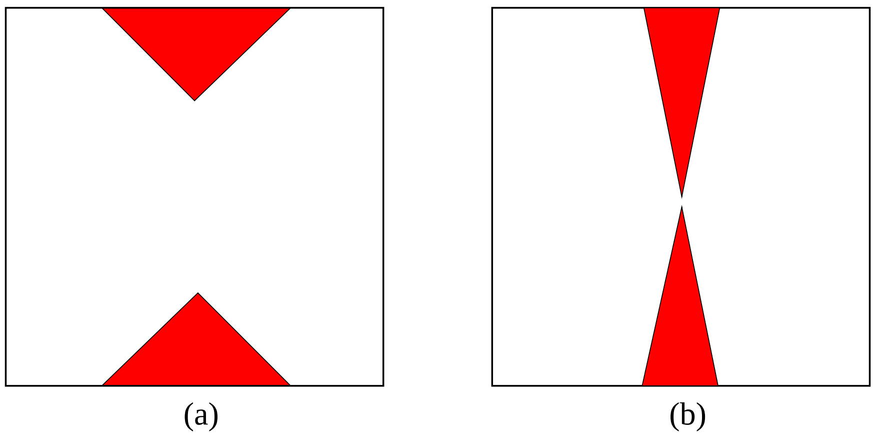

Figure 1.2. A simple space and a difficult space.

of the object. With some modifications, our randomized algorithm for path planning is extended to address this much broader class of problems that takes into account kinematic and dynamic constraints on the motion of objects. Our framework is also general enough to deal with moving obstacles in the environment.

The efficiency of randomized motion planning algorithms has been widely observed in this and earlier work, but there has been little theoretical work to explain the experimental success. We introduce the notion of *expansive spaces* as a geometric characterization of the environment and use it to analyze the performance of our planners. Our definition of expansive spaces uses the volumes of certain subsets in the space of all possible configurations of an object, rather than the more traditional measures such as the number of dofs of the object and the number of polynomials or polygons describing obstacle boundaries. Consider the 2-D examples in Figure 1.2. Both spaces have the same dimensionality and the same number of edges describing the obstacle boundaries. However, experiments show that the space in Figure 1.2a, which no narrow passages, is much easier for a randomized motion planner, than the one in Figure 1.2b, which has a narrow passage connecting the left and right portions of the space. Expansiveness gives a quantitative measure of the impact of narrow passages on the difficulty of motion planning. Expansiveness is an important notion, because it provides a conceptual framework for understanding why randomized motion planning algorithms work well (or not so well) and under what conditions. Our definition also allows kinematic and dynamic constraints on the motion of objects to be incorporated during the analysis.

At the beginning of the chapter, we have mentioned that path planning is directly useful in a number of practical applications. In addition, an efficient randomized path planner can be used as a primitive operation to solve more complex problems. We demonstrate this on the robot placement problem, an important application from the manufacturing industry. In a robot placement problem, our goal is to find a base location for a fixed-base robot manipulator so that specified tasks are executed as efficiently as possible. This is a crucial issue in robot workcell design. Our proposed algorithm combines a randomized path planner with iterative optimization techniques to compute simultaneously a base location and a corresponding collision-free path that are optimized with respect to the execution time of tasks.

1.3 Outline

This thesis presents randomized algorithms for motion planning and practical applications of the algorithms.

The first part (Chapters 2–4) forms the core of the work, in which we examine randomized motion planning from both an experimental and a theoretical point of view. Chapter 2 treats the path planning problem. We present general schemes for three variants of the problem and a specific planner for one variant. We also discuss implementation of the planner and experiments with articulated and rigid-body objects in 3-D environments. Chapter 3 extends the planner to deal with objects whose motion is under kinematic and dynamic constraints. Our algorithm represents both types of constraints by a control system and treats them in a unified framework. Experiments of the planner on two example systems are reported. The first one consists of two wheeled mobile robots that maintain a direct line of sight as well as a minimum and a maximum distance between them. The second one is a hovercraft with simplified dynamics. Chapter 4 is devoted to the probabilistic analyses of our planners. We show that under a suitable characterization of the environment, called expansiveness, the failure probability of our planners decreases exponentially as running time increases.

The second part of the thesis (Chapter 5) illustrates the usefulness of randomized motion planning techniques in a practical application. It discusses the robot placement problem, in which we would like to find a base location for a robot manipulator so that specified tasks are executed as efficiently as possible.

Finally, in Chapter 6, we summarize the main results and point out directions for future research.

CHAPTER 2

Path Planning in High-Dimensional Configuration Spaces

In a path planning problem, our goal is to find a collision-free path for an object to move from an initial configuration to a goal configuration. This is a provably hard problem [Rei79], because the complexity of path planning increases dramatically as the number of dofs of moving objects grows.

Randomization is the most effective technique for reducing the high cost associated with path planning of moving objects with many dofs. The main objective of this chapter is to present randomized techniques for path planning. We start with a brief description of configuration space, a powerful framework for understanding motion planning problems (Section 2.1) and review related work on path planning (Section 2.2). We then move on to general schemes for three variants of the path planning problem (Section 2.3). These schemes help to identify the commonalities among different randomized path planners and pinpoint the key issues in their design. Next we focus on a specific planner for the single-query problem (Sections 2.4 and 2.5) and discuss techniques for optimizing paths generated by randomized planners (Section 2.6). Our planner has demonstrated very good performance on both rigid-body and articulated objects. Experimental results on the planner are reported in Sections 2.7–2.9. We end the chapter with some comments on the strength of our new planner and areas for further improvement (Section 2.10).

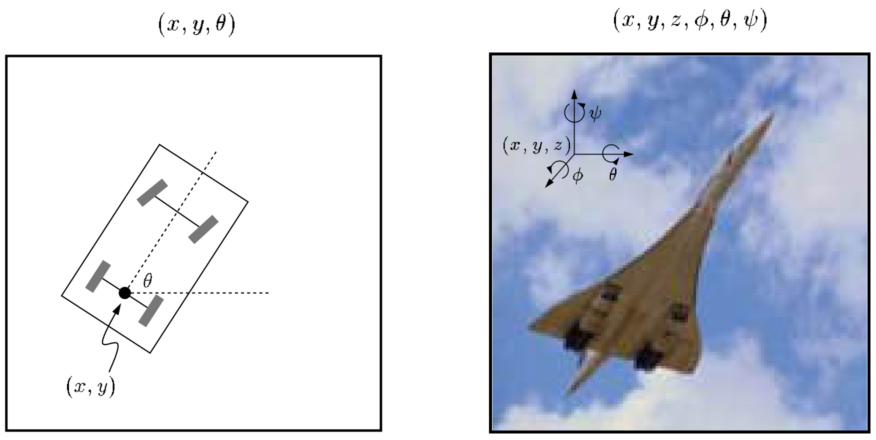

Figure 2.1. The configuration of a rigid body.

2.1 Configuration Spaces

The *configuration* of a moving object M is a set of parameters that uniquely determines the position of every point in M. For example, the configuration of a rigid body is its position and orientation (Figure 2.1); the configuration of an articulated object is usually a list of joint angles (Figure 2.2).

The set of all configurations of M forms the *configuration space* $\mathcal{C}$. A configuration q is *free* if M does not collide with the obstacles in the environment or with itself when placed at q. The set of all free configurations forms the free space $\mathcal{F} \subset \mathcal{C}$.

For a polygonal rigid body P translating in a 2-D polygonal environment, we can construct the configuration space $\mathcal{C}$ explicitly by computing the Minkowski difference of P and the obstacles [LP83, GRS83]. Basically we "grow" the obstacles by the shape of P and shrink P to a point. The idea of transforming the moving object to a point in a suitable space was first introduced by Udupa for collision avoidance [Udu77], and exploited more systematically by Lozano-Pérez and Wesley in proposing the first path planner for polygonal robots translating without rotation [LPW79]. However, explicitly constructing $\mathcal{C}$ is very complex and inefficient for high-dimensional configuration spaces. So instead, we represent $\mathcal{C}$ implicitly by a function CLEARANCE : $\mathcal{C} \mapsto$ R, which maps a configuration $q \in \mathcal{C}$ to the approximate distance between an object placed at q and the obstacles. The function returns 0 if q is in collision. It admits very efficient implementation by hierarchical

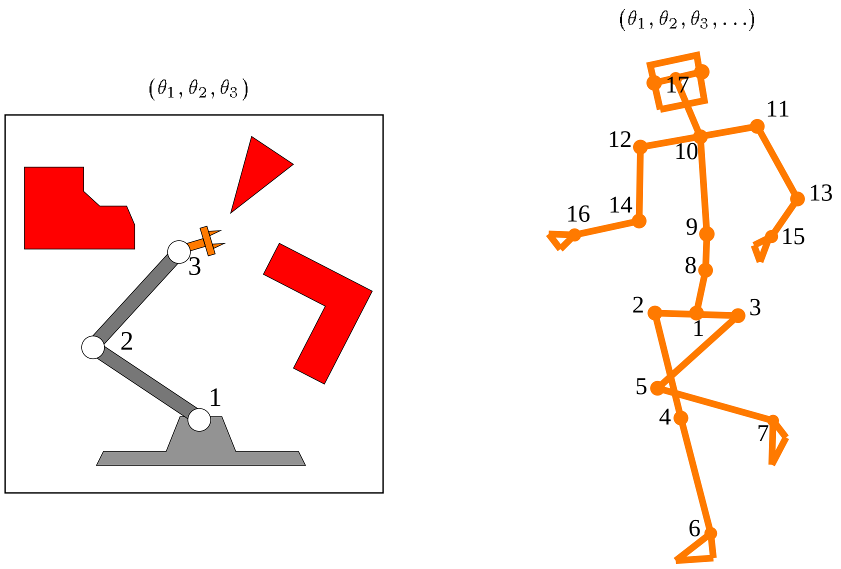
$(\theta_1, \theta_2, \theta_3)$
1
2
3
$(\theta_1, \theta_2, \theta_3, \ldots)$
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17

Figure 2.2. The configuration of an articulated object.

collision detection or distance computation algorithms (*e.g.*, [Qui94, GLM96, GHZ99]).

In the configuration space, the moving object M is mapped to a point. The path planning problem for M becomes that of finding a path for a point in the free space $\mathcal{F}$. If M has n degrees of freedom, then it takes a minimum of n parameters to specify a configuration of M. So the dimension of $\mathcal{C}$ is n. As we may expect, path planning becomes increasingly more difficult as the dimension of $\mathcal{C}$ grows.

2.2 Path Planning Algorithms

Traditional path planning techniques typically follow one three of the approaches: roadmap, cell decomposition, and artificial potential field (see [Lat91b] for a complete survey). Some of these techniques are very efficient for moving objects with a small number of dofs (2 or 3). However, their performance quickly degrades as the number of dofs increases. Depending on the speed of hardware, they can solve path planning problems for objects with up to four or five dofs. So they cannot compute motion for rigid bodies translating and rotating freely in 3-D space or 6-dof articulated robot manipulators, two important cases in practice.

Theoretical studies have confirmed the difficulty of path planning. They indicate that a complete planner may take time exponential in the number of dofs of the moving object. Reif gave the first lower bound for path planing. He showed that the *generalized mover's problem* is PSPACE-hard [Rei79]:

> Suppose that a moving object M is composed of a set of polyhedra linked together at some vertices and that the environment consists of static polyhedral obstacles. Finding a collision-free path for M between two given configurations is PSPACE-hard.

Later on, Canny showed that the generalized mover's problem is in PSPACE [Can88b], thus establishing that it is PSPACE-complete. A number of other path planning problems have also been shown to be PSPACE-hard, including that of planar linkages [HJW84] and multiple moving rectangles in a plane [HSS84]. These lower bounds suggest that the complexity of path planning likely grows exponentially with the dimension of the configuration space.

The algorithm of Shwartz and Sharir [SS83a] provides an upper bound on the complexity of path planning:

> Suppose that a configuration space $\mathcal{C}$ has dimension n and obstacles in $\mathcal{C}$ are defined by p polynomial constraints of maximum degree d. A collision-free path between two given configurations can be found in time exponential in n and polynomial in p and d.

Their algorithm computes an algebraic decomposition of the configuration space and is doubly exponential in n. Canny's roadmap technique [Can88a], which captures the connectivity of the configuration space in a network of one-dimensional curves, reduces the running time to singly exponential in n. It is the best complete algorithm known for path planing. These algorithms have helped calibrating the complexity of path planning and understanding its combinatorial nature [Lat00], but they are very complicated and impractical to implement.

In recent years, random sampling has emerged as a promising new approach for path planning. Randomized path planners are capable of solving complex path planning problems for moving objects with many dofs. They are simple to implement and very efficient in practice. These benefits of random sampling have been exploited in other domains as well, *e.g.,* Monte Carlo integration and simulation of stochastic systems [KW86]. Here instead of using random sampling to estimate a numerical value, we use it to estimate the connectivity of the configuration space.

Planners based on random sampling are not complete. Some of them satisfy a weaker, but still interesting property called *probabilistic completeness*: if a path exists, the probability that the planner finds it converges to 1 quickly, as running time increases.

Early randomized path planners often use randomization to augment a more traditional planning method. The randomized potential field planer [BL91] alternates between best-first motions that track the negated gradient of a potential function measuring progress towards the goal and random motions to escape the local minima of the potential function. The algorithm proposed by Glavina generates random intermediate configurations to help a goal-directed best-first search to escape local minima [Gla90]. Although it was implemented only for polygonal objects moving in 2-D environments, Glavina's algorithm contains several major ingredients of the powerful probabilistic roadmap (PRM) method that came a few years later.

PRM planners proceeds in two stages. In the preprocessing stage, it samples collision-free configurations at random and connects them by simple canonical paths, thus creating a *probabilistic roadmap*. In the query stage, it connects the two query configurations to the roadmap and searches the roadmap for a path. The first PRM planners [KL94a, ŠO95a, HST94] use a straightforward uniform distribution for sampling new configuration, possibly followed by an enhancement step to increase the sampling density in critical regions [KL94a]. Recently a number of other sampling strategies have been developed, including shrinking the obstacles [HKL+98], sampling near the free space boundaries [ABD+98] or medial axis of the configuration space [WAS99], and using "guards" to reject unwanted samples [NSL99].

Since PRM planners perform relatively expensive pre-computation, they are most suitable for processing multiple queries in a static environment. The algorithm that we are going to present addresses a different variant of the path planning problem, in which the environment changes frequently and expensive pre-computation is wasteful. The algorithm, initially proposed in [HLM97], builds a roadmap on the fly by sampling first in the neighborhood of the query configurations and then iteratively in the neighborhoods of the newly-sampled configurations. It stops as soon as a path is found between the initial and the goal configuration. Compared to a distribution that samples configuration space with non-zero probability everywhere [Ove92], the sampling strategy presented here is more efficient, especially when the free space contains many connected components, because it samples only those components relevant to the current query. The algorithm proposed by LaValle

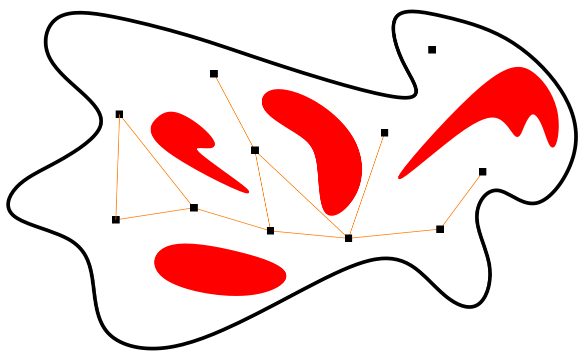

Figure 2.3. An example of probabilistic roadmaps.

and Kuffner is similar to ours, but uses a slightly different sampling strategy [LK99].

2.3 General Schemes

To find a collision-free path, a randomized path planner builds a roadmap graph G that captures the connectivity of the configuration space. It samples at random a set of free configurations called *milestones* and inserts them into G as vertices. There is an edge between two milestones in G if they can be connected by a canonical path from a pre-determined set of paths (usually straight line segments). The roadmap G is thus a sampled representation of the configuration space; it can be searched to answer a path planning query. See Figure 2.3 for an illustration.

There are three variants of the path planning problem, and the algorithms for constructing the roadmaps are slightly different in each case. In the first variant, we pre-compute a roadmap so that multiple planning queries about the same environment can be answered as fast as possible. This variant often occurs in mobile robot navigation, where a robot moves between two given locations in a static environment. In the second variant, we assume that the moving object has a *home* configuration q_h and a query always asks for a path between q_h and another configuration. Both human arms and robot manipulators have natural resting configurations that serve as home configurations. We still pre-compute a roadmap G, but to improve the speed of pre-computation, G contains only milestones that can be connected to the home configuration. In the third variant, we perform no pre-computation at all and build a small roadmap on the fly in order to answer the query. This scenario occurs if environments change frequently and pre-computation is not feasible. As far as roadmap construction is concerned, these three variants differ in the amount of query information available when the roadmap is generated: nothing is known about the query in the first case, one of the two query configurations is known in the second case, and the query is completely known in the last case. We call the first variant the *all-pairs problem*, the second variant, the *single-source problem*, and the third variant, the *single-pair problem*.

The all-pairs problem In an all-pairs problem, the planner builds a roadmap G by first sampling the configuration space at random according to a suitable probability distribution and retains the free configurations as milestones in G. It inserts an edge between two milestones in G if they can be connected by a straight-line path. Once G is constructed, the planner answers queries by first connecting the initial configuration q_{init} and the goal configuration q_{goal} to G and then searching G for a path between q_{init} and q_{goal} [BKL+97]. A sketch of the roadmap construction algorithm is shown below.

Scheme 2.1 Roadmap construction for all-pairs problems.

```
1.  repeat
2.    Pick $q$ from $\mathcal{C}$ at random with probability $\pi(q)$.
3.    if CLEARANCE$(q) > 0$ then
4.      Insert $q$ into the roadmap $G$ as a milestone.
5.      for every milestone $q' \in G$ such that $q' \neq q$
6.        if LINK$(q, q')$ then
7.          Insert an edge into $G$ between $q$ and $q'$.
```

In line 6 of the scheme, LINK(q, q') is a boolean function that returns TRUE if there is a straight-line path between q and q' and FALSE otherwise.

A straightforward distribution for generating new configurations is the uniform distribution. Figure 2.4 gives an example of roadmaps generated by the uniform sampling strategy. Although it is very simple, we will show that the uniform sampling strategy is fairly efficient if the configuration space possesses certain geometric properties (Section 4.2).

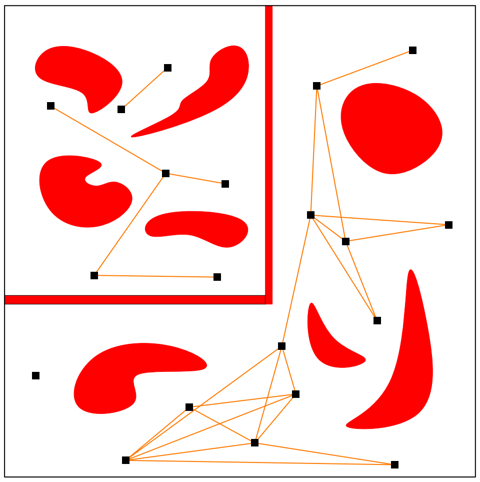

Figure 2.4. A roadmap generated by the uniform sampling strategy for an all-pairs problem in the 2-D configuration space.

The single-source problem Roadmap construction for single-source problems is very similar to that for all-pairs problems. The main difference is that part of the query, the home configuration q_h, is known in advance. The planner inserts q_h into the roadmap G at the beginning, and G contains only milestones that are connected to q_h. As a result, G consists of only one connected component of the configuration space. If q_h lies in a very restricted part of the environment, this modification reduces the size of G and speeds up the roadmap construction.

To process a planning query, the planner tries to connect the other query configuration q to the roadmap in the same way as that in all-pairs problems, and searches the roadmap for a path between q to q_h.

The single-pair problem In both the all-pairs and the single-source problem, we construct the roadmap in the preprocessing stage. Our objective there is to compute a roadmap that captures the connectivity of $\mathcal{C}$ as accurately as possible in a reasonable amount of time. In contrast, there is no preprocessing stage in the single-pair problem. Instead the planner constructs a small roadmap on the fly to answer the query quickly. It does so by sampling only the connected components that contain either the initial configuration q_{init} or the goal configuration q_{goal}. The roadmap for a single-pair problem consists of two trees rooted at q_{init} and q_{goal} respectively (Figure 2.5). The planner samples new milestones at random

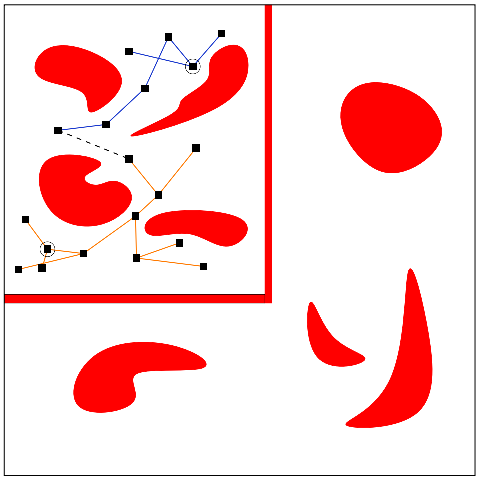
Figure 2.5. A roadmap for a single-pair problem in the 2-D configuration space. The two circles mark q_{init} and q_{goal}.

from $\mathcal{C}$ and inserts them into the trees as milestones, until the two trees "meet", *i.e.*, a milestone in one tree is connected to a milestone in the other. The two trees are constructed in an identical way. The pseudocode below sketches the algorithm for building a tree rooted at a given configuration.

Scheme 2.2 Building a tree T rooted at configuration q_0.

```
1. Insert $q_0$ into $T$.
2. **repeat**
3.   Pick $q$ from $\mathcal{C}$ at random with probability $\pi(q)$.
4.   **if** CLEARANCE$(q) > 0$ **then**
5.     Find some milestone $q' \in G$ such that $q'$ is close to $q$ and $q' \neq q$.
6.     **if** LINK$(q, q')$ **then**
7.       Insert $q$ into $T$ along with an edge between $q$ and $q'$.
```

Note that in contrast to Scheme 2.1, a new configuration is inserted into T only if it can be connected to some milestone already in T. So by construction, there is a path between q_0 and every milestone in T.

In the all-pairs problem, the roadmap G potentially has multiple connected components.

In the single-source problem, G has a single connected components. In a single-pair problem, G contains two trees rooted at q_{init} and q_{goal}. Despite the difference, the key element that determines the efficiency of all these planning schemes is the sampling distribution π for generating new configurations. A distribution good for one type of problems may not be the best for others. For example, the uniform sampling strategy is a viable candidate for the all-pairs problem: since we do not know the query configurations in advance, it seems reasonable to have the milestones uniformly cover $\mathcal{C}$. Uniform sampling can also be used for the single-pair problem, but it is not a good choice because it makes equal amount of effort in the entire configuration space, which may contain many connected components irrelevant to the current query. In the next section, we will look in detail how to construct a more efficient sampling strategy for the single-pair problem.

2.4 Randomized Expansion

Most previous work on randomized path planning is devoted to the all-pairs problem. They can also be easily adapted for the single-source problem. On the other hand, many practical applications require a planner for the single-pair problem, because environments change frequently, and there is not enough time to pre-compute a roadmap for the entire configuration space. In a single-pair problem, although the free space may contain several connected components, at most *two* of them are relevant to the query being processed, and it is clearly undesirable to perform expensive preprocessing to construct a roadmap of the entire configuration space. Instead we would prefer to build only the part of the roadmap that is relevant to the query, *i.e.*, the part that contains only the configurations connected to the query configuration q_{init} or q_{goal}.

Following the scheme in the previous section, our planner constructs two trees of milestones (the roadmap) rooted at q_{init} and q_{goal} respectively. It samples new configurations first in the neighborhoods of q_{init} and q_{goal}, and then iteratively, in the neighborhoods of newly-generated milestones. It stops as soon as the two trees become one connected component, and a path between q_{init} and q_{goal} is then extracted from the roadmap. We call this algorithm *randomized expansion*.

To add a new milestone to a tree T (Figure 2.6), the planner picks at random a milestone q in T with probability $\pi_T(q)$ and samples a new free configuration q' uniformly at random

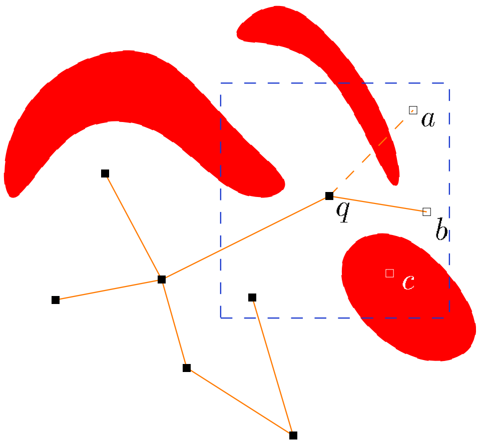

Figure 2.6. Adding a new milestone. The configurations a and c are rejected, but b is accepted as a new milestone.

from the neighborhood of q. If there is an straight-line path between q and q', then q' is inserted into T as a milestone along with an edge between q and q'.

An important issue for our planner is to avoid oversampling any region of the configuration space, especially around q_{init} and q_{goal}. Ideally we would like the milestones to eventually distribute rather uniformly over the connected components containing q_{init} or q_{goal}. To achieve this objective, our planner defines a weight for every milestone q in T. The weight of q is the number of milestones in T within some pre-defined neighborhood of q. The weight function $w(q)$ measures how densely the neighborhood of q has already been sampled, and the planner picks a milestone q in T with probability inversely proportional to the weight of q, *i.e.*, $\pi_T(q) \propto 1/w(q)$. So a milestone with a smaller weight has a greater chance of being picked. Other monotonically decreasing functions of weights are likely to work, too, but $1/w(x)$ is easy to compute and has worked well in our experiments. It also facilitates the analysis of the planner in Chapter 4.

Every time that the planner adds a new milestone q to T, it checks whether there is a straight-line path between q and the milestones in the other tree. If so, a path between q_{init} and q_{goal} is found, and the planner terminates. In practice, it is unlikely that there is a collision-free straight-line path between two milestones far away from each other. To improve the performance, the planner checks the straight-line connection between two milestones only if they are close enough.

Like all other randomized path planners, our planner will not stop if no path exists. We must explicitly set the maximum number of milestones to be sampled. Alternatively we may choose to terminate the algorithm and report that no solution exists if the minimum weight over all the milestones in the two trees exceeds a certain value, because this indicates that we have adequately sampled the configuration space, but are still unable to find a path.

2.5 Implementation Issues

Two primitive operations, CLEARANCE and LINK, are needed to implement the randomized path planners presented in Sections 2.3 and 2.4.

The procedure CLEARANCE(q) computes the approximate distance between a moving object M placed at q and the obstacles. It can be realized in various ways, and there is a trade-off among efficiency, accuracy of results, and generality. At one extreme, CLEARANCE performs only collision detection and returns simply 0 or 1 depending on whether q is in collision. At the other extreme, it computes the exact distance between M placed at q and the obstacles. The cost of computing the exact distance can be quite expensive.

A very fast technique for collision detection is to pre-compute a bitmap of the environment indexed by the configuration of M [LRDG90, Lat91a]. After the pre-computation, it takes only constant time to determine whether a given configuration is in collision by indexing into the bitmap. This technique works well if the environment is planar or can be projected to a plane. If the environment contains only convex polygonal objects, the best theoretical algorithm can compute the exact distance between two objects in poly-logarithmic time after linear time preprocessing [DK90]. In practice, we can often do better by tracking the closest pair of features (vertices, edges, and faces) [LC91, Mir98, GHZ99] as objects move continuously in space. To handle non-convex polygonal objects with complex geometry, most algorithms build hierarchical bounding volumes. The basic idea is to construct successively simpler approximations to the underlying polygonal object. When two objects are far apart, we use the simpler approximations to speed up the computation. When two objects are close together, we travel deeper into the hierarchies and use more detailed approximations, but only in localized regions. Some of these hierarchical algorithms perform only collision detection [GLM96, Hub96]; others can compute the approximate distance as well [Qui94]. Algorithms for non-convex objects may be a little slower than their

Table 2.1. Trade-off among three types of implementation for CLEARANCE.

Algorithm	Input	Output	Query Time
bitmap	planar or $2\frac{1}{2}$-D objects	binary	very fast
feature tracking	convex polygonal objects	exact distance	fast
hierarchical bounding volumes	non-convex polygonal objects	approximate or exact distance	fast

more specialized counterparts for planar or convex objects, but they are nevertheless very efficient in general and are capable of of handling complex environments with hundreds of thousands polygons. The trade-off among the three types of algorithms mentioned above are summarized in Table 2.1.

The procedure LINK determines whether there is a collision-free straight-line path between two configurations. A simple implementation of LINK would discretize the path into a sequence of configurations and regard the path free if all these configurations are free. Problems may arise if the discretization is not fine enough, but they can usually be addressed by leaving some tolerance around the obstacles. A better way is to bisect the straight-line path between its endpoints recursively into two segments until either an endpoint of segments is in collision or the collision-free spheres centered at the endpoints cover the segments completely and thus certify that the segments are collision-free (Figure 2.7) [BKL$^+$97]. The radii of the spheres covering the segment are computed from the distance information returned by CLEARANCE.

The discretization method requires only a collision detection routine in the implementation. In contrast, the recursive bisection method needs some form of distance computation. Collision detection is faster than distance computation; using a collision detection routine reduces the time spent for each collision check. On the other hand, although distance computation takes longer to execute, it provides more information, which is used by the recursive bisection method to reduce the total number of collision checks. Our experience seems to indicate that the second approach works better.

For the randomized expansion planner, we also need to compute the weight function $w(q)$. A naive method to compute $w(q)$ would enumerate all the milestone in the tree T that lie in the neighborhood of q. It takes $O(n)$ time, where n is the total number of

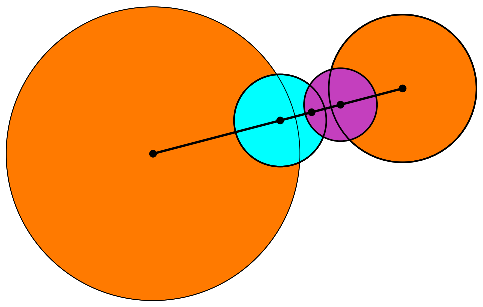

Figure 2.7. Recursively bisecting the straight line segment into two segments until the collision-free spheres centered at the endpoints of segments cover the segments completely or one endpoint is in collision.

milestones in T. The method works only if n is relatively small. As an improvement, we may put a grid on the configuration space, insert milestones into the grid, and then use the grid to locate the milestones close to q. This method works very well for low-dimensional configuration spaces, but the size of the grid grows exponentially with the dimension of the configuration space. Range search techniques [GO97], which has been studied extensively in computational geometry, provide the most efficient way to compute $w(q)$. They typically achieve poly-logarithmic computation time, using a reasonable amount of space, but are more complicated to implement.

2.6 Path Optimization

Paths generated by a randomized path planner typically contain many unnecessary zig-zags, because of the random steps takes by the planner. They cannot be used directly and need to be smoothed or optimized first. We would like to develop an algorithm that is efficient and works for configuration space of arbitrary dimensions.

Research in computational geometry has yielded many fast algorithms for the shortest path problem in 2-D environments under various metrics [GO97, pp. 445–466]. The corresponding problem in three or higher dimensions is considerably harder: the shortest-path problem in 3-D has been proven to be NP-hard [CR87]. Therefore to find a minimum-length path for a moving object with many dofs, we have to resort to approximation techniques. One possibility is to plan a collision-free path first and then deform the path iteratively to reduce its length [SD91]. This is the approach taken here. Alternatively, for a simple object with two or three dofs, one may discretize the configuration space of the moving object and search the space exhaustively for an approximation to the minimum-cost path [DXCR93].

In this section, we describe an efficient algorithm for optimizing the length of a path. The idea is to look at a path at multiple levels of resolution [SDS96] and iteratively replace pieces of the current path by shorter ones. We consider only piecewise-linear paths, but this is not a severe restriction, since any reasonable path can be well-approximated by a piecewise-linear one.

Let us define the length of a straight-line path between two configurations p and q as the Euclidean distance $d(p, q)$ between them. The length of a piecewise-linear path γ with n vertices $v_1, v_2, \ldots, v_n$ is then the sum of the lengths of straight-line paths between the successive vertices on γ: $L(\gamma) = \sum_{i=1}^{n-1} d(v_i, v_{i+1})$.

The Euclidean distance is a metric; so it satisfies the triangle inequality $d(p, q) \leq d(p, r) + d(r, q)$ for any $p, q, r \in \mathcal{C}$. If we replace a portion of a path by a straight-line segment in $\mathcal{C}$, the length of the path can only decrease (or stay the same).

Lemma 2.1 *Let p and q be two points on a path γ. If γ' is a new path obtained by replacing the part of γ between p and q by the straight-line segment between p and q, then $L(\gamma') \leq L(\gamma)$.*

Lemma 2.1 helps characterizing paths of minimum length: there is a minimum-length path that is locally "straight" at every point where the path is not touching the obstacles. More precisely, let $\widehat{\gamma}$ be a collision-free path of minimum-length, and let Γ be the set of collision-free paths such that for every $\gamma \in \Gamma$, γ has zero curvature at every point p on γ where γ is not tangent to obstacles in the configuration space. If $\widehat{\gamma}$ is not in Γ, we can repeatedly replacing sub-paths of $\widehat{\gamma}$ by straight-line paths and in the limit reach a path $\widehat{\gamma}' \in \Gamma$. By Lemma 2.1, the length of $\widehat{\gamma}'$ cannot be longer than that of $\widehat{\gamma}$. Hence the following lemma.

Lemma 2.2 *The set Γ contains a collision-free path of minimum length.*

One way to optimize $\gamma = (v_1, v_2, \ldots, v_n)$ is then to break γ recursively into two sub-paths $\gamma_1 = (v_1, v_2, \ldots, v_{n/2})$ and $\gamma_2 = (v_{n/2}, v_{n/2+1}, \ldots, v_n)$, and check whether γ_1 and γ_2 can be replaced by straight-line paths. If they can, Lemma 2.1 guarantees that the length of the new path is shorter than that of γ. We call this recursive procedure SHORTCUT.

The procedure SHORTCUT is reasonably efficient, taking $O(n)$ time to execute, but unfortunately it may stop far short of reaching the minimum-length path. In the example

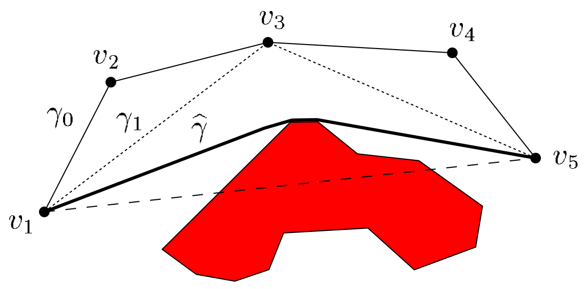

Figure 2.8. SHORTCUT gets stuck at γ_1, which is far from the minimum $\widehat{\gamma}$. The path γ_0 is the input to SHORTCUT, and γ_1 shows the path after one iteration.

shown in Figure 2.8, it terminates after reaching the path γ_1. No further improvement is possible because the straight-line path (v_1, v_5) is in collision. Of course, if there were more vertices on γ_1, we might be able to reach the true minimum-length path, but it is difficult to know in advance how many vertices are needed.

To address this problem, after SHORTCUT stops, we use an oracle to scan through each vertex v_i in the path and add additional vertices around v_i if necessary. Specifically the oracle puts two additional vertices v_l and v_r on the straight-line segments (v_{i-1}, v_i) and (v_i, v_{i+1}) respectively and try to replace the sub-path (v_l, v_i, v_r) by the straight line segment (v_l, v_r). A bisection method is used to determine v_l and v_r so that (v_l, v_r) is collision-free. First set v_l to be the midpoint of (v_{i-1}, v_i), and v_r to be the midpoint of (v_i, v_{i+1}). If the line segment (v_l, v_r) is not collision-free, bisect again and set v_l to be the midpoint between v_i and the previous v_l, and v_r to be the midpoint between v_i and the previous v_r. Continue until (v_l, v_r) lies completely in the free space. In general, $v_l = (1/2^k)v_{i-1} + ((2^k - 1)/2^k)v_i$ and $v_r = (1/2^k)v_{i+1} + ((2^k - 1)/2^k)v_i$, at the *k*th step for $k = 1, 2, \ldots$. The procedure is guaranteed to terminate because v_i is a free configuration and hence there exists an open ball $\mathcal{B}$ that contains v_i and lies entirely in the free space. Once both v_l and v_r are inside $\mathcal{B}$, the line segment between them must be collision-free, because $\mathcal{B}$ is convex. After the oracle adds additional vertices to the path, SHORTCUT is invoked again. The process terminates when no further improvement is possible. We call SHORTCUT with an oracle ADAPTIVE-SHORTCUT.

The first iteration of ADAPTIVE-SHORTCUT calls only SHORTCUT and all later iterations perform the oracle computation followed by SHORTCUT. A computed example is shown in

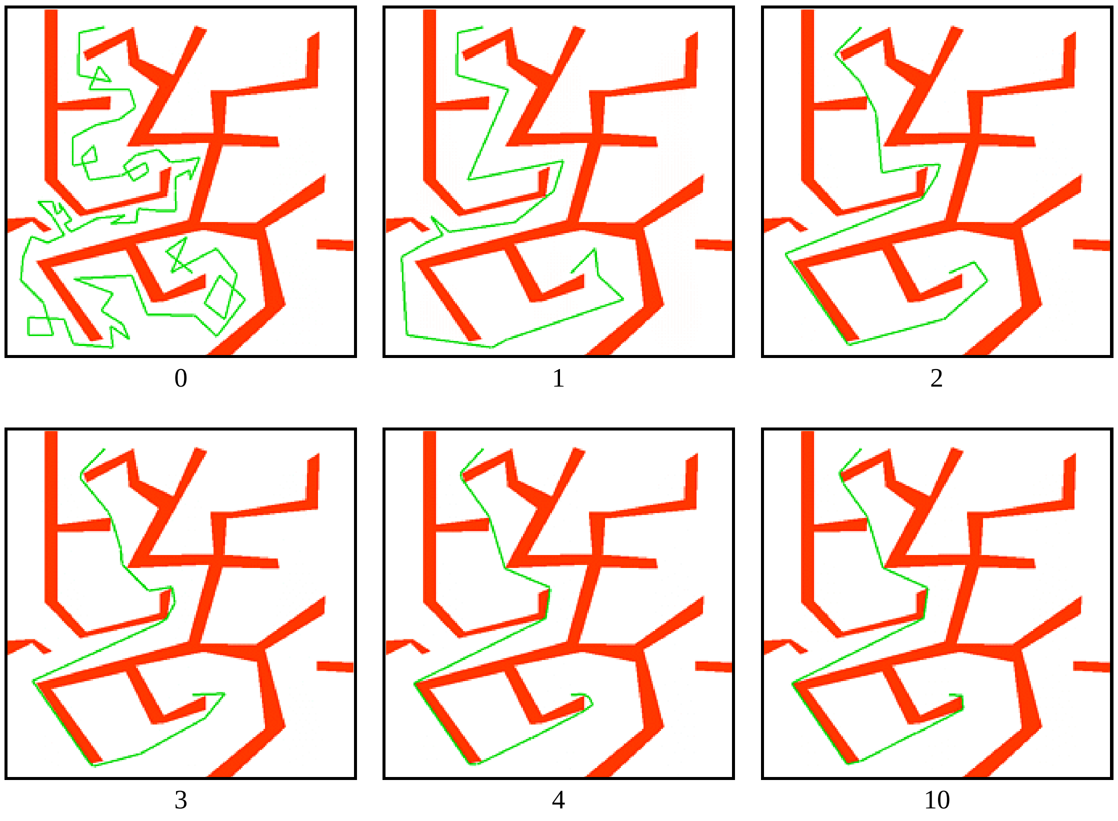

Figure 2.9. Minimum-length paths computed by ADAPTIVE-SHORTCUT after 0, 1, 2, 3, 4, and 10 iterations.

Figure 2.9. We have verified analytically that in this example, the optimized path obtained after 10 iterations is almost exactly the same as the true minimum. Note that the result after the first iteration is the local minimum that we would have reached without the oracle; as shown in the figure, it is quite some distance away from the true minimum.

We can shed some light on the efficiency of ADAPTIVE-SHORTCUT by analyzing the space of paths that it operates on. Let F_i be the space of piecewise-linear paths having i vertices. Any path with i vertices can also be represented by a path with j vertices for $j \geq i$. Hence $F_i \subset F_j$ for $i \leq j$, and $F_2, F_3, \ldots$ form a sequence of nested function spaces. Let $F = \bigcup_{i=2}^{\infty} F_i$ be the space of all piecewise-linear paths. We are interested in $\widehat{\gamma}$, the optimal path in F. If we restrict the space of paths being considered to F_i for some fixed i, then $\widehat{\gamma}_i$, the optimal path in F_i, remains a good approximation to $\widehat{\gamma}$ provided i is sufficiently large. However, a large i means more vertices (variables) needed to represent

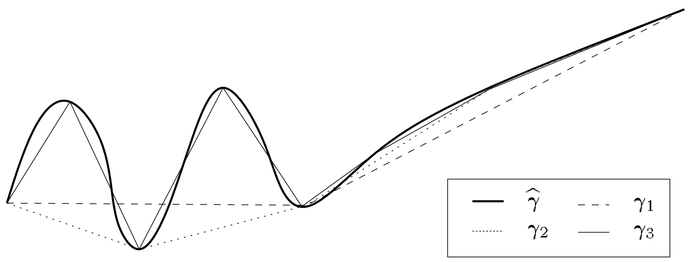

Figure 2.10. Piecewise linear approximations to a smooth path. The paths $\gamma_1, \gamma_2, \gamma_3$ are three-, five-, and nine-vertex piecewise-linear paths approximating $\widehat{\gamma}$. In the right portion of $\widehat{\gamma}$, all approximations are reasonably good, while in the left portion of $\widehat{\gamma}$, only γ_3 approximates $\widehat{\gamma}$ well.

a path, and thus the optimization procedure may take longer to converge to a minimum. On the other hand, if i is too small, $\widehat{\gamma}_i$ may be a poor approximation to $\widehat{\gamma}$. Furthermore different portions of $\widehat{\gamma}$ may have different levels of smoothness. On the part where $\widehat{\gamma}$ is smooth, a few vertices are enough to approximate it well; on the part where $\widehat{\gamma}$ varies widely, many more vertices are needed. See Figure 2.10 for an example in two dimensions. In ADAPTIVE-SHORTCUT, the SHORTCUT procedure removes unnecessary vertices when replacing sub-paths by straight line segments, and the oracle adds more vertices where needed. By moving up and down among various spaces $F_2, F_3, \ldots$, ADAPTIVE-SHORTCUT quickly converges to a good approximation to $\widehat{\gamma}$.

Since our algorithm iteratively makes small modification of the current path, it can still be stuck in a local minimum. To make the algorithm more robust, we can have the randomized path planner return multiple paths, optimize each of them separately, and keep the best result.

The correctness of our algorithm depends only on the triangle inequality. Thus it can be applied to other measures of path cost, provided that the triangle inequality is satisfied. Consider, for example, the time that it takes a c-joint robot manipulator to execute a piecewise-linear path. If the manipulator has maximum speed $\nu_1, \nu_2, \ldots, \nu_c$ for the joints, the time that it needs to travel along a straight-line path in $\mathcal{C}$ between two configurations $p = (p_1, p_2, \ldots, p_c)$ and $q = (q_1, q_2, \ldots, q_c)$ is then the maximum of times required by each joint

$$\tau(p, q) = \max_{1 \le i \le c} \frac{|p_i - q_i|}{\nu_i}.$$

Again the cost of a piecewise-linear path γ is the sum of the cost of straight-line paths between successive vertices of γ. The function τ is a weighted L_∞ metric on $\mathcal{C}$, which of course satisfies the triangle inequality.

2.7 Experiments with Articulated Objects

In this section and next, we apply the randomized expansion algorithm to two common types of moving objects, articulated robot manipulators and free-flying rigid bodies, in 3-D environments. Our goal is two-fold: (i) demonstrate the generality of our planner by implementing it for different types of moving objects; (ii) demonstrate the planner's ability to solve difficult path planning problems in complex geometric environments. Our algorithm is implemented in C++. The running times reported in these two sections were obtained on an SGI Octane workstation with a 270 MHz MIPS R12000 processor and 256 MB memory.

2.7.1 Implementation Details

First let us look at articulated robot manipulators that consist of rigid links connected together sequentially by one-dof revolute joints. Each joint of the robot has a limited range due to mechanical constraints.

Configuration space representation The configuration of a manipulator is specified by a list of joints angles $(\theta_1, \theta_2, \ldots, \theta_c) \in \mathrm{R}^c$, where c is the total number of joints. Since the joint angles have limited ranges, the configuration space is a bounded rectangular region in R^c.

Sampling free configurations To sample a new configuration q' in the neighborhood of a given configuration q, we choose each coordinate of q' independently by picking a value uniformly at random from a small interval centered at the corresponding coordinate of q. The sizes of these intervals are determined by pre-selected constants. We keep the new configuration if it is collision-free.

Computing the weight function Our current implementation computes the weight of a milestone q by simply enumerating all the milestones that lie in the neighborhood of q. Since the computation of CLEARANCE dominates the running time of the algorithm, the crude implementation of weight function calculation does not cause a severe problem, as long as the number of milestones are reasonably small (say, a few thousands). Of course, more efficient techniques (see Section 2.5) will improve the performance of our algorithm in complex problems where a huge number of milestones are needed.

Implementing CLEARANCE Quinlan's distance computation algorithm [Qui94] is used to implement CLEARANCE. We build one spherical bounding hierarchy for each rigid link of a manipulator and another one for all the obstacles. To calculate the minimal distance from the manipulator to the obstacles, we invoke Quinlan's algorithm between each rigid link and the obstacles, and keep the smallest distance returned.

Implementing LINK In our current implementation for articulated objects, the fixed-resolution discretization method is used for LINK.

2.7.2 Results

We tested the planner on a large number of data sets. Figures 2.11–2.13 show four of them. Each figure contains several query configurations as well as computed examples.

The three robots used in the tests are similar. They all have six dofs and kinematic structures identical to that of the famous PUMA robot, but they differ in their shape. The robot in scene 3 has a large end-effector that is difficult to maneuver. The robot in scene 4 is very skinny so that it can navigate in narrow spaces.

The four test scenes vary in their complexity. Scenes 1 and 2 contain about twice as many triangles as the other two (Figure 2.11). In addition, scene 2 contains three PUMA robots with a total of 18 dofs. Scene 3 consists of an arrangement of horizontal and vertical bars set up around the robot to restrict its movement (Figure 2.12). There are holes of various sizes in the scene, which may trap the large end-effector of the robot. Scene 4 contains a cluster of obstacles placed closely to one another (Figure 2.13). To reach the goal configuration, the robot has to travel through narrow spaces between the obstacles.

For every query, we ran our planner 30 times independently with different random seeds. The test results are reported in Table 2.2. The numbers shown are the average over 30 runs. Column 2 of the table shows the dimension of the configuration space for every test scene. Column 3 lists the total number of triangles, including the robot and the obstacles. It gives some rough idea of the complexity of input geometry. Column 4 specifies the query configurations. Columns 5–8 give the average running times, the average number of calls

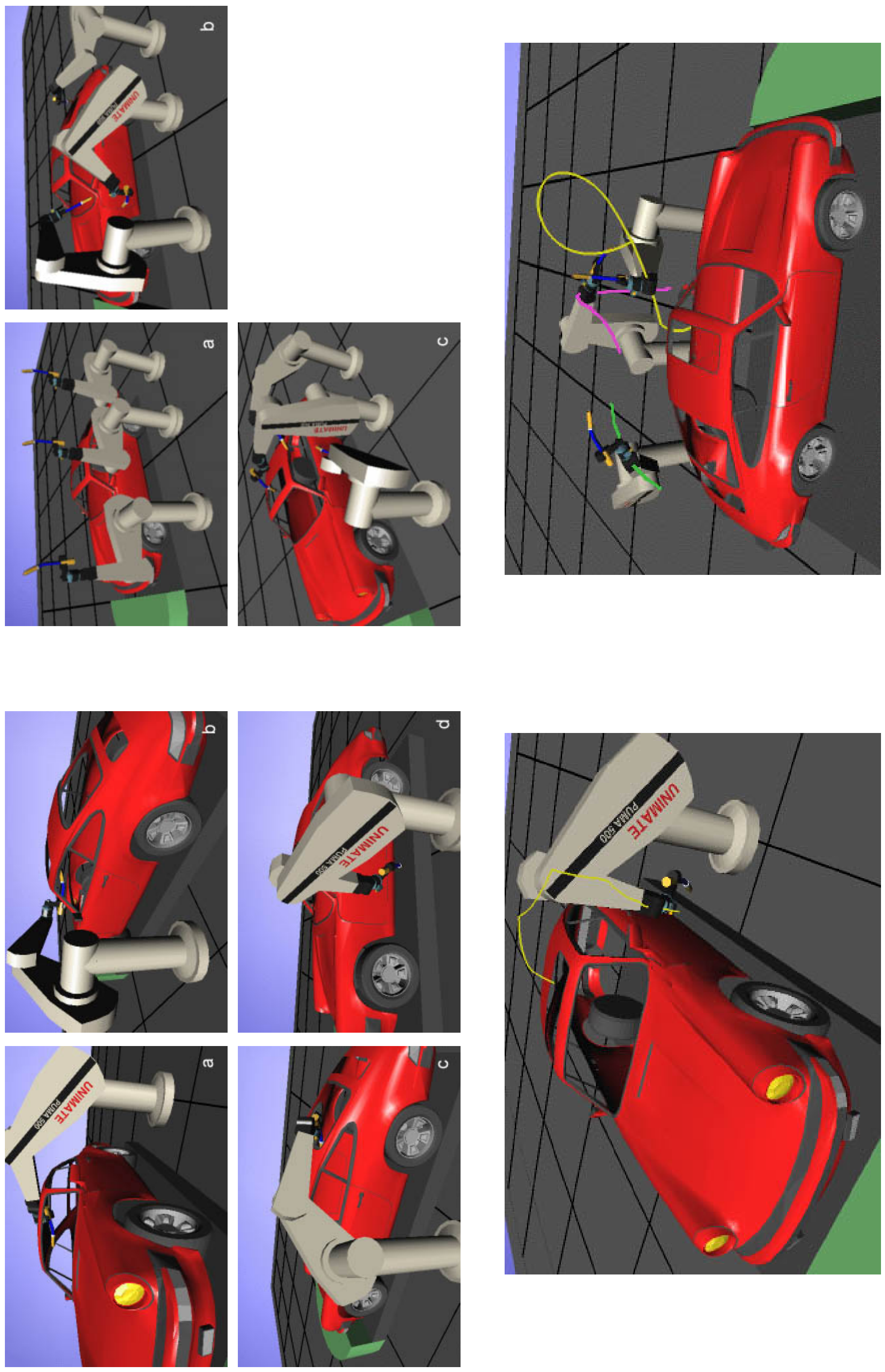

Figure 2.11. Test scenes 1 and 2 for articulated objects. Scene 1: A PUMA robot performs welding operations on a car (left). Scene 2: Three PUMA robots with 18 dofs in total perform welding operations together (right).

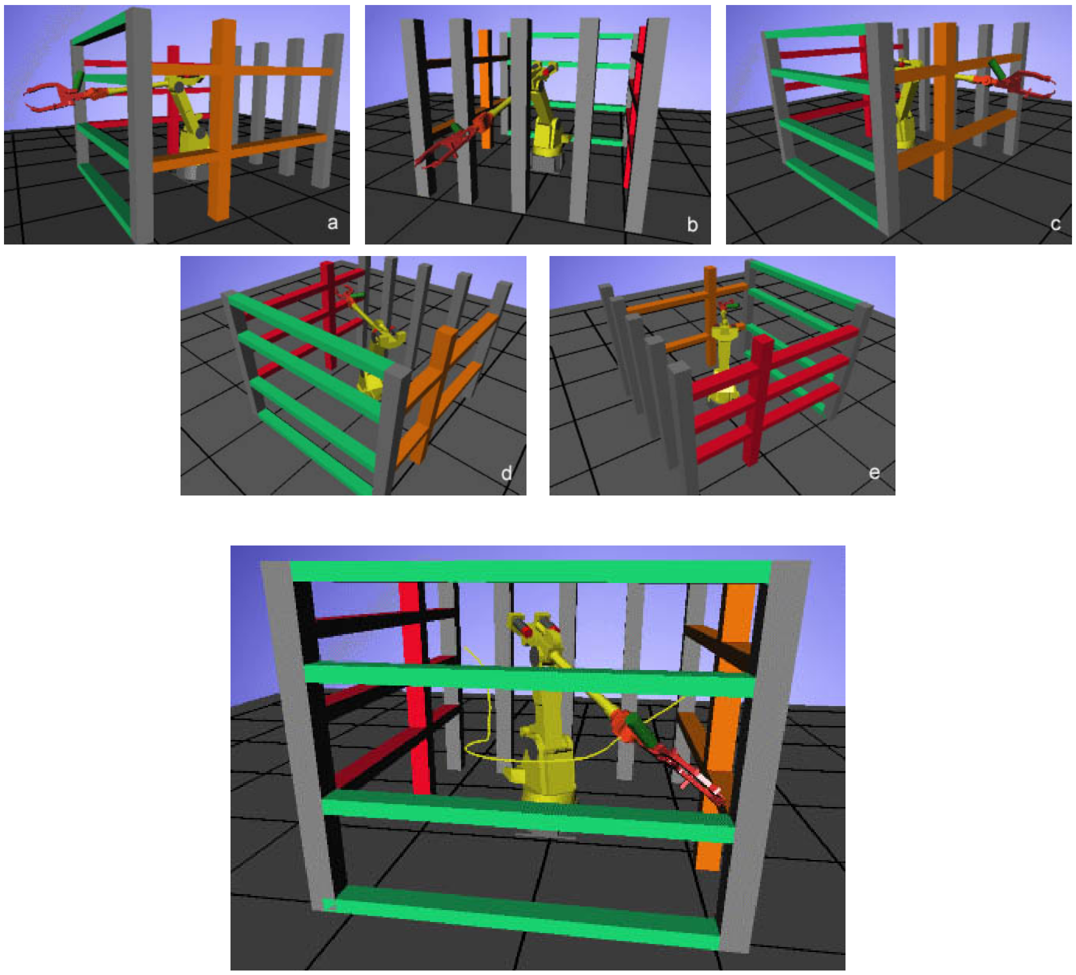

Figure 2.12. Test scene 3 for articulated objects. The horizontal and vertical bars acting as obstacles restrict the movement of the FANUC-like robot, which has a large end-effector.

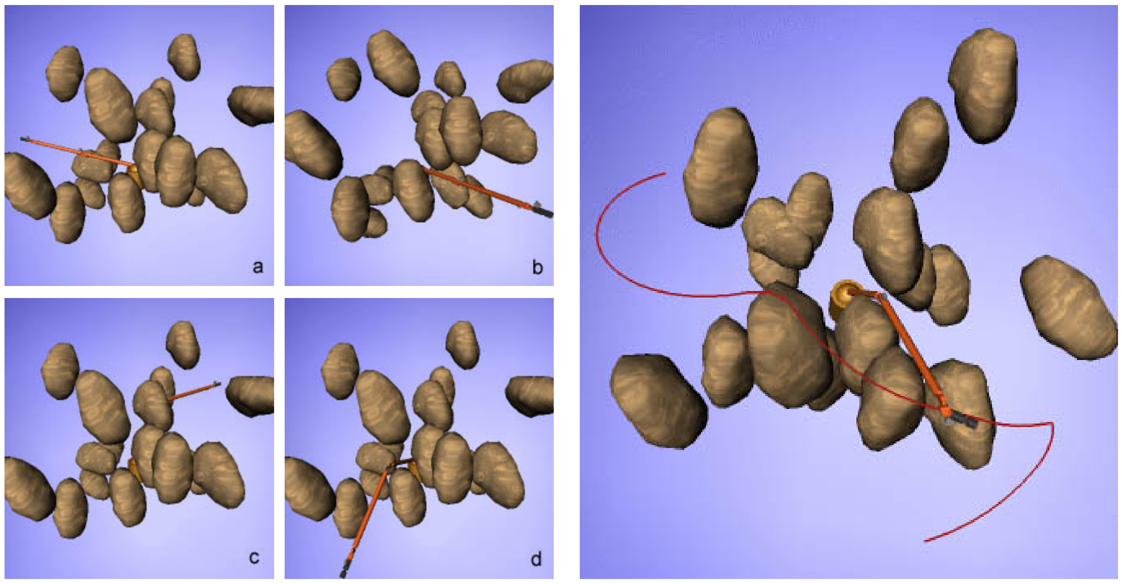

Figure 2.13. Test scene 4 for articulated objects. The skinny robot manipulator arm passes through narrow spaces between the obstacles.

Table 2.2. Performance statistics of the planner on articulated objects.

Scene	Dim.	N_{tri}	Query	Time (sec.) mean	Time (sec.) std	N_{clear} mean	N_{clear} std	Rejection Rate (%)	N_{mil}	N_{link}
1	6	11724	a,b	0.97	0.73	1024	802	38.5	15	35
			a,c	5.91	6.32	6309	6806	11.7	88	108
			a,d	2.34	2.08	2171	2008	39.3	51	86
			b,d	2.64	2.85	2379	2641	35.4	50	106
2	18	15758	a,b	35.07	30.47	8788	7701	40.7	45	81
			b,c	72.62	42.23	18235	10698	60.4	70	144
3	6	4697	a,b	4.90	3.41	4753	3324	46.4	346	875
			b,c	14.91	9.78	13418	8811	41.6	398	2636
			d,e	2.91	2.79	2524	2476	39.9	229	361
			b,e	3.63	2.29	3048	1954	40.0	276	456
4	6	5887	a,b	13.63	10.38	12152	8987	33.7	596	1202
			a,c	12.70	10.84	11419	9316	36.0	588	1146
			b,d	14.77	11.52	12710	9808	36.0	473	1797

N_{tri} : number of triangles	N_{clear}: number of calls to CLEARANCE
N_{mil}: number of milestones	N_{link} : number of calls to LINK

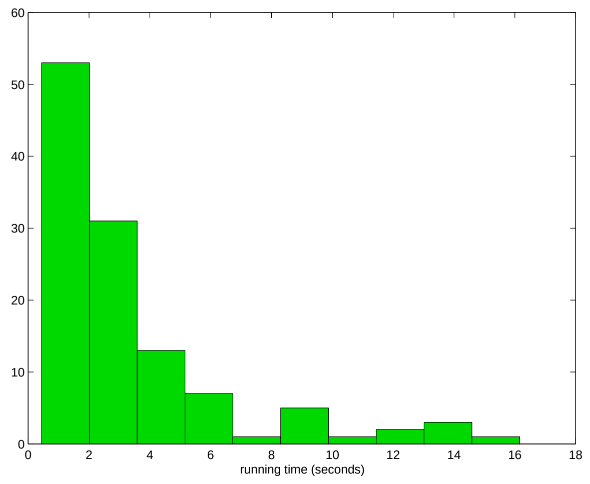

Figure 2.14. A histogram of more than 100 independent runs for a particular query. The mean of running times is 3.4 seconds, and the four quartiles are 1.5, 2.1, 3.8, and 16.1 seconds.

to CLEARANCE, and their standard deviations. The running time ranges from a few seconds in simple cases to tens of seconds in more difficult ones. Given the difficulty of the test problems and the fact that there is no pre-computation, these are very good results. Queries in both scene 2 and scene 4 take much longer to execute than the others, but for different reasons. In scene 2, the cost of every collision checking is much higher, because we have to check collision not only between the robots and the obstacles, but also among the robots themselves. In scene 4, the skinny robot arm has to squeeze through narrow spaces between the obstacles in order to reach the goal, a very difficult scenario for all path planners based on random sampling. Column 9 shows the percentage of randomly-sampled configurations rejected due to collision with the obstacles. The data indicate that our planner is very efficient: on the average, it takes roughly two samples to obtain a free configuration. Columns 10 and 11 give the total number of milestones generated in the two trees and the number of calls to LINK respectively. Despite that the robots have six or higher dimensional configuration spaces, a few hundreds milestones are sufficient to answer the queries. These results, once again, confirm the effectiveness of random-sampling techniques for path planning in high dimensional configuration spaces.

The standard deviations shown in Table 2.2 seem to be very large. In some cases, they even exceed the means, which is very disturbing. It turns out that the distribution of running times is not Gaussian. Figure 2.14 plots a histogram of more than 100 independent runs for a particular query. The shape of the distribution is typical of all the tests that we have performed. In most runs, the execution time of the planner is below the mean or slightly above. This indicates that our planner performs well most of the time. The large deviation is caused by very long execution time (as much as four or five times the mean) in a few runs, as indicated by the long and thin tail of the distribution. Further work is needed to reduce the occasional long execution time caused by random variations.

2.8 Experiments with Rigid Bodies

Now let us consider another important class of moving objects, free-flying rigid bodies in 3-D environments.

2.8.1 Implementation Details

Free-flying rigid bodies have six dofs, just as the manipulators that we have seen in the previous section. However, they can rotate in 3-D space arbitrarily, which makes the implementation more difficult. Choosing a good representation for the 3-D rotation space $\mathrm{SO}(3)$ is a crucial aspect of the implementation.

Configuration space representation The configuration of a free-flying rigid body has a translational and a rotational component. The translation is specified by the position of a reference point on the rigid body. The rotation is specified by a unit quaternion. Compared to other choices, such as Euler angles and matrices, the quaternion representation best captures the topology of the 3-D rotation space. Furthermore it is low in memory usage and is robust against floating point errors. Details on representing 3-D rotation using quaternions are available in [Sho85].

Sampling free configurations Suppose that we would like to sample a new configuration q' in the neighborhood of a configuration q. Let x' be the translational component of q'. Since x' is simply a point in R^3, we can sample x' in the same way as that in Section 2.7. Sampling the rotational component is a little more tricky. Let θ and a unit vector $\vec{n}$ be the angle and the axis of the rotational component of q. To sample the rotational component of q', we first pick θ' in a small interval centered at θ, and then sample a unit vector $\vec{n}'$

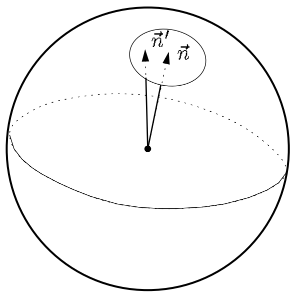

Figure 2.15. Sampling a point $\vec{n}'$ on the unit sphere in the neighborhood of $\vec{n}$.

uniformly at random in a small area centered at $\vec{n}$ on the unit sphere so that the angle between $\vec{n}$ and $\vec{n}'$ lies in a small interval (Figure 2.15).

Computing the weight function The weight function is computed the same way as that described in the previous section.

Implementing CLEARANCE We convert the unit quaternion into a rotation matrix and send the matrix along with the translation vector to Quinlan's algorithm to calculate the distance between the rigid body and the obstacles.

Implementing LINK Our implementation for free-flying rigid bodies uses recursive bisection to check whether the straight-line path between two configurations is collision free. Here the quaternion representation offers significant advantage over other representations. After normalization, the straight-line interpolation between two unit quaternions u and u' gives the minimal rotation needed to bring an object with orientation u to u'. It would be much more troublesome if we used another representation for rotation. For example, matrices obtained by straight-line interpolation of two rotation matrices do not represent rotation in general, and there is no natural normalization that can be applied to correct the problem.

2.8.2 Results

Figures 2.16–2.18 show three test scenes, each with several query configurations and a computed example. The moving objects are free-flying rigid bodies translating and rotating in 3-D, and thus each has six dofs. Scene 1 contains the same arrangement of obstacles as

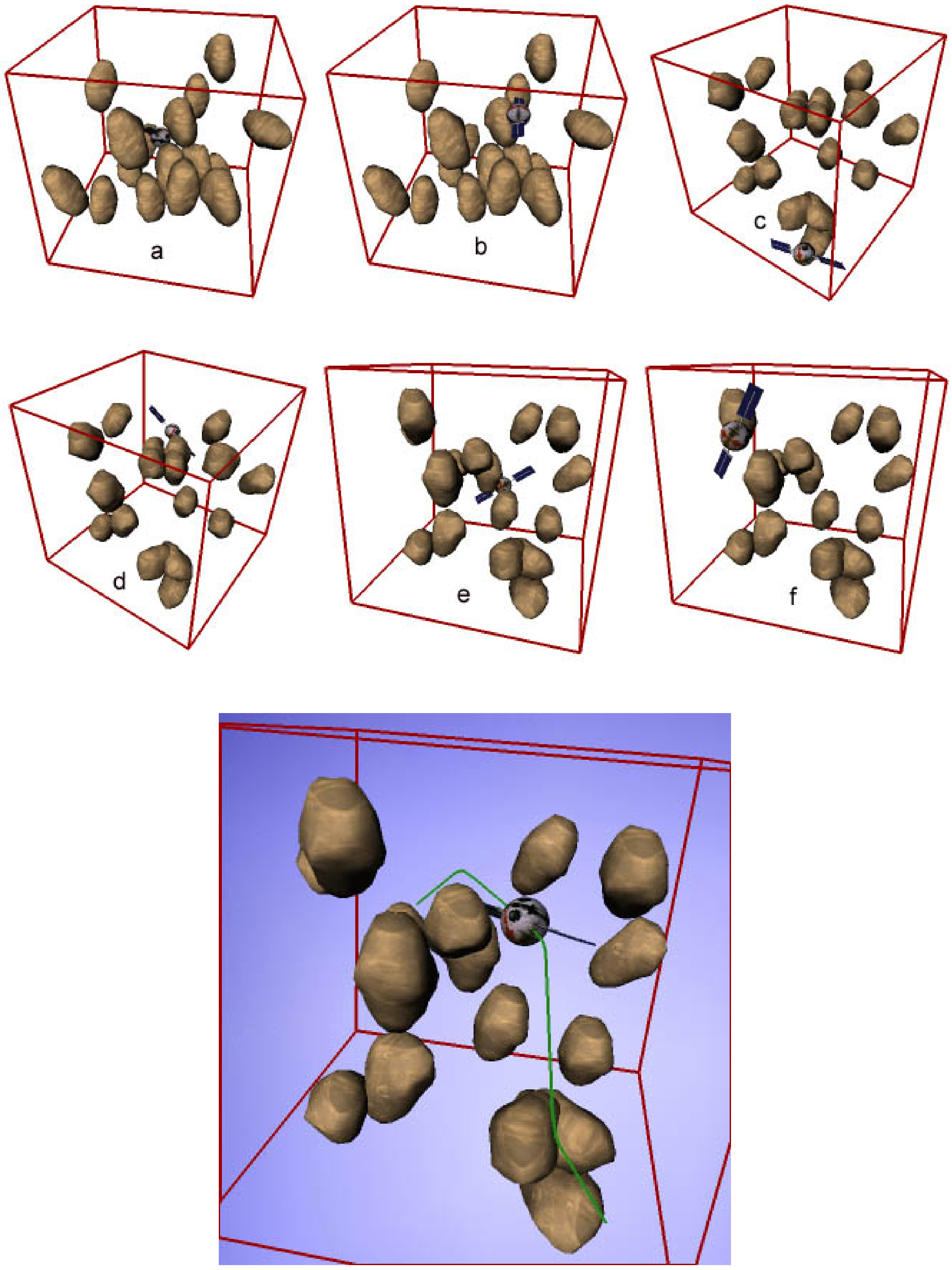

Figure 2.16. Test scene 1 for rigid bodies: a satellite flying among “asteroids”.

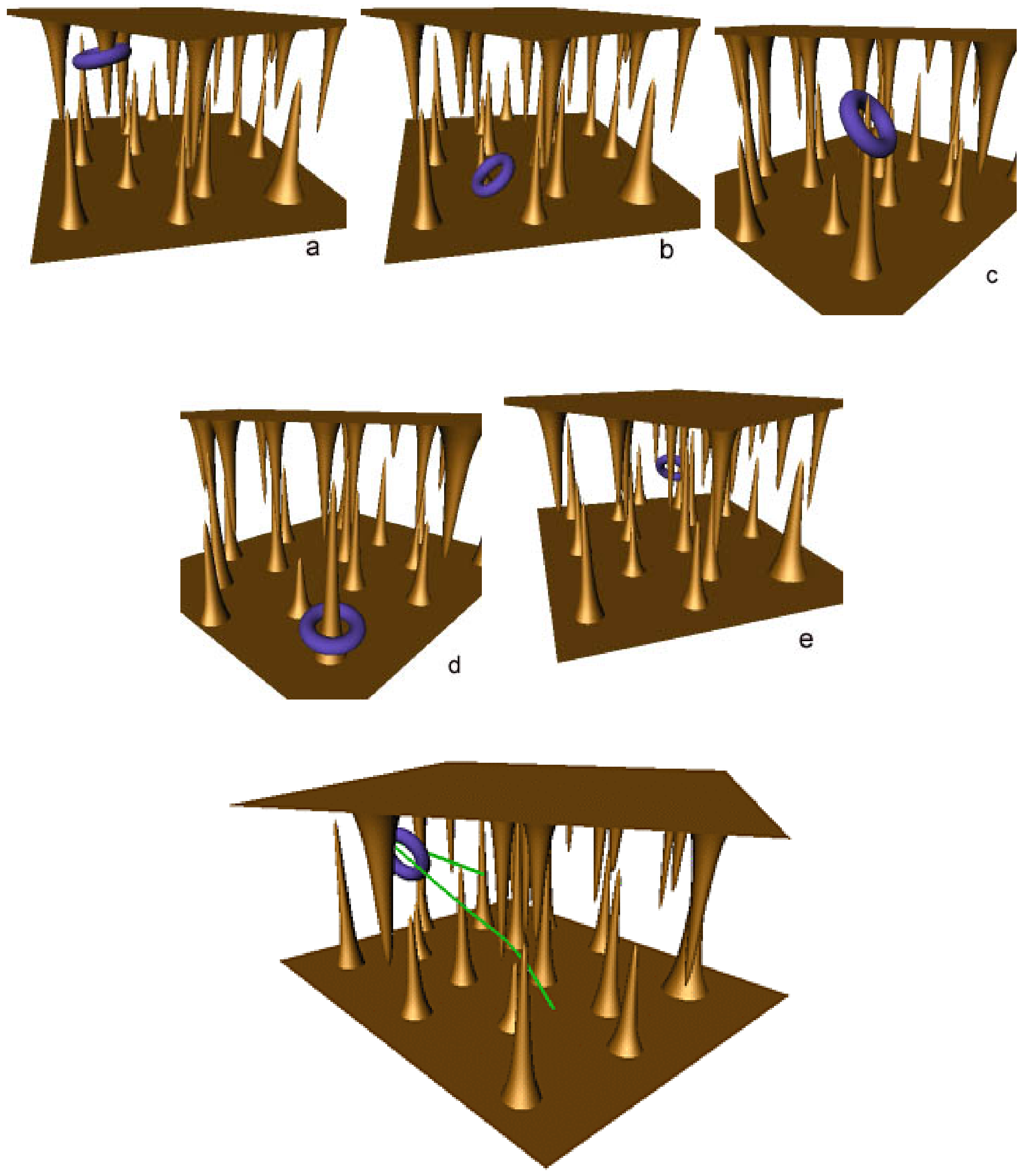

Figure 2.17. Test scene 2 for rigid bodies: a torus-shaped object moving in a cave-like environment.

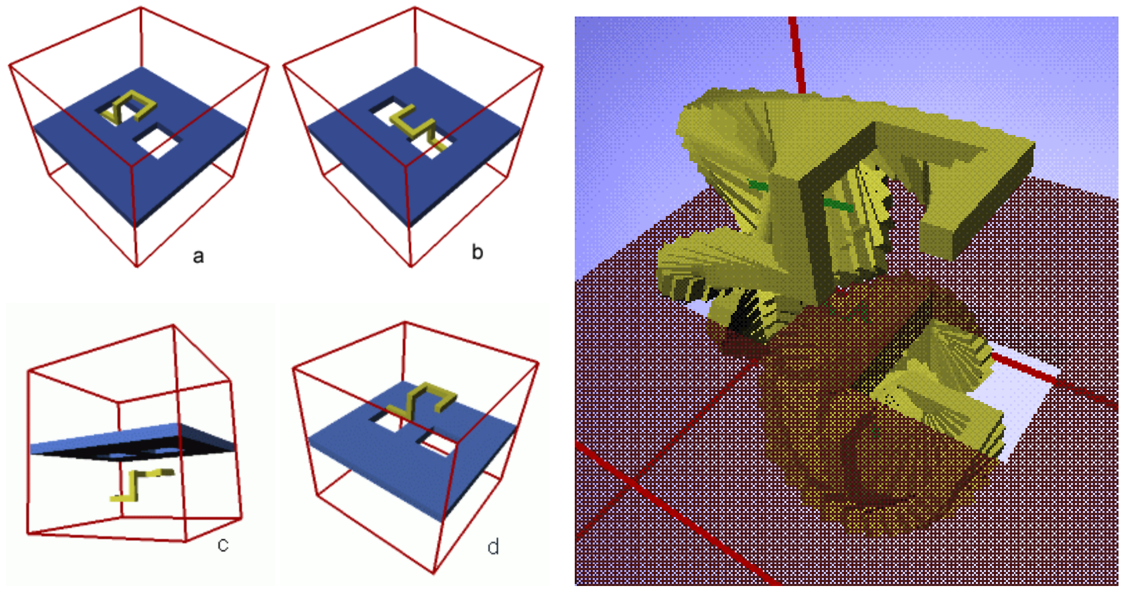

Figure 2.18. Test scene 3 for rigid bodies: a snake-shaped object passing through the small holes in the obstacle.

that in Figure 2.13, but in this case, there is no narrow passage in the configuration space, because the free-flying rigid body is too big to fit through any of the narrow spaces between the obstacles (Figure 2.16). As a result, the running times of the planner are much shorter. In scene 2, a torus-shaped object moves among spikes sticking out from the top and the bottom (Figure 2.17). In scene 3, a snake-like object has to pass through holes in a wall (Figure 2.18). The shape of the object and the relatively small size of the holes together make it difficult for the snake to maneuver through the holes. This results in the longer time in one of the queries (see Table 2.3).

Table 2.3 reports the results of our experiments. The layout of the table is similar to that of Table 2.2. The running times are a few seconds in almost all the cases except the last one, in which the snake-like object has to go through a small hole from one side of the wall to the other. Comparing the running times for scene 2 and scene 3, we see that although scene 2 contains a lot more triangles, the running time of the planner is not much longer. Thus two remarks are in order. First, the hierarchical distance computation algorithm has performed very well for our problems and kept the cost of collision checking under control. Second, to a large extent, the running time of the planner depends more on the difficulty of

Table 2.3. Performance statistics of the planner on free-flying rigid bodies.

Scene	N_{tri}	Query	Time (sec.)		N_{clear}		Rejection Rate (%)	N_{mil}	N_{link}
			mean	std	mean	std			
1	5192	a,b	0.29	0.20	288	234	38.2	14	47
		c,d	1.13	0.91	1118	968	26.6	49	302
		e,f	1.14	1.40	990	1222	34.6	36	274
2	19160	a,b	1.10	1.20	697	974	42.0	46	109
		c,d	4.19	3.52	2853	3053	30.9	229	697
		b,e	4.02	4.40	2099	2473	33.4	84	306
3	120	a,b	3.66	3.91	5277	5735	56.2	360	1069
		c,d	9.42	3.88	14356	3886	78.1	1108	3013

N_{tri} :	number of triangles	N_{clear}:	number of calls to CLEARANCE
N_{mil}:	number of milestones	N_{link} :	number of calls to LINK

final motion required than on the number of triangles in the scene.

2.9 Additional Experiments

To further evaluate the effectiveness of our path planner, we applied it to two practical problems: assembly maintainability checking and motion synthesis for animated characters.

In assembly maintainability studies, we would like to know whether there exists a collision-free path to remove a specified component from an assembly of mechanical parts (see Figure 2.19b). Maintainability is an important issue in mechanical design [CL95]. Engineers must ensure that the required paths exist to remove certain critical parts for routine maintenance or replacement. In the past, this has been a labor-intensive task, accomplished manually by manipulating physical mock-ups or CAD models. Furthermore maintainability must be verified every time a design change affects the geometry of the assembly. Engineers thus find it very attractive to have an interactive tool that performs assembly maintainability checking automatically as they make changes.

Not surprisingly, an efficient path planner is a good candidate for this task. We model the part to be removed as a free-flying rigid body M in a 3-D environment and treat the rest of the assembly as obstacles. The input to the planner is CAD data describing the geometry of the assembly. The environment usually consists of tens of thousands of

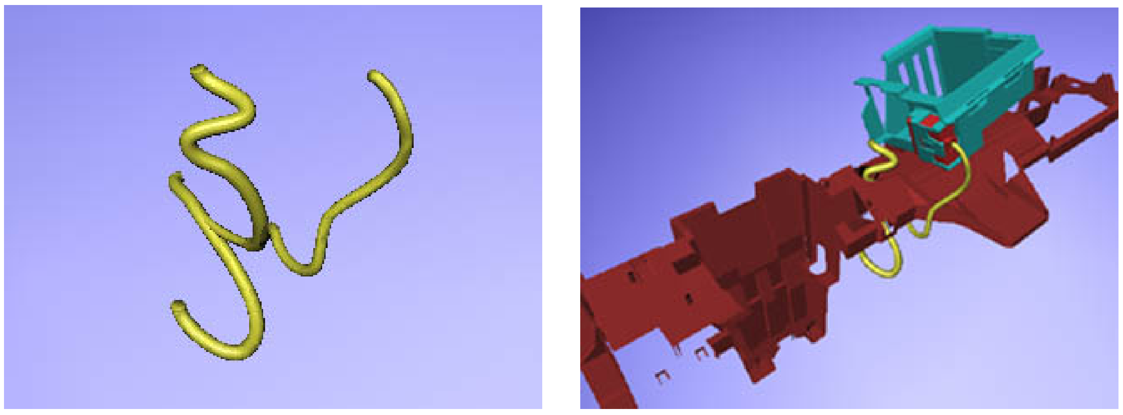

Figure 2.19. An example of assembly maintainability checking. (a) The part to be removed. (b) The part in the installed position.

polygons and is very cluttered due to designers' desire to pack everything into limited space. The initial configuration $q_{\rm init}$ of M is its installed position, and the goal $q_{\rm goal}$ is some arbitrary configuration that detachs M from the rest of the assembly. Since $q_{\rm goal}$ is totally unconstrained, to improve the speed, the planner builds only one tree of sampled milestones from $q_{\rm init}$.

We tested our planner on several real-life data sets from the automotive industry. These data sets contain complex CAD models that describe cluttered environments. A typical problem that we have attempted has about 20,000 triangles, and the planner can solve the problem in about 4 to 10 minutes on an old SGI Crimson workstation with one 100 MHz MIPS R4000 processor and 256 MB of memory. Two of the problems that our planner solved are particularly interesting. In one case, the planner needs to take out an oil pan below a car engine without colliding with the long protrusion underneath the engine and other parts around the engine. In the other case, a pipe behind the dashboard needs be removed (Figure 2.19). The pipe has three branches. A slight movement from its installed configuration may result in one or more of its branches colliding with the parts nearby. Due to the special geometric arrangement of these two assemblies, the parts to be removed must execute complicated maneuvers in order to clear all the obstacles (Figure 2.20). The planner solved the first problem in 386 seconds and the second problem in 405 seconds. The number of calls to CLEARANCE were 4257 and 7822, respectively. The largest example

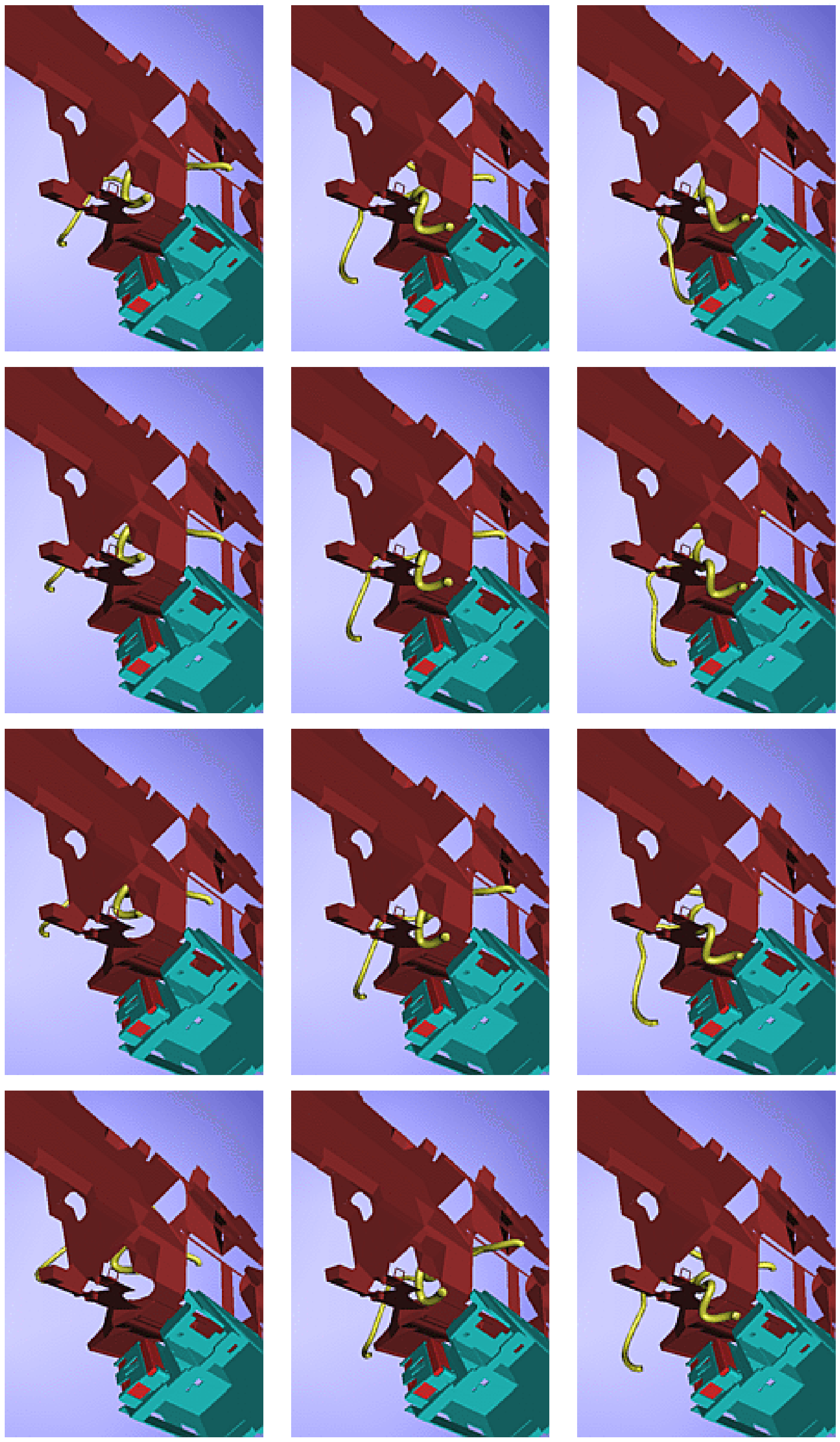

Figure 2.20. A collision-free path for removing the pipe.

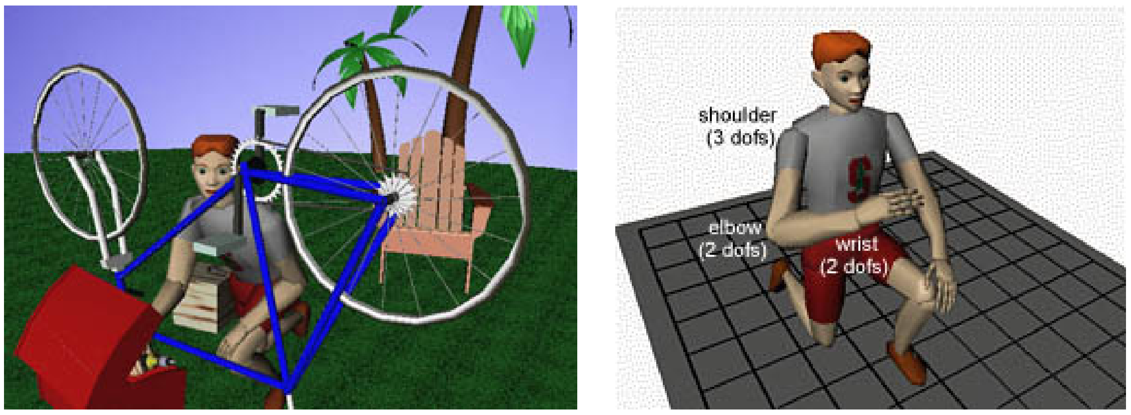

Figure 2.21. An articulated model of a human character.

that we have run contains about 200,000 triangles. The objective is to remove the casing of the transmission mechanism, clearing the dashboard and the shift stick. The planner found a path in about 35 minutes.

Among the problems that we have worked on, there is one case in which the planner failed to find a path after running for more than eight hours, but we were unable to determine whether a path actually exists or not.

Another interesting application of motion planning is motion synthesis for animated human characters. Generating natural-looking motion for characters under high-level control is an important issue in computer animation. This is a very challenging problem, because the model of a human character has complex, multi-jointed structure with many dofs.

Motion planning provides a new tool, which allows the user to specify motion with task-level commands such as “pick up the apple on the table”. Motion planners automatically handle the complex geometric interaction between the character and the environment, and allow the user to specify *what* to do rather than *how* to do it. We illustrate this with a simple example, in which a cyclist extends his arm to get a tool from a tool box (Figure 2.21a). The arm is modeled as a 7-dof kinematic chain (Figure 2.21b): three dofs at the shoulder, two at the elbow, and two at the wrist. The joint angles are restricted in order to generate motion that appears natural. A short motion sequence produced by our path planner is shown in Figure 2.22. The snapshots depict a cyclist reaching out and putting his arm through the

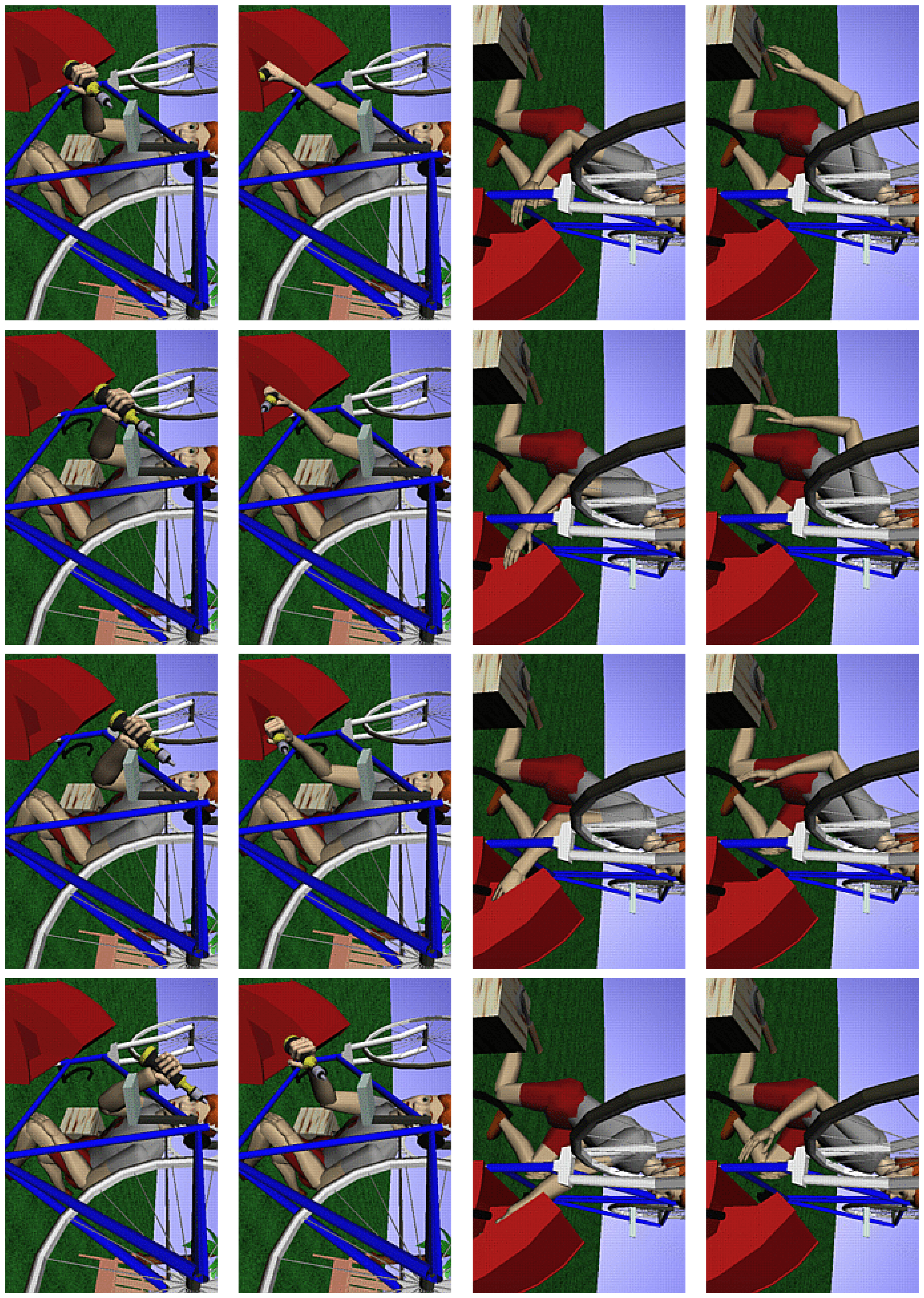
Figure 2.22. Sample motion computed by the path planner. The cyclist reaches for a tool through the bicycle frame and puts it in the specified position.

bicycle frame. He then grasps a screw-driver in the tool box and places it at the final configuration. Three input configurations are given to the planner: q_{init}, q_{grasp}, and q_{goal}. The grasp configuration q_{grasp} specifies the posture of the arm right before the hand grasps the screw-driver. The final motion is obtained by concatenating the results of two queries, one from q_{init} to q_{grasp} and another one from q_{grasp} to q_{goal}. The environment, including the character, the bike, and the tool box, contains about 11,000 triangles. The planner computed the motion in 15 seconds on an SGI Octane workstation with a 270 MHz MIPS R12000 processor and 256 MB memory.

2.10 Discussion

We have presented general schemes for three variants of the path planning problem. These schemes point out the commonalities and differences among various randomized path planners and provide general framework for their design; a crucial aspect in devising a randomized path planner is the sampling distribution for generating milestones.

The focus of this chapter has been to present an efficient planner for the single-query problem. Our algorithm iteratively builds two trees of sampled milestones rooted at the initial and the goal configuration. In contrast to traditional PRM methods, it samples only regions of the configuration space that is relevant to the current query, thus avoiding the cost of pre-computing a roadmap for the entire configuration space. As an additional advantage, our algorithm has much lower rejection rate: according to the experiments, roughly half of the sampled configurations are rejected. In comparison, many PRM methods have rejection rate as high as 99% [KL94a], so most of the configurations picked are in collision with obstacles and have to be discarded.

Although the randomized expansion planner has demonstrated strong performance in the experiments, several aspects of the algorithm deserve further investigation. Our current implementation of the algorithm uses a fixed-size neighborhood around an existing milestone to sample new configurations. The size of neighborhoods has a big impact on the distribution of milestones. If the size is too small, the milestones tend to cluster around the initial and the goal configuration and leave large portions of the free space with no samples. If the size is very large, the samples likely distribute more evenly in the free space, but the rejection rate also increases significantly. Ideally we would like to choose a size that is

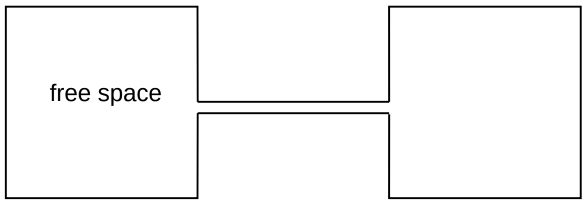

Figure 2.23. A narrow passage in the configuration space.

large enough for the samples to eventually distribute rather uniformly in the free space, but without significantly increasing the rejection rate. One possible way to achieve this is to determine the size of the neighborhood adaptively for every existing millstone. We increase the size of the neighborhood whenever possible and reduce it if the rejection becomes too high.

The most significant difficulty for our algorithm, as well as all other randomized motion planners, is the presence of narrow passages in the configuration space (Figure 2.23). This is indicated by relatively long running times in several experiments, *e.g.*, the skinny robot manipulator arm moving among closely-spaced obstacles (Section 2.7) and the snake-like rigid body maneuvering through small holes (Section 2.8). We will give a formal characterization of narrow passages in Chapter 4 and discuss its implication in detail, but intuitively it is clear that sampling at random a point in a small set is difficult. Several proposals have been made to address the issue. One possibility is to dilate the free space by allowing some penetration of the object into the obstacles [HKL+98]. First a roadmap G' is computed in the dilated free space. Then milestones in G' that do not lie in the free space $\mathcal{F}$ are pushed back into $\mathcal{F}$ by local resampling. Preliminary results based on this idea are encouraging. However, the penetration distance needed for dilating the free space is difficult to define and compute efficiently. When the moving object and the obstacles have complex geometry, computing the penetration distance remains a question with no satisfactory answer. An alternative approach is to sample more densely near obstacle boundaries [ABD+98, BOvdS99]. The intuition is that there is likely a higher concentration of obstacle boundaries near narrow passages. So sampling densely near obstacle boundaries may allow more samples to fall into the narrow passages. The effectiveness of this approach has only been demonstrated in low-dimensional configuration spaces.

Finally, although the randomized expansion method has been primarily used for the single-pair problem, it can also be applied to the all-pairs problem. The idea would be to sample uniformly a very small number of free configurations from the configuration space and use the randomized expansion planner to expand from these configurations in order to generate additional milestones, thus reducing the high rejection rate of many traditional PRM algorithms.

CHAPTER 3

Motion Planning under Kinematic and Dynamic Constraints

Path planning considers only the geometry of moving objects and obstacles. Despite its fundamental importance, it does not address some key aspects of the physical world: inherent limits in mechanical or biological systems restrict the motion that is possible. For example, a car cannot move sidewise. These limits cause certain configurations to be invalid, even if an object does not collide with obstacles at those configurations. In this chapter, we consider two important classes of constraints, non-holonomic (kinematic) constraints and dynamic constraints. Unlike obstacles in the environment, these constraints cannot always be represented in the configuration space. They involve not only the configuration, but also the velocity and possibly the acceleration of moving objects.

To address this issue, we use *state space*, a straightforward generalization of configuration space. Every point in the state space contains information on both the configuration and the velocity of an object. Our goal is to find, in the state space, a trajectory that is both collision-free and satisfies the kinematic or dynamic constraints on motion. This new class of problems is often called *kinodynamic motion planning* [DXCR93].

We would like to extend the random-sampling algorithm, used for path planning in the previous chapter, to solve kinodynamic motion planning problems. We start with a mathematical characterization of kinematic and dynamic constraints (Section 3.1) and review related work (Section 3.2). Next we formulate the kinodynamic motion problem in the state space (Section 3.3) and present an efficient algorithm for it based on an extension of the randomized expansion method from the previous chapter (Section 3.4). Our algorithm has been tested on two different systems in simulation and demonstrated good performance. The experimental results are reported in Sections 3.5 and 3.6. Finally in Section 3.7, we further demonstrate the generality and effectiveness of our algorithm by showing how to apply it to a real-time robot system operating under strict dynamic constraints in an environment with moving obstacles.

3.1 Kinematic and Dynamic Constraints

In the previous chapter, we assumed that there is no constraint on the motion of a moving object M; each dof of M is free to change its value independently. The presence of constraints, however, introduces dependency among various dofs and increases the complexity of motion planning. The constraints that we are going to consider fall into two main categories, kinematic and dynamic.

Kinematic constraints impose a relationship between the configuration q of M and its velocity $\dot{q}$. They can be written down mathematically as

$$F(q, \dot{q}) = 0. \tag{3.1}$$

Kinematic constraints can be further classified into holonomic and non-holonomic ones.

Holonomic constraints do not involve the velocity of a moving object; they have the special form $F(q) = 0$. A set of holonomic constraints can be used to eliminate some of the configuration parameters and reduce the dimension of the configuration space. In fact, articulated objects are examples of moving objects that obey holonomic constraints, if we treat each link of the articulated objects as a rigid body, whose configuration is determined by six parameters, three for the translation and three for the rotation. If there are c links, such a representation requires $6c$ parameters. The configuration space $\mathcal{C}$ can thus be embedded as a manifold in R^{6c}, often referred to as the *ambient space* of $\mathcal{C}$. Notice, however, that any two adjacent links of an articulated object are connected by a joint, which restricts the relative motion between the two links. If the joint angles are used to parameterize $\mathcal{C}$, we need only c parameters, assuming that each joint has one dof. In addition, these parameters are all

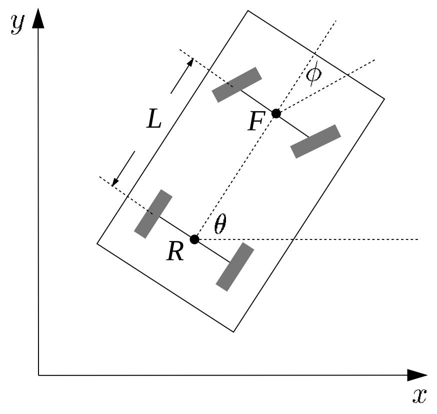

Figure 3.1. A simplified model for a car-like robot.

independent. By choosing a suitable parameterization of $\mathcal{C}$, we can convert a problem with holonomic constraints into one with no constraints and apply the algorithm from Chapter 2.

Of course, such a global parameterization may not always be possible. In this case, there are several ways to proceed if we wish to apply random-sampling techniques. For example, consider a closed-chain robot, which is another common example involving holonomic constraints. We can sample at random a set of independent parameters first and then solve the constraints locally for the other parameters [HA00]. Another possibility is to sample a point in the ambient space and then project the sampled configuration onto the manifold defined by the constraints.

Non-holonomic constraints are fundamentally different from holonomic ones. They are not integrable, meaning that we cannot eliminate $\dot{q}$ via integration and convert them to the form $F(q) = 0$. The constraints that affect the motion of a car-like mobile robot are an important example of non-holonomic constraints (Figure 3.1). Let (x, y) be the position of the midpoint *R* between the rear wheels of the robot and θ be the orientation of the rear wheels with respect to the x-axis. Assume that the wheels do not skid. Then the robot cannot move sidewise. This constraint can be written as $\tan\theta = \dot{y}/\dot{x}$, which clearly has the form $F(q, \dot{q}) = 0$. What is not clear is that the constraint is not integrable. We will not get into the details here. It suffices to say that the mathematical conditions for integrability is known, but for a given set of constraints, checking these condition is a non-trivial task. A more complete description of these issues is available in [Lat91b, pages 403–451].

Dynamic constraints are closely related to non-holonomic constraints. They involve not only the configuration and the velocity of M, but also its acceleration. In general, Lagrange's equations of motion have the form

$$G(q, \dot{q}, \ddot{q}) = 0, \tag{3.2}$$

where q, $\dot{q}$, and $\ddot{q}$ are the configuration, velocity, and acceleration of M. Defining $s = (q, \dot{q})$, we can rewrite (3.2) as $F(s, \dot{s}) = 0$ which is the same as (3.1). So kinematic constraints and dynamic constraints have the same mathematical form.

The motion of objects can also be constrained by inequalities of the forms $F(q, \dot{q}) \leq 0$ or $G(q, \dot{q}, \ddot{q}) \leq 0$. Such constraints restrict the set of admissible states to a subset of the state space.

3.2 Related Work

Non-holonomic motion planning refers to problems in which an object's motion must satisfy non-holonomic constraints. It has attracted considerable interest in robotics [Lau86, LCH89, BL93, LJTM94, LM96, SŠLO97], because the motion of wheeled mobile robots typically obeys non-holonomic constraints. One approach for non-holonomic planning is to proceed in two stages [Lau86]. First we generate a collision-free path that disregards the non-holonomic constraints and then transform the path into an *admissible* one, *i.e.*, a path that conforms to the non-holonomic constraints. This two-stage algorithm can be extended in various ways, which are all based on the idea of successive path transformation, but differ in what transformations to use and how to perform the transformations [SŠLO97, SL98, Fer98]. Techniques for finding admissible paths that have certain nice properties have also been investigated (see, *e.g.*, [LJTM94, MC92]). A natural question to ask about these path-transformation methods is whether it is always possible to transform a collision-free path into an admissible path that obeys the non-holonomic constraints. The answer is yes for car-like robots [Lau86]. One can also prove that the result holds in general for any locally controllable system by applying tools from non-linear control theory [LCH89, LS89, BL93]. Unfortunately the transformation suggested by the proof does not generate paths that are useful in practice. So these methods are only applicable to systems for which an efficient transformation method is available, *e.g.*, systems possessing a chained form [MS90].

Other approaches to non-holonomic motion planning follow classic paradigms in path planning. Jacobs and Canny proposed a roadmap planner for car-like robots [JC89]. Their algorithm discretizes the boundaries of polygonal obstacles and connects pairs of points on the boundaries by canonical curves composed of circular and straight-line segments. Barraquand and Latombe used the cell-decomposition approach [BL93]. Their planner builds a search tree systematically in a discretized state space. At each iteration, it expands a node of the tree by integrating the robot's equations of motion for a short duration of time under some admissible control. The algorithm works for car-like robots and tractor-trailer robots with a relatively small number of dofs. It does not require the robots to be locally controllable.

Although most of the work on non-holonomic motion planning focuses on car-like and tractor-trailer robots, some of the results are applicable to other scenarios as well, *e.g.*, systems for pushing [LM96, ALMR97] and dextrous manipulation [HLS88]

Approaches for handling dynamic constraints parallel those for non-holonomic motion planning. One possibility is to compute a collision-free geometric path first and then use variational techniques to deform the path into one that both conforms to the dynamic constraints and optimizes a certain criterion such as minimal execution time [BDG85, SD91]. A drawback of this approach is that it may not always be possible to transform a collision-free path into an admissible one due to limits on the available actuator forces and torques. Also no formal guarantee of performance has been established for these planners. Alternatively one may place a regular grid on the state space and searches for an admissible path directly using dynamic programming [DXCR93]. The latter approach is similar to the cell-decomposition method of Barraquand and Latombe for non-holonomic motion planning. It offers provable performance guarantees, but is only applicable to robot with a small number of dofs, because the size of the grid grows exponentially with the number of dofs. Our planner is related to this approach, but discretizes the state space via random sampling rather than placing a regular grid over it. The planner in [LK99] resembles ours, but no guarantee of performance has been established for it.

With few exceptions (*e.g.*, [Fra99]), earlier work considers non-holonomic and dynamic constraints separately. However, as we have seen in the Section 3.1, the mathematical nature of these two types of constraints is the same, and so they can be treated in a unified framework.

3.3 State-Space Formulation

We consider moving objects whose motion is governed by an equation of the form

$$\dot{s} = f(s, u), \tag{3.3}$$

where $s \in \mathcal{S}$ is the object's state, $\dot{s}$ is the derivative of s with respect to time, and $u \in \Omega$ is the control input. The set $\mathcal{S}$ and Ω are called the *state space* and *control space*, respectively. We assume that $\mathcal{S}$ and Ω are bounded manifolds of dimensions n and m ($m \leq n$). By defining appropriate charts on these manifolds, we can treat $\mathcal{S}$ and Ω as subsets of R^n and R^m.

Eq. (3.3) can represent both kinematic and dynamic constraints discussed in Section 3.1. Suppose that there are k kinematic or dynamic constraints $G_i(s, \dot{s}) = 0$ for $i = 1, 2, \ldots, k$. We can solve these k equations for $\dot{s}$. In general, if k is less than n, the solution is not unique, but we can parameterize the set of solutions by $u \in \mathrm{R}^{n-k}$ and write them down, at least formally, as $\dot{s} = f(s, u)$ for some suitable function f. More precisely, it can be shown that under appropriate conditions, the set of constraints $G_i(s, \dot{s}) = 0$ for $i = 1, 2, \ldots, k$ is equivalent to (3.3), in which u is a point in $\mathrm{R}^m = \mathrm{R}^{n-k}$ [BL93].

To deal with inequality constraints of the form $G(s, \dot{s}) \leq 0$, we typically restrict the state space $\mathcal{S}$ and control space Ω to suitable subsets of R^n and R^m.

These notions are illustrated below with two examples that will be useful later in the chapter:

Example 1 (simplified non-holonomic car navigation). Consider the car example in Figure 3.1. Let $(x, y, \theta) \in \mathrm{R}^3$ be the state of the car. The non-holonomic constraint $\tan\theta = \dot{y}/\dot{x}$ is equivalent to the system

$$\begin{aligned} \dot{x} &= v\cos\theta \\ \dot{y} &= v\sin\theta \\ \dot{\theta} &= (v/L)\tan\phi. \end{aligned}$$

This reformulation corresponds to defining the car's state to be its configuration (x, y, θ) and choosing the control input to be the vector (v, ϕ), where v and ϕ are the car's speed and steering angle. Bounds on (x, y, θ) and (v, ϕ) can be used to restrict $\mathcal{S}$ and Ω to subsets of R^3 and R^2, respectively. For instance, if the maximum speed of the car is 1, then we have $|v| \leq 1$. $\triangleleft$

Example 2 (point-mass robot with dynamics). For a point-mass robot P moving on a horizontal plane, we typically want to control the forces applied to P. This leads us to define the state of P as $s = (x, y, v_x, v_y)$, where (x, y) and (v_x, v_y) are the configuration and the velocity of P. The control inputs are chosen to be the forces applied to P in the x- and y-direction. Hence the equations of motion are

$$\begin{array}{rclcrcl} \dot{x} & = & v_x & \qquad & \dot{v_x} & = & u_x/m \\ \dot{y} & = & v_y & \qquad & \dot{v_y} & = & u_y/m, \end{array} \tag{3.4}$$

where m is the mass of P and (u_x, u_y) is the applied force. The velocity (v_x, v_y) and force (u_x, u_y) are restricted to subsets of R^2 due to limits on the maximum velocity and forces. $\triangleleft$

Obstacles in the environment are mapped into $\mathcal{S}$ as forbidden regions. The free space $\mathcal{F} \subset \mathcal{S}$ contains all the states that are collision-free. A trajectory $\sigma\colon [a, b] \mapsto \mathcal{S}$ is *admissible* if, for all $t \in [a, b]$, $\sigma(t)$ lies in $\mathcal{F}$ and obeys the motion constraints.

A planning query is specified by an initial state s_{init} and a goal state s_{goal}. A solution to the query is a control function $u\colon [a, b] \mapsto \Omega$ that produces an admissible trajectory from s_{init} to s_{goal}.

3.4 Control-Driven Randomized Expansion

Like the planner for path planning (see Section 2.4), our algorithm for kinodynamic motion planning iteratively builds a tree-shaped roadmap T from the initial state s_{init}. However, there are two important differences. First the tree is constructed in the state space of a moving object rather than its configuration space. Second the trajectories connecting two milestones in T may neither be straight nor be reversible due to the constraints. As a result, we need a new way of sampling milestones.

At each iteration, the algorithm picks at random an existing milestone s from T, a control function u, and a small time duration δ. It then integrates (3.3) from s with u for the time duration δ and obtains the trajectory induced by u. If the trajectory is admissible, its endpoint s' is added to T as a new milestone. The planner also inserts in T a directed edge from s to s' and stores u with this edge. The motion constraints are thus naturally enforced in all trajectories connecting pairs of milestones in T. The planner exits with success when a sampled milestone falls into an *endgame* region containing the goal.

Milestone selection At each iteration, the planner picks an existing milestone s from T at random with probability $\pi_T(s)$, which is inversely proportional to the weight of s. Similar to the case for path planning, the weight of s is equal to the number of other milestones in the neighborhood of s. So a milestone lying in a sparsely-sampled region is more likely to be selected than one in a densely-sampled region. This technique avoids oversampling any particular region of $\mathcal{F}$.

Control function selection To facilitate the analysis later on, we consider only piecewise-constant control functions. A function $u\colon [a,b] \mapsto \Omega$ is piecewise constant if the interval $[a,b]$ admits a finite partition $a = t_0 < t_1 < \ldots < t_\ell = b$ such that u is a constant $c_i \in \Omega$ over the open interval (t_{i-1}, t_i) for $i = 1, 2, \ldots, \ell$. In addition, we require $t_i - t_{i-1} \leq \delta_{\max}$ for some strictly positive value $\delta_{\max}$. The function u can thus be represented compactly as a sequence of pairs (c_i, δ_i) for $i = 1, 2, \ldots, \ell$, where $\delta_i = t_i - t_{i-1} \leq \delta_{\max}$. Let $\mathcal{U}_\ell$ denote the set of such piecewise-constant functions with ℓ constant segments. Our algorithm picks a control function $u \in \mathcal{U}_\ell$, for some pre-specified ℓ and $\delta_{\max}$, by sampling each constant piece of u independently. For each piece, it picks c_i and δ_i uniformly and independently at random from Ω and $[0, \delta_{\max}]$. The specific choices of the parameters ℓ and δ_{max} will be discussed in Subsection 4.3.4. In the actual implementation of the algorithm, however, one may choose $\ell = 1$, because any trajectory passing through several consecutive milestones in the tree T is obtained by applying a sequence of constant controls.

Endgame connection The above "control-driven" sampling technique does not allow us to reach the goal state s_{goal} exactly. We need to expand s_{goal} into a relatively large endgame region that the sampling algorithm will eventually attain with high probability.

To do so, we build a secondary tree T' of milestones from s_{goal} in the same way as that for the primary tree T, except that (3.3) is integrated backwards in time. Let s' be a new

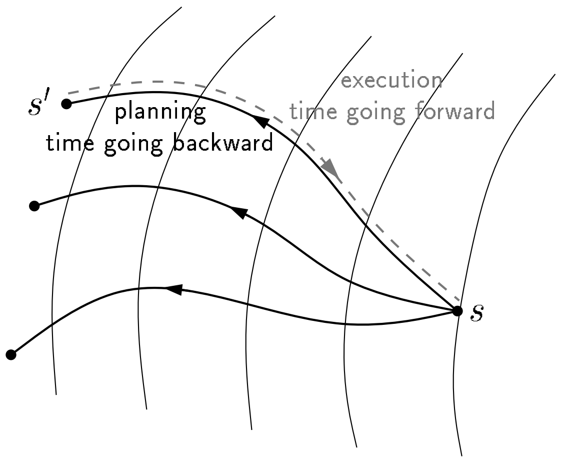

Figure 3.2. Building a secondary tree of milestones by integrating backwards in time.

milestone obtained by integrating backwards from an existing milestone s. By construction, if the time goes forward, the control function drives the robot from s' to s (Figure 3.2). Thus there is a known trajectory from every milestone in T' to the goal. The sampling process terminates with success when a milestone $s \in T$ is in the neighborhood of a milestone $s' \in T'$. In this case, the endgame region is the union of the neighborhoods of milestones in T'. To generate the final trajectory, we simply follow the appropriate edges of T and T'; however, there is a small gap between s and s'. The gap can often be dealt with in practice. For example, beyond s, one can use a PD controller to track the trajectory extracted from T'. Constructing endgame regions by backward integration is a very general technique and can be applied to any system described by (3.3).

Alternatively, for some systems, it is possible to compute analytically one or several canonical control functions that connect two given states exactly while obeying the kinematic or dynamic constraints. An example is the Reeds and Shepp curves established for non-holonomic car-like robots [RS90]. If such control functions are available and can be computed efficiently, we can test whether a milestone s belongs to the endgame region by checking that a canonical control function induces an admissible trajectory from s to s_{goal}.

Algorithm in pseudocode The algorithm is summarized in the following pseudocode.

Algorithm 3.1 Control-driven randomized expansion for kinodynamic motion planning.

```
1.  Insert $s_{\text{init}}$ into a tree $T$; $i \leftarrow 1$.
2.  repeat
3.    Sample a milestone $s$ from $T$ with probability $\pi_T(s)$.
4.    Sample a control function $u$ from $\mathcal{U}_\ell$ uniformly at random.
5.    $s' \leftarrow$ PROPAGATE$(s, u)$.
6.    if $s' \neq$ nil then
7.      Add $s'$ to $T$; $i \leftarrow i + 1$.
8.      Create a directed edge $e$ from $s$ to $s'$ and store $u$ with $e$.
9.      if $s' \in$ ENDGAME then exit with SUCCESS.
10.     if $i = N$ then exit with FAILURE.
```

In line 5, PROPAGATE(s, u) integrates the equation of motion from state s with control u. It returns a new milestone s' if the computed trajectory is admissible; otherwise it returns *nil*. If there exists no admissible trajectory from s_{init} to s_{goal}, the algorithm cannot detect it. Therefore, in line 10, we bound the total number of milestones to be sampled by an input constant N.

3.5 Experiments with Non-Holonomic Constraints

We tested our algorithm on three systems with different kinematic and dynamic constraints. The first one consists of two wheeled mobile robots that maintain a direct line of sight as well as a minimum and a maximum distance between them. The second one simulates the behavior of a hovercraft with simplified dynamics. The third one is an air-cushioned robot that is propelled by air thrusters and operates among moving obstacles on a flat table. In this section, we discuss implementation issues and present experimental results for the wheeled mobile robots. We will do the same for the hovercraft in the next section and give a brief account of the experiments with the air-cushioned robot in Section 3.7. Our algorithm is implemented in C++. The running times reported in these sections were obtained on an SGI Indigo 2 workstation with an 195 MHz processor and 384 MB memory.

Wheeled mobile robots are a classical example for non-holonomic motion planning. The

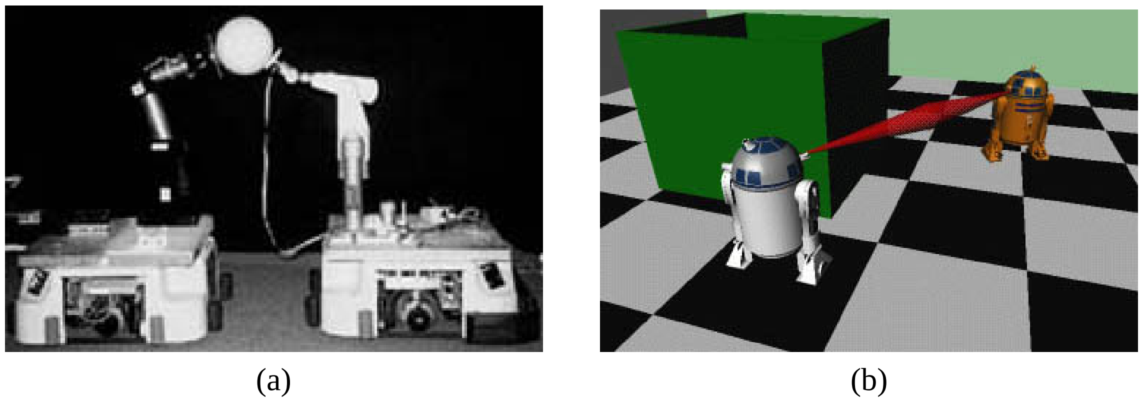

Figure 3.3. Two-cart non-holonomic robots. (a) Cooperative mobile manipulators. (b) Two wheeled non-holonomic robots that maintain a direct line of sight and a distance range.

robot considered here is a new variation on this theme. It consists of two independently-actuated carts moving on a flat surface (Figure 3.3). Each cart obeys a non-holonomic constraint and has non-zero minimum turning radius. In addition, the two carts are connected by a telescopic link whose length is lower and upper bounded. This system is inspired by two scenarios. One is the mobile manipulation project in the GRASP Laboratory at The University of Pennsylvania [DK99]; the two carts are each mounted with a manipulator arm and must remain within a certain distance range so that the two arms can cooperatively manipulate an object (Figure 3.3a). The manipulation area between the two carts must be free of obstacles. In the other scenario, two mobile robots patrolling an indoor environment must maintain a direct line of sight and stay within a certain distance range, in order to allow visual contact or simple directional wireless communication (Figure 3.3b).

3.5.1 Implementation Details

Description of the system We project the geometry of the carts and the obstacles onto the horizontal plane (Figure 3.4). For $i = 1, 2$, let R_i be the midpoint between the rear wheels of the *i*th cart, F_i be the midpoint between the front wheels, and L_i be the distance between R_i and F_i. We define the state of the system as $s = (x_1, y_1, \theta_1, x_2, y_2, \theta_2)$, where (x_i, y_i) are the coordinates of R_i, and θ_i is the orientation of the rear wheels of the *i*th cart with respect to the x-axis. To maintain a distance range between the two carts, we require

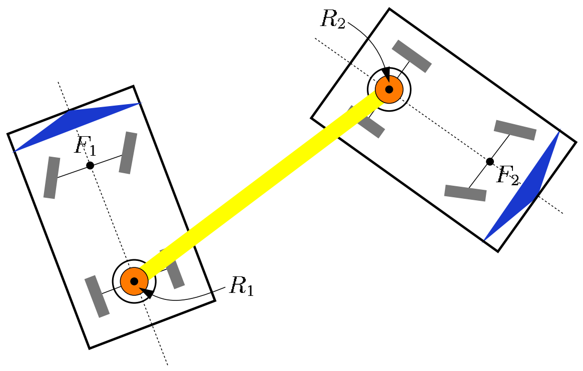

Figure 3.4. A planar model of the two robot carts.

$d_{\min} \leq \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \leq d_{\max}$ for some constants $d_{\min}$ and $d_{\max}$.

Each cart has two scalar controls, u_i and ϕ_i, where u_i is speed of R_i, and ϕ_i is the steering angle. The equations of motion for the system are

$$\begin{aligned} \dot{x_1} &= u_1 \cos\theta_1 & \dot{x_2} &= u_2 \cos\theta_2 \\ \dot{y_1} &= u_1 \sin\theta_1 & \dot{y_2} &= u_2 \sin\theta_2 \\ \dot{\theta}_1 &= (u_1/L_1)\tan\phi_1 & \dot{\theta}_2 &= (u_2/L_2)\tan\phi_2. \end{aligned} \tag{3.5}$$

The control space is restricted by $|u_i| \leq u_{\max}$ and $|\phi| \leq \phi_{\max}$, which bound the carts' velocities and steering angles.

Since all obstacles are stationary, the planner samples milestones in the 6-D state space of the carts.

Implementing PROPAGATE Given a milestone s and a control function u, PROPAGATE uses the Euler method with a fixed step size to integrate (3.5) from s and computes a trajectory σ of the system under the control u. More sophisticated integration methods, *e.g.*, fourth-order Runge-Kutta or extrapolation method [PTVP92], can improve the accuracy of integration, but at a higher computational cost.

We then discretize σ into a sequence of states and returns *nil* if any of these states is in collision. For each cart, we pre-compute a 3-D bitmap that represents the collision-free configurations of the cart prior to planning. It then takes constant time to check whether a given configuration is in collision. A well-known disadvantage of this method is that if the resolution of the bitmap is not fine enough, we may get wrong answers. In the experiments reported below, we used an $128 \times 128 \times 64$ bitmap, which was adequate for our test cases.

Endgame connection We obtain the endgame region by constructing a secondary tree of sampled milestones from s_{goal}.

3.5.2 Results

We experimented with the planner in a large number of environments. Each one is a 10 m $\times$ 10 m square region with static obstacles. The two robot carts are identical, each represented by a polygon contained in a circle of radius 0.4 m, and $L_1 = L_2 = 0.5$ m. The speed of the carts ranges from -3 m/s to 3 m/s, and its steering angle varies between $-30°$ and $30°$. The distance between the two carts ranges from 1.4 m to 3.3 m.

Figures 3.5–3.8 show four of the test scenes that we used. Every figure contains several query states as well as a computed trajectory for some particular query. Figure 3.5 shows a structured indoor environment, in which the two carts move from one "room" into another while obeying all the motion constraints. Figure 3.6 shows a maze; the carts navigate from one side of the maze to the other. Figure 3.7 contains two large obstacles separated by a narrow passage. The two robots, which are initially parallel to each other, change formation and proceed in a single file through the passage, before becoming parallel again in the end. Figure 3.8 shows an environment consisting of two rooms cluttered with obstacles and connected by a hallway. The carts need to move from the lower room to the upper one. The maximum steering angles and the size of the circular obstacles conspire to increase the number of required maneuvers.

We ran the planner on several different queries for every environment. For each query, we ran the planner 30 times independently with different random seeds. The results summarized in Table 3.1 are the average over 30 runs. Every row of the table corresponds to a particular query. Column 2 specifies the query configurations. Columns 3–7 list the average running time, the average number of collision checks, and their standard deviations. Columns 8–9 give the total number of milestones sampled and the number of calls to PROPAGATE. The running times range from less than a second to a few seconds, and the final roadmaps typically contain a few thousand milestones. One query in the last test scene takes much longer because the carts must perform several complicated maneuvers in the hallway before

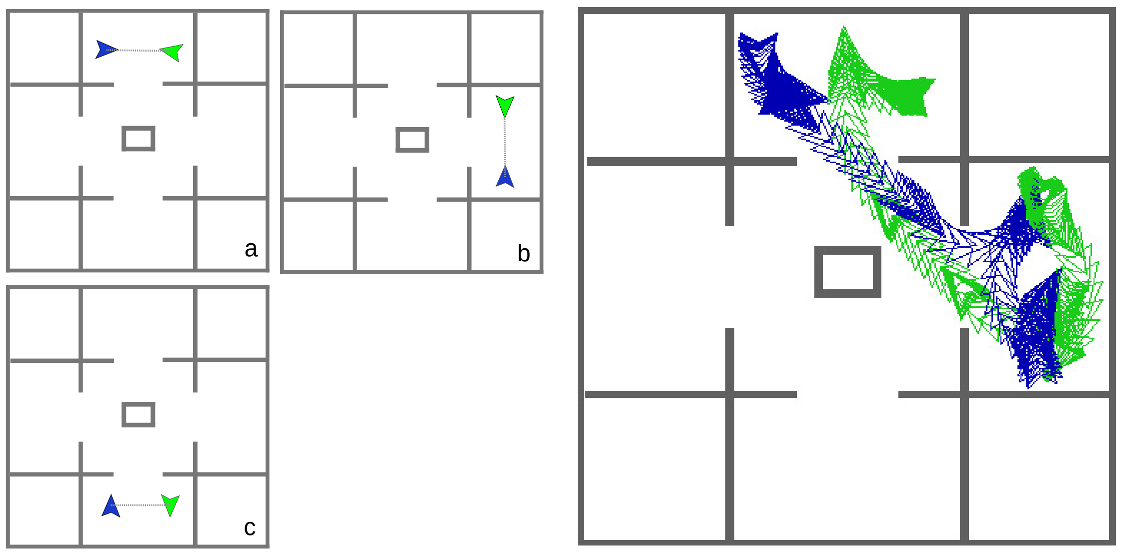

Figure 3.5. Test scene 1 for the wheeled robot carts: a structured indoor environment.

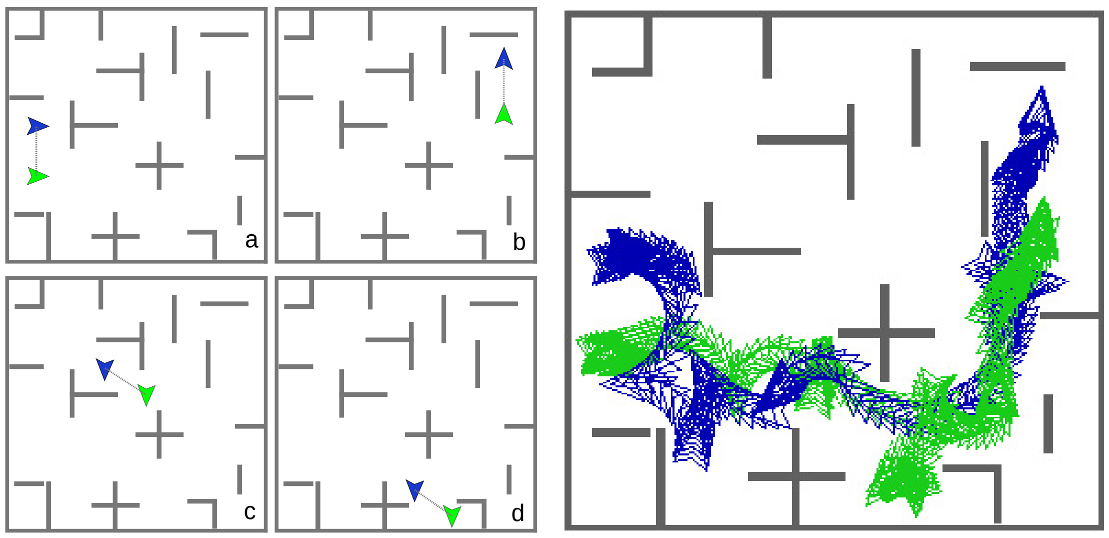

Figure 3.6. Test scene 2 for the wheeled robot carts: a maze.

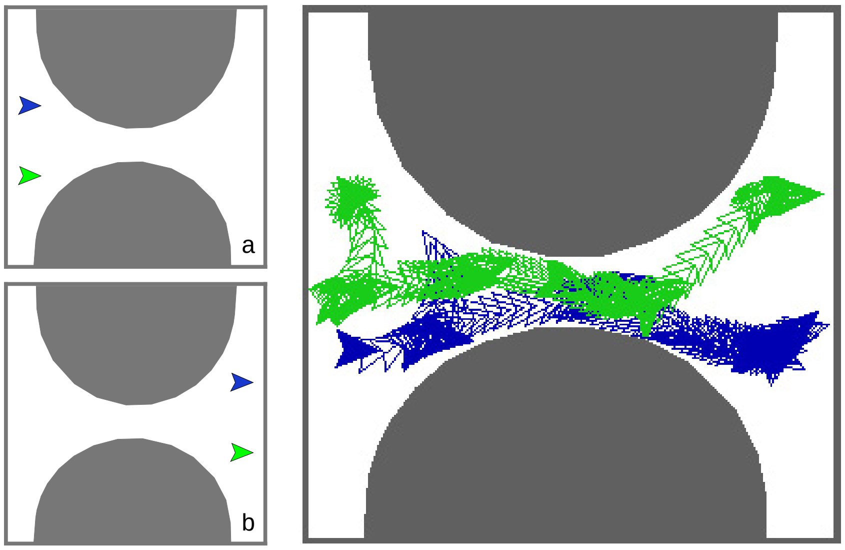

Figure 3.7. Test scene 3 for the wheeled robot carts: an environment with a narrow passage.

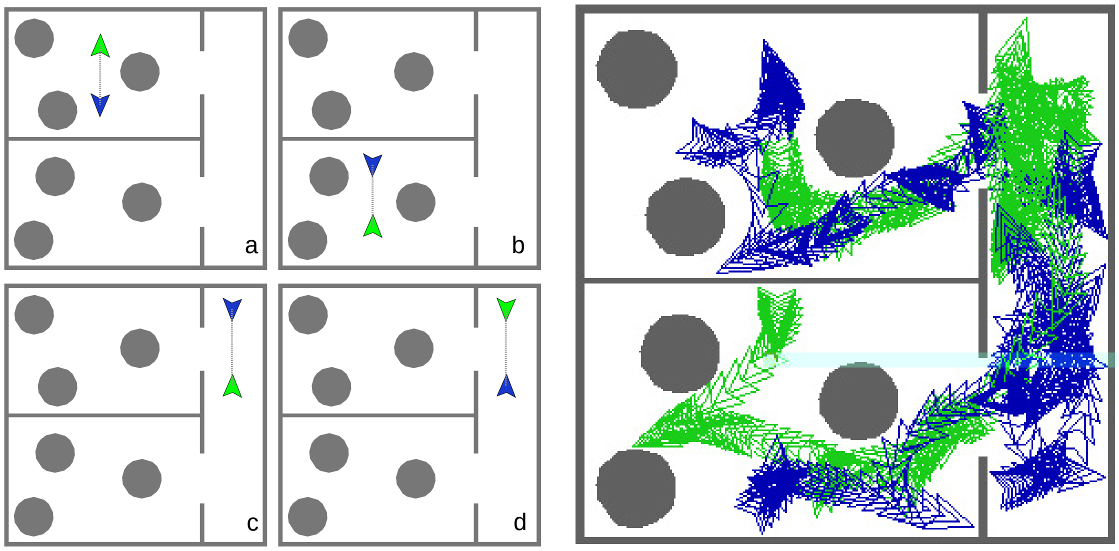

Figure 3.8. Test scene 4 for the wheeled robot carts. The environment consists of two rooms that contain large circular obstacles and are connected by a narrow hallway.

Table 3.1. Performance statistics of the planner on non-holonomic robot carts.

Scene	Query	Time (sec.)		N_{clear}		N_{mil}	N_{pro}
		mean	std	mean	std		
1	a, b	0.43	0.27	33199	14553	1125	12077
	a, c	1.10	0.60	60993	22010	2096	22018
	b, c	0.52	0.40	36061	18891	1259	12859
2	a, b	1.39	0.91	62402	27001	2473	21316
	a, c	0.74	0.65	43564	23640	1630	15315
	b, d	0.54	0.41	35960	18410	1318	12815
	c, d	0.55	0.44	38384	20772	1310	14066
3	a, b	4.45	3.92	126126	61836	4473	45690
4	a, b	14.09	7.42	287828	86987	9123	107393
	c, d	0.92	0.51	56367	20825	1894	20250

N_{clear}: number of collision checks N_{mil}: number of milestones
N_{pro} : number of calls to PROPAGATE

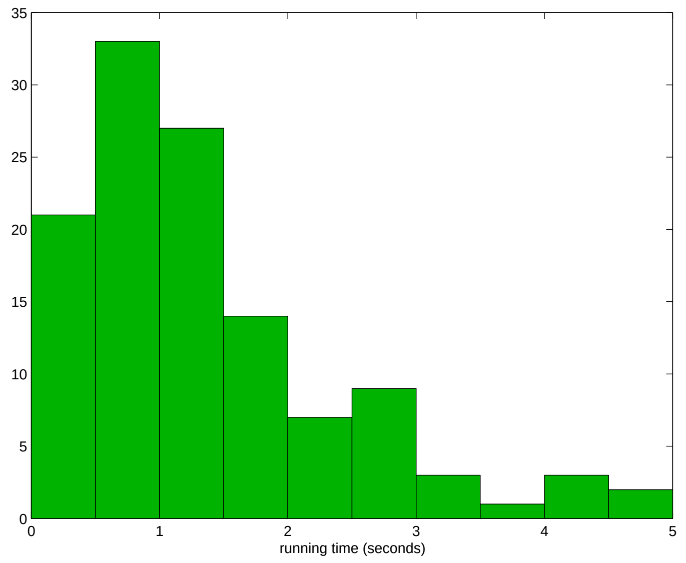

Figure 3.9. A histogram of running times for more than 100 runs on a particular query. The average running time is 1.4 seconds, and the four quartiles are 0.6, 1.1, 1.9, and 4.9 seconds.

reaching the final goal (Figure 3.8).

The standard deviations in Table 3.1 are larger than we would like. In Figure 3.9, we show a histogram of more than 100 independent runs for a particular query. It indicates that our planner performs well most of the time. The large deviation is caused by a few runs that take as long as three times the mean. The long and thin tail of the distribution is typical in all our tests. Note also that the shape of the distribution is similar to the one shown in Figure 2.14.

3.6 Experiments with Dynamics

We also tested our algorithm on a system with dynamics. Dynamics makes motion planning more difficult. The set of possible velocities that an object may achieve is severely restricted because the motion of the object must obey the laws of physics, and only limited actuator forces and torques are available. For example, a heavy fast-moving object cannot stop instantaneously. The motion planner must take this into account when generating the trajectory.

The system that we are going to consider simulates the behavior of a hovercraft propelled by two thrusters going both forward and backward. Hovercraft is an interesting example for motion planning, but previous work [LM97, Lyn99] tends to focus on the controllability issues. Here we use it as an example to evaluate the performance of our planner when the system is subject to dynamic constraints. We also incorporate damping forces proportional to the linear and angular velocity of the hovercraft in our test cases.

3.6.1 Implementation Details

The hovercraft is modeled as a symmetric polygonal body P (Figure 3.10), navigating in a 2-D environment with static obstacles. Let (x, y) be the position of the center of P, and θ be the orientation of P with respect to the x-axis of a fixed coordinate system. We define the state of P as $(x, y, \theta, v_x, v_y, \omega)$, where (v_x, v_y) is the linear velocity of P, and ω is the angular velocity. The lateral distance between two thrusters is L.

The hovercraft has two thrusters providing forces f_1 and f_2 along the main axis of the craft. Although we can use f_1 and f_2 as control inputs directly, a better set of controls is the

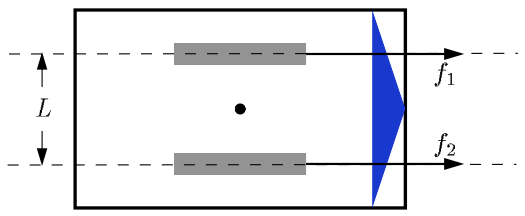

Figure 3.10. A planar model of the hovercraft.

total force $u_1 = f_1 + f_2$ and the total torque $u_2 = (f_1 - f_2)L/2$ applied to the craft by the thrusters. By using u_1 and u_2, it is much easier to restrict the maximum torque and avoid setting the craft spinning. The equations of motion for the hovercraft are

$$\begin{aligned} \dot{x} &= v_x & \dot{v}_x &= (u_1 \cos\theta - K_v v_x)/m \\ \dot{y} &= v_y & \dot{v}_y &= (u_1 \sin\theta - K_v v_y)/m \\ \dot{\theta} &= \omega & \dot{\omega} &= (u_2 - K_\omega \omega)/I, \end{aligned} \tag{3.6}$$

where m and I are the mass and the moment of inertia of P, and K_v and K_ω are the coefficients of the damping forces.

Our planner samples milestones in the 6-D state-space of the hovercraft. The implementation of our algorithm for the hovercraft is basically the same as that for the non-holonomic carts in the previous section, except that (3.6) replaces (3.5) in PROPAGATE when it integrates the equations of motion.

3.6.2 Results

We tested the planner in a number of different environments, each containing static obstacles in a 10 m $\times$ 10 m square region. The hovercraft is approximately 1.5 m in length and 0.7 m in width.

Three test scenes are shown in Figures 3.11–3.13. Scene 1 contains two large obstacles, representing anchored ships and has lots of free space for the hovercraft to maneuver (Figure 3.11). It is a relatively simple environment. Scene 2 consists of many small triangular obstacles distributed in a random fashion (Figure 3.12). The amount of free space available for maneuvering is much more restricted. Note also that for the computed

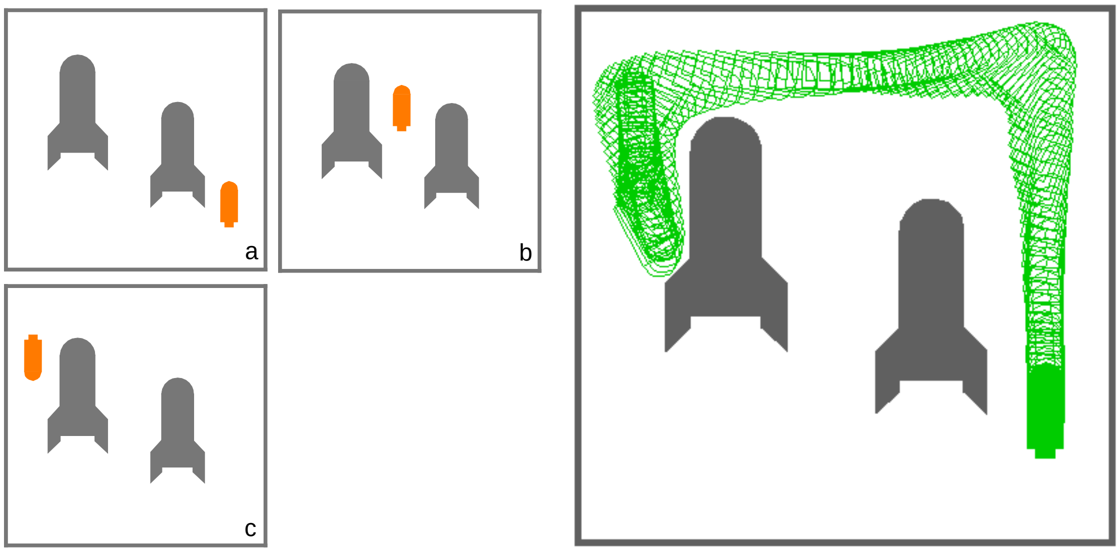

Figure 3.11. Test scene 1 for the hovercraft: a simple environment with two anchored ships as obstacles.

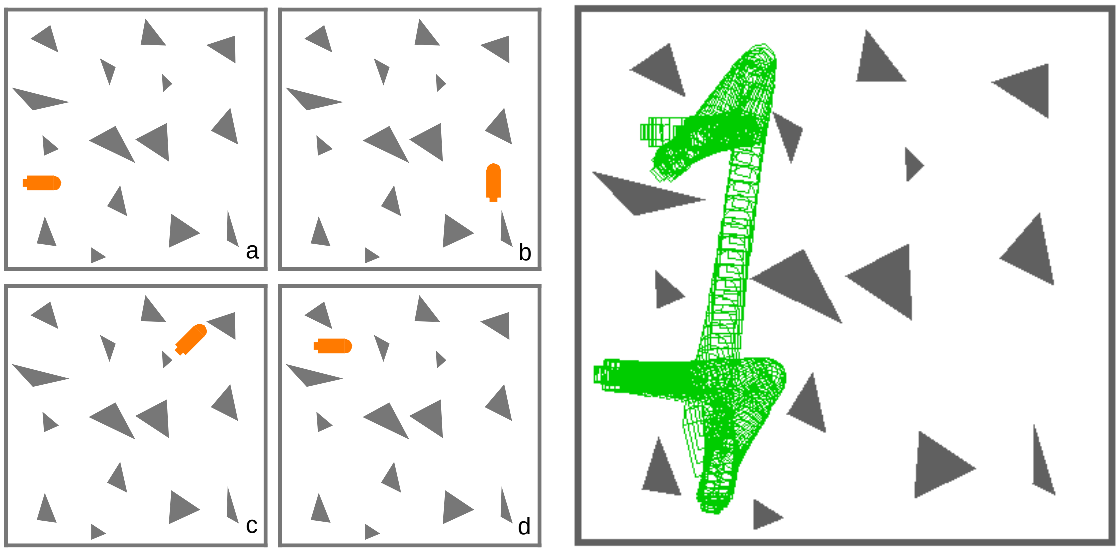

Figure 3.12. Test scene 2 for the hovercraft: many small obstacles distributed randomly.

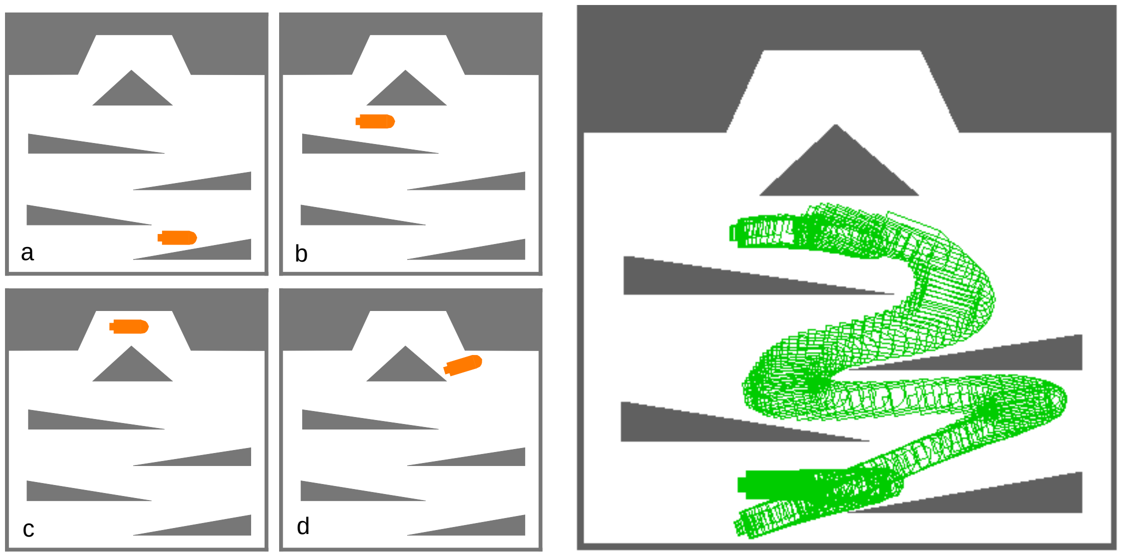

Figure 3.13. Test scene 3 for the hovercraft: a zigzag corridor defined by obstacles.

Table 3.2. Performance statistics of the planner on a hovercraft with simplified dynamics.

Scene	Query	Time (sec.)		N_{clear}		N_{mil}	N_{pro}
		mean	std	mean	std		
1	a,b	0.84	1.03	46518	31603	2345	15698
	a,c	2.71	3.17	93085	64686	4394	32179
	b,c	1.56	1.39	69060	40223	3368	23763
2	a,b	2.22	2.58	90427	69670	3449	33400
	a,d	3.55	2.54	137382	68358	5196	51193
	b,c	0.65	0.67	42467	29429	1714	15776
	c,d	1.93	1.60	97205	52190	3424	37442
3	a,b	6.07	7.09	250635	187726	6711	104567
	b,c	1.75	1.93	62600	47091	3130	21266
	b,d	1.06	1.93	41884	40202	2062	14314

N_{clear}: number of collision checks N_{mil}: number of milestones
N_{pro} : number of calls to PROPAGATE

example shown in the figure, the hovercraft has to perform several maneuvers near the end of the trajectory (top-left corner of the space) in order to reorient itself and achieve the final state. Scene 3 includes a query in which the hovercraft must zigzag through a corridor defined by obstacles (Figure 3.13). Since the corridor does not have enough space for turning around easily, the hovercraft alternates between forward motion and backward motion to arrive at the final state. It is interesting to note that the hovercraft exhibits sidewise motions during part of the trajectory. Since the hovercraft does not have propellers thrusting sidewise, it can accomplish this only by taking advantage of dynamics.

Again we ran the planner 30 times independently for every query. The results are reported in Table 3.2, which has the same format as that of Table 3.1. The typical running times for our test cases vary between less than a second to several seconds, and it usually takes a few thousand milestones to process a query. Like the other experiments that we have performed, the standard deviation of the running times are large, because of long execution times in a small number of runs, but overall the planner is very fast.

3.7 Discussion

We have generalized the randomized expansion planner, first used for path planning in Chapter 2, to solve a much broader class of problems that incorporate kinematic and dynamic constraints on the motion of objects. Our algorithm represents the motion constraints by an equation of the form $\dot{s} = f(s, u)$ and constructs a roadmap of sampled milestones in the state space of a moving object. It first picks at random a point in the space of admissible control functions and then maps the point into the state space by integrating the equations of motion. Thus the motion constraints are enforced naturally during the construction of roadmaps. The algorithm is general and can be applied to a wide class of systems, including ones that are not locally controllable. We have tested our planner on two systems, one with non-holonomic constraints and one with dynamic constraints. The experimental results demonstrate that our planner operates efficiently in state space of moderately high dimensions (6) and under complex kinematic and dynamic constraints.

In addition, a variant of our algorithm has been implemented on a real robot in an environment with moving obstacles [Kin00]. The robot system was developed in the

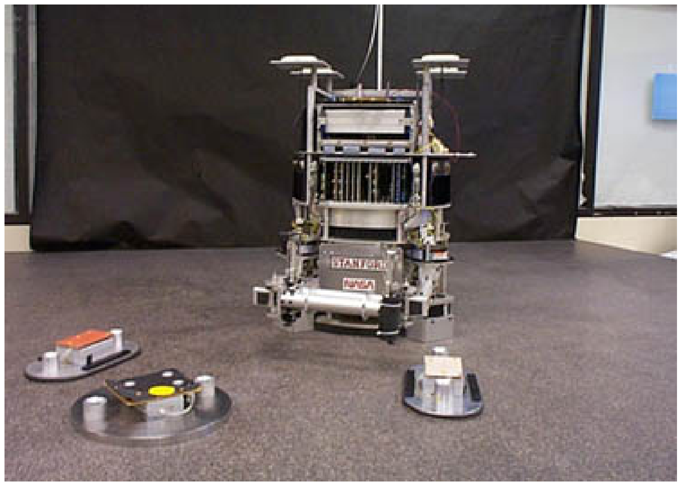

Figure 3.14. The air-cushioned robot among moving obstacles.

Stanford Aerospace Robotics Laboratory for testing space robotics technology. The air-cushioned robot moves frictionlessly on a flat granite table (Figure 3.14). It has eight air thrusters providing omni-directional motion capability, but the force is small compared to the robot's mass, resulting in tight acceleration limits. We model the robot as a disc for planning purposes and describe its motion by (3.4). An overhead vision system estimates the motion of moving obstacles in the environment and send the information to the planner, which runs on an off-board computer. The planner is then allocated a short, pre-defined amount of time to compute a trajectory, as required by the real-time nature of the system,

To deal with moving obstacles, the planner augments the state space with a time axis and computes a trajectory for the robot in the state-time space rather than the state space. Although the planner assumes that the obstacles move with constant linear velocities during the planning, the vision module continuously monitors the obstacles while the robot executes the computed trajectory. If an obstacle deviates from its predicted trajectory, the planner re-computes the robot's trajectory on the fly. The snapshots in Figure 3.15 show the robot executing the motion computed by the planner in one of our experiments. The robot's goal is to move from the back middle of the table to the front middle. Initially the obstacle in the middle is stationary, and the other two obstacles are moving towards the robot (snapshot 1). The robot dodges the faster-moving obstacle from the left and proceeds toward the goal (snaphot 2). The obstacle is then redirected twice (in snapshots 3 and 5) to block the

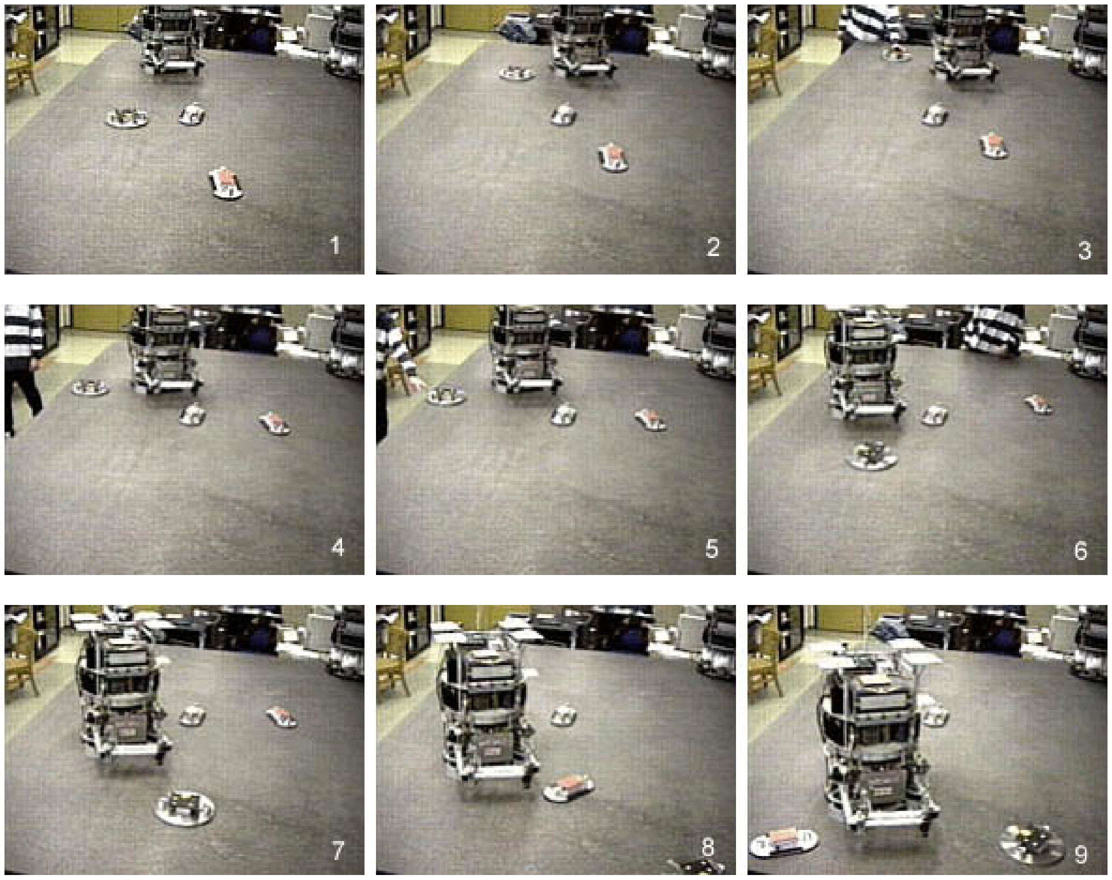

Figure 3.15. An example of on-line replanning for the air-cushioned robot (courtesy of Robert Kindel).

trajectory of the robot, causing it to slow down and stay behind the obstacle to avoid collision (snapshots 3–6). Right before snapshot 7, the rightmost obstacle is directed back towards the robot. The robot waits for the obstacle to pass (snapshot 8) and finally attains the goal (snapshot 9). Details on the implementation of our planner on this system and experimental results are available in [HKLR00].

The success of our planner on this real-time system indicates that the algorithm remains effective despite many adversarial conditions, including (i) severe dynamic constraints on the motion of the robot, (ii) moving obstacles, and (iii) various time delays and uncertainties inherent to an integrated system operating in a physical (as opposed to a simulated) environment.

Currently the trajectory computed by our planner is not optimized and may include unnecessary maneuvers. To improve the quality of the trajectory, the planner may continue sampling more milestones until it finds several solutions and pick the best one as the final answer according to some criteria. This method is used in the implementation of our planner on the hardware robot test bed described above. It fits well with the real-time nature of the system. Our planner is every efficient and often does not use up the allocated time to find the first solution. So it continues sampling new milestones until the end of the time period and outputs the best solution found. Although this idea is simple to implement and works reasonably well in practice, it does not have any formal guarantee of optimality. More sophisticated methods based on calculus of variations can be applied in a post-processing stage in order to generate a locally optimal trajectory [BLL90].

CHAPTER 4

Expansive Spaces

Experimental results in the preceding chapters have demonstrated that our planners are capable of solving difficult motion planning problems efficiently in environments with complex geometry and possibly various kinematic and dynamic constraints. However, some important questions cannot be answered by experiments alone. Our planners sample milestones at random; do they always find a trajectory if one exists? How does the performance of the planners depend on the complexity of environments?

There have been a few attempts aimed at providing theoretical justification for the good performance of randomized motion planners [LL96, BKL$^+$97, HLS99], but the success of these algorithms is still better observed than understood. In this chapter, we analyze the performance of our randomized planners formally. In particular, we show that our planners are probabilistically complete* by giving bounds on the number of milestones needed in order to find a trajectory with high probability, if one exists. In fact, these bounds state that if a solution trajectory exists, the failure probability of our planners decreases exponentially as more milestones are sampled. So a small number of milestones are sufficient to capture the connectivity of the configuration (or state) space and answer the queries.

In the rest of the chapter, we first introduce the notion of *expansive spaces*, which is intended to characterize the difficulty of sampling a good set of milestones (Section 4.1). We then illustrate the usefulness of the expansiveness definition in two analyses: uniform

*Recall that a motion planner is probabilistically complete if the probability of finding a trajectory converges to 1 quickly whenever such a trajectory exists.

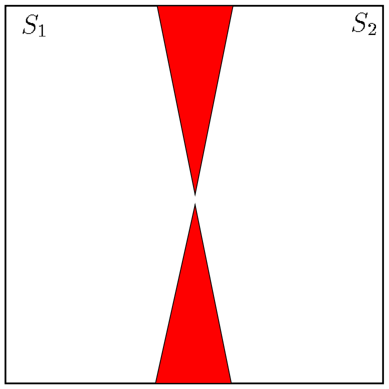

Figure 4.1. A free space with a narrow passage.

sampling for the all-pairs path planning problem (Section 4.2) and control-driven randomized expansion for motion planning under kinematic and dynamic constraints (Section 4.3). We end the chapter with some comments on the importance of these results (Section 4.4).

4.1 Expansiveness

To analyze the performance of our planners, we first need to characterize the complexity of the free space $\mathcal{F}$. As we have seen in the experiments, the presence of narrow passages in $\mathcal{F}$ poses significant difficulty for randomized planners. As a simple example, consider a path planning problem in the free space shown in Figure 4.1. Assume that there is no kinematic or dynamic constraints on motion. Let us say that two points in $\mathcal{F}$ *see* each other, or are mutually *visible*, if the straight-line segment between them lies entirely in $\mathcal{F}$. The *visibility set* of a point $p \in \mathcal{F}$ is then the set of points in $\mathcal{F}$ that p sees. The free space in Figure 4.1 consists of two subsets S_1 and S_2 separated by a narrow passage. Few points in S_1 see a large fraction of S_2, and therefore the probability that a randomized planner picks a milestone in S_1 whose visibility set contains a large fraction of S_2 is very small. This makes it difficult to connect milestones in S_1 and milestones in S_2.

More generally, let the *lookout* of a subset $S \subset \mathcal{F}$ be the set of points in S that can see a large fraction of the points path-connected to S, but outside of S. In our example, the set S_1 has a very small lookout: few points in S_1 see a large fraction of S_2 (Figure 4.2). The

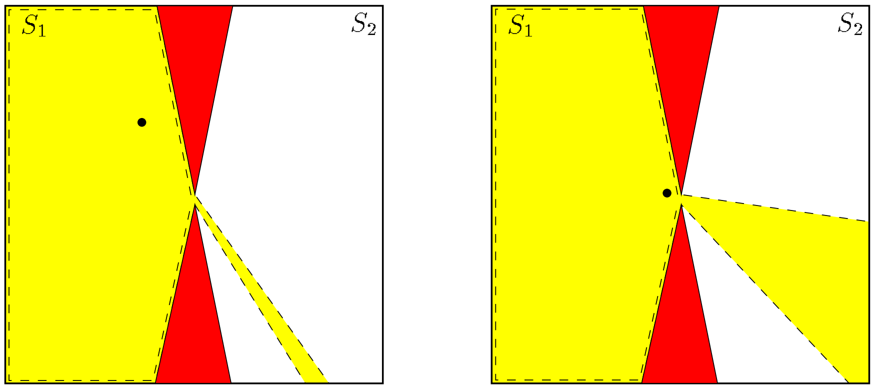

Figure 4.2. Lookout sets. The set S_1 has a small lookout, because only a small subset of points in S_1 near the narrow passage can see a large fraction of S_2. The areas with dashed boundaries indicate visibility sets.

example suggests that we can characterize narrow passages by the size of lookout sets. If $\mathcal{F}$ contains a subset that has a small lookout, the planner may have difficulty sampling a set of milestones that correctly captures the connectivity of the free space.

Path planning is, of course, a special case of the kinodynamic motion planning problem considered in Chapter 3. The basic issues are the same in both cases, but for kinodynamic motion planning, the notion of visibility (connecting milestones with straight-line paths) is inadequate. Our kinodynamic motion planner generates a different kind of roadmaps, in which trajectories between milestones may be neither straight nor reversible. This leads us to generalize the notion of visibility to that of *reachability*.

Given two points p and p' in the free space $\mathcal{F}$, p' is *reachable* from p if there exists a control function $u\colon [a, b] \mapsto \Omega$ that induces an admissible trajectory from p to p'. If p' remains reachable from p by using $u \in \mathcal{U}_\ell$, a piecewise-constant control with at most ℓ constant segments as defined in Section 3.4, then we say that p' is *locally reachable*, or *ℓ-reachable*, from p. Let $\mathcal{R}(p)$ and $\mathcal{R}_\ell(p)$ denote the set of points reachable and ℓ-reachable from p; we call them the *reachability set* and the *ℓ-reachability set* of p. For any subset $S \subset \mathcal{F}$, the reachability (ℓ-reachability) set of S is the union of the reachability (ℓ-reachability) sets of all points in S:

$$\mathcal{R}(S) = \bigcup_{p \in S} \mathcal{R}(p) \quad \text{and} \quad \mathcal{R}_\ell(S) = \bigcup_{p \in S} \mathcal{R}_\ell(p).$$

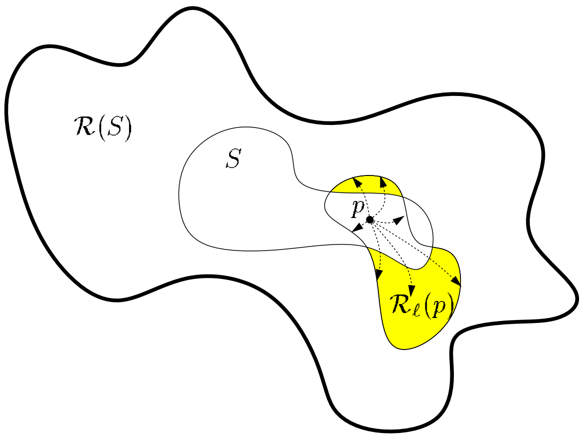

Figure 4.3. The lookout of a set S.

Formally we define the lookout of a subset $S \in \mathcal{F}$ as the set of all points in S whose ℓ-reachability sets overlap significantly with their reachability sets outside of S (Figure 4.3):

Definition 4.1 Let β be a constant in $(0, 1]$. The β-lookout of a set $S \subset \mathcal{F}$ is

$$\beta\text{-LOOKOUT}(S) = \{p \in S \mid \mu(\mathcal{R}_\ell(p) \setminus S) \geq \beta\, \mu(\mathcal{R}(S) \setminus S)\},$$

where $\mu(S)$ denotes the volume of a set $S \subset \mathcal{F}$.

The free space $\mathcal{F}$ is expansive if the reachability set of every point in $\mathcal{F}$ has a large lookout:

Definition 4.2 Let α and β be constants in $(0, 1]$. For a point $p \in \mathcal{F}$, the set $\mathcal{R}(p)$ is (α, β)-expansive if for any connected subset $S \subset \mathcal{R}(p)$,

$$\mu(\beta\text{-LOOKOUT}(S)) \geq \alpha\, \mu(S).$$

The free space $\mathcal{F}$ is (α, β)-expansive if for every point $p \in \mathcal{F}$, $\mathcal{R}(p)$ is (α, β)-expansive.

To better grasp these definitions, think of S as the ℓ-reachability set of a set M of sampled milestones. If α and β are both large, then it is easy to pick additional points in S so that adding them to M results in expanding S significantly. In fact, we will show that with high probability, the ℓ-reachability set of the sampled milestones expands quickly to

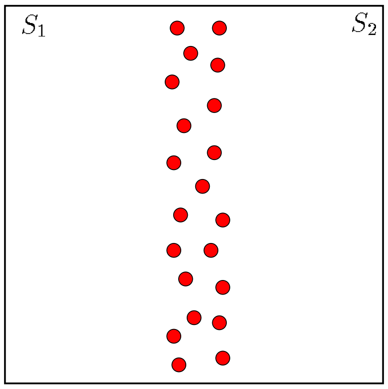

Figure 4.4. A free space punctured by many small obstacles.

cover most of $\mathcal{R}(s_{\text{init}})$, the reachability set of the initial state; hence, if the goal state lies in $\mathcal{R}(s_{\text{init}})$, the planner will find an admissible trajectory quickly with high probability.

Let us look again at the path planning example in Figure 4.1. Since there are no motion constraints, $\mathcal{R}(p)$ is simply the connected component of $\mathcal{F}$ containing p, in this case, the entire free space $\mathcal{F}$, because $\mathcal{F}$ is a single connected component. The ℓ-reachability set $\mathcal{R}_\ell(p)$ of p is the visibility set of p. The lookout of S_1 is a small subset of S_1 located near the passage between S_1 and S_2. So if β is large, the relative volume α of the lookout, will be small. We can increase the value of α, by choosing a smaller value for β, but α and β cannot both be made large simultaneously.

As another interesting illustration of the definition, consider the environment shown in Figure 4.4. The free space $\mathcal{F}$ is punctured by many small obstacles in a strip near the middle of $\mathcal{F}$. The subsets S_1 and S_2 to the left and right of the strip are connected through *many* narrow passages. By increasing the number of obstacles and decreasing their sizes, we can create many narrow passages whose clearances become arbitrarily small, yet α and β remain large. Consequently this is a rather simple environment for the randomized path planner according to our characterization; we have verified through experiments that it is indeed the case.

The parameters α and β measure the extent to which the space is expansive. The smaller these parameters are, the less expansive the free space is. We will show, in the following sections, that the cost of sampling a good set of milestones increases as α and β get smaller.

The notion of expansiveness defined here is entirely consistent with the more restrictive version for path planning (see [HLS99]). As shown in our example, if ℓ-reachability sets are specialized to visibility sets, we get back to the old definition. However, the new definition is more general and is useful for analyzing motion planning under kinematic and dynamic constraints as well.

Let M be a sequence of randomly-sampled milestones $p_0, p_1, p_2, \ldots$ and $R_i = \bigcup_{j=0}^{i} \mathcal{R}_\ell(p_j)$ be the ℓ-reachability set of first i milestones $p_0, p_1, \ldots, p_i$. A milestone p_i is a *lookout point* if p_i lies in the lookout of R_{i-1}. Lemma 4.1 states that the ℓ-reachability set of M spans a large volume if it contains enough lookout points.

Lemma 4.1 *Let $\mathcal{X} = \mathcal{R}(p)$ for some $p \in \mathcal{F}$. If a sequence M of randomly-sampled milestones $p_0 = p, p_1, p_2, \ldots$ contains k lookout points, then $\mu(\mathcal{R}_\ell(M)) \geq (1 - e^{-\beta k})\mu(\mathcal{X})$.*

Proof. Without loss of generality, assume $\mu(\mathcal{X}) = 1$. Let $p_{i_0}, p_{i_1}, \ldots, p_{i_k}$ be the subsequence of lookout points in M. For $i = 1, 2, \ldots$, we have

$$\mu(R_i) = \mu(R_{i-1}) + \mu(\mathcal{R}_\ell(p_i) \setminus R_{i-1}). \tag{4.1}$$

Thus $\mu(R_i) \geq \mu(R_j)$ for any $i \geq j$. In particular,

$$\mu(\mathcal{R}_\ell(M)) \geq \mathcal{R}_\ell(R_{i_k}), \tag{4.2}$$

for all k.

Using (4.1) with $i = i_k$ in combination with the fact that p_{i_k} is a lookout point, we get

$$\mu(R_{i_k}) \geq \mu(R_{i_k-1}) + \beta\, \mu(\mathcal{X} \setminus R_{i_k-1}).$$

Let $v_i = \mu(R_i)$. Since $\mu(\mathcal{X} \setminus R_{i_k-1}) = \mu(\mathcal{X}) - \mu(R_{i_k-1}) = 1 - v_{i_k-1}$, we have $v_{i_k} \geq v_{i_k-1} + \beta\,(1 - v_{i_k-1})$, which can be rewritten as

$$v_{i_k} \geq v_{i_{k-1}} + \beta\,(1 - v_{i_{k-1}}) + (1-\beta)(v_{i_k-1} - v_{i_{k-1}}). \tag{4.3}$$

Note that $i_k - 1 \geq i_{k-1}$ (Figure 4.5), and thus $v_{i_k-1} - v_{i_{k-1}} \geq 0$. It then follows from (4.3) that

$$v_{i_k} \geq v_{i_{k-1}} + \beta\,(1 - v_{i_{k-1}}).$$

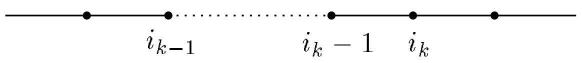

Figure 4.5. A sequence of sampled milestones.

Setting $w_k = v_{i_k}$ leads to the recurrence $w_k \geq w_{k-1} + \beta\,(1 - w_{k-1})$, which has the solution

$$w_k \geq (1-\beta)^k w_0 + \beta \sum_{j=0}^{k-1} (1-\beta)^j = 1 - (1-\beta)^k (1 - w_0).$$

Using the facts $w_0 \geq 0$ and $1 - \beta \leq e^{-\beta}$, we get $w_k \geq 1 - e^{-\beta k}$. Combined with (4.2), it yields

$$\mu(\mathcal{R}_\ell(M)) \geq 1 - e^{-\beta k}.$$

$\square$

4.2 Uniform Sampling for All-Pairs Path Planning

Our first use of expansiveness is to examine the efficiency of uniform sampling for the all-pairs path planning problem. Since we deal only with path planning in this section, it is sufficient to consider the more restrictive setting of connecting milestones with straight-line paths and visibility sets.

Recall that the algorithm for the all-pairs problem proceeds in two stages (see Section 2.4). The pre-computation stage builds a probabilistic roadmap G that captures the connectivity of the free space. With uniform sampling, we build G by picking new configurations uniformly at random and connecting pairs of collision-free configurations with straight-line paths in the free space. The query-processing stage first connects the initial and the goal configuration to G and then searches G for a path.

A good probabilistic roadmap G for the all-pairs problem should satisfy two requirements. First, it provides adequate coverage of the free space: the visibility set of the milestones in G should cover all but a small fraction of the free space $\mathcal{F}$. As a result, query configurations can be easily connected to milestones in G. Second, G must correctly represent the connectivity of the free space, meaning that there is a one-to-one correspondence between the connected components of G and those of $\mathcal{F}$.

Adequate coverage of the free space depends on a property of the free space called ϵ-*goodness* [BKL+97]:

Definition 4.3 Let ϵ be a constant in $(0, 1]$. A free space $\mathcal{F}$ is ϵ-good if for every $p \in \mathcal{F}$, $\mu(\mathcal{V}(p)) \geq \epsilon\mu(\mathcal{F})$, where $\mathcal{V}(p)$ is the visibility set of p.

A set of milestones provides an *adequate coverage* for an ϵ-good free space $\mathcal{F}$ if the volume of the points in $\mathcal{F}$ not visible from any of these milestones is at most $(\epsilon/2)\mu(\mathcal{F})$ [BKL+97]. For an ϵ-good free space $\mathcal{F}$, every connected component of $\mathcal{F}$ has volume at least $\epsilon\mu(\mathcal{F})$. So if a set of milestones provides an adequate coverage of $\mathcal{F}$, then every connected component of $\mathcal{F}$ contains at least one milestone. It has been shown that uniform random sampling generates a set of milestones that provides an adequate coverage of $\mathcal{F}$ with high probability; the number of milestones needed grows proportional to $(1/\epsilon)\ln(1/\epsilon\gamma)$, where γ is the probability that sampling uniformly at random fails to generate a set of milestones that provides an adequate coverage of $\mathcal{F}$ [BKL+97].

Here we would like to show that in addition to providing an adequate coverage, a set of uniformly-sampled milestones generates a probabilistic roadmap G whose connectivity conforms to the connectivity of the free space. By using the additional property of expansiveness, Theorem 4.1 establishes that with high probability, no two connected components of G lie in the same connected component of $\mathcal{F}$. So there is a one-to-one correspondence between the connected components of G and those of $\mathcal{F}$. Combined with the earlier result in [BKL+97], Theorem 4.1 implies that a PRM planner with uniform sampling finds a path between any two given configurations in $\mathcal{F}$ with high probability, if such a path exists.

In the rest of this section, we assume that the free space $\mathcal{F}$ is ϵ-good and that every connected component of $\mathcal{F}$ is (α, β)-expansive. We call such a space $\mathcal{F}$ an $(\epsilon, \alpha, \beta)$-expansive space. Our proof begins with the definition of the linking sequence of a point $p \in \mathcal{F}$ (Figure 4.6).

Definition 4.4 The linking sequence of a point $p \in \mathcal{F}$ is a sequence of points $p_0 = p, p_1, p_2, \ldots$ and a sequence of sets $V_0 = \mathcal{V}(p_0), V_1, V_2, \ldots \subset \mathcal{F}$ such that for all $i \geq 1$, $p_i \in \beta\text{-LOOKOUT}(V_{i-1})$ and $V_i = V_{i-1} \cup \mathcal{V}(p_i)$.

Note that the sets $V_0, V_1, V_2, \ldots$ are completely determined by the sequence of points

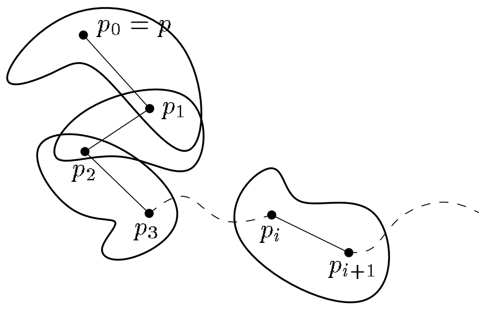

Figure 4.6. The linking sequence of p.

$p_0, p_1, p_2, \ldots$, and so for brevity, we refer to just the sequence of points $p_0, p_1, p_2, \ldots$ as the linking sequence of p.

The following two lemmas underscore the significance of this definition. Lemma 4.2 states that the visibility sets associated with a linking sequence spans a large volume if it is sufficiently long. Lemma 4.3 estimates the probability that a set M of uniformly-sampled milestones contains a linking sequence of a given length for some milestone in M. Together they imply that with high probability, a relatively short linking sequence of uniformly-sampled milestones spans a large volume.

Lemma 4.2 *Suppose that $\mathcal{F}'$ is a connected component of $\mathcal{F}$. Let $\mu_k = \mu(V_k)$ denote the volume of the* kth *set V_k determined by a linking sequence $p_0 = p, p_1, p_2, \ldots$ for a point $p \in \mathcal{F}'$. For $k \geq \beta^{-1} \ln 4 \approx 1.39/\beta$, $\mu_k \geq (3/4)\mu(\mathcal{F}')$.*

Proof. This result is an obvious consequence of Lemma 4.1. For path planning problems, the reachability set $\mathcal{R}(p)$ of a point $p \in \mathcal{F}$ is simply $\mathcal{F}'$, the connected component of $\mathcal{F}$ that contains p. By the definition of a linking sequence, every point p_i is a lookout point; a linking sequence of length k contains k lookout points. Thus it follows from Lemma 4.1 that $\mu_k \geq (3/4)\mu(\mathcal{F}')$, if $k \geq \beta^{-1} \ln 4$. □

Lemma 4.3 *Let M be a set of n milestones chosen independently and uniformly at random from the free space $\mathcal{F}$. Let $r = 1/\alpha\epsilon$. Given any milestone $p \in M$, there exists a linking sequence in M of length k for p with probability at least $1 - re^{-(n-k-1)/r}$.*

Proof. Assume, for convenience, that $\mu(\mathcal{F}) = 1$. Let L_i be the event that there exists a linking sequence in M of length i and $\overline{L_i}$ be the event that there does not exist such a sequence. Then

$$\begin{aligned}\Pr(\overline{L}_i) &= \Pr(\overline{L}_i \mid \overline{L}_{i-1})\Pr(\overline{L}_{i-1}) + \Pr(\overline{L}_i \mid L_{i-1})\Pr(L_{i-1}) \\ &\le \Pr(\overline{L}_{i-1}) + \Pr(\overline{L}_i \mid L_{i-1}).\end{aligned}$$

We would like to estimate $\Pr(\overline{L}_i \mid L_{i-1})$. Namely, given that there exists a linking sequence $p_0 = p, p_1, p_2, \ldots, p_{i-1}$ in M with length $i-1$, what is the probability that M contains no linking sequence of length i for p? All we need is that M contains no point lying in β-LOOKOUT(V_{i-1}). Note that $p, p_1, p_2, \ldots, p_{i-1}$ are conditioned and we cannot expect them to lie in β-LOOKOUT(V_{i-1}). However, the remaining $n-i$ points in M are unconditioned and chosen uniformly and independently from $\mathcal{F}$. Since $\mathcal{V}(p) = V_0 \subseteq V_{i-1}$, we have

$$\mu(V_{i-1}) \ge \mu(\mathcal{V}(p)) \ge \epsilon,$$

because $\mathcal{F}$ is ϵ-good. Further, since $\mathcal{F}$ is $(\epsilon, \alpha, \beta)$-expansive, we obtain

$$\mu(\beta\text{-LOOKOUT}(V_{i-1})) \ge \alpha\mu(V_{i-1}) \ge \alpha\epsilon = 1/r.$$

It follows that the probability that M does not contain a point in β-LOOKOUT(V_{i-1}) is at most

$$(1 - 1/r)^{n-i} \le e^{-(n-i)/r}.$$

Hence we have

$$\Pr(\overline{L}_i) \le \Pr(\overline{L}_{i-1}) + e^{-(n-i)/r}$$

and

$$\Pr(\overline{L}_k) \le \sum_{i=1}^{k} e^{-(n-i)/r} = e^{-(n-1)/r}\sum_{i=0}^{k-1} e^{i/r} = e^{-(n-1)/r}\frac{e^{k/r}-1}{e^{1/r}-1}.$$

Noting that $e^{1/r} - 1 \ge 1/r$, we obtain the desired bound

$$\Pr(\overline{L}_t) \le s e^{-(n-k-1)/r}.$$

So with probability at least $1 - re^{-(n-k-1)/r}$, M contains a linking sequence of length k for p. □

We are now ready to state our main result. It relates the notion of linking sequences to a set of randomly-sampled milestones. Suppose that a set M of milestones are sampled uniformly at random from $\mathcal{F}$. Let G be the probabilistic roadmap obtained by taking all the milestones in M as vertices and introducing an edge between any two milestones in M that can see each other. For every connected component $\mathcal{F}_j$ in $\mathcal{F}$, let $M_j \subseteq M$ be the set of milestones belonging to $\mathcal{F}_j$, and G_j be the subgraph of G containing the set M_j of vertices.

Theorem 4.1 *Let γ be a constant in $(0,1]$. Suppose a set M of $2n+2$ milestones, for $n \geq 8\ln(8/\epsilon\alpha\gamma)/\epsilon\alpha + 3/\beta$, is chosen independently and uniformly at random from the free space $\mathcal{F}$. Then, with probability at least $1-\gamma$, each of the roadmap graphs G_j is a connected graph.*[†]

Proof. Again assume, without loss of generality, that $\mu(\mathcal{F}) = 1$. Suppose that we sample a total of $2n+2$ milestones from $\mathcal{F}$. Consider any two milestones p and q in M_j for some j. Divide the rest $2n$ milestones into two subsets, M' and M'', of n milestones each. It follows from Lemma 4.3 that any milestone in $\{p\} \bigcup M'$ has a linking sequence of length k in M' with probability at least $1 - re^{-(n-k)/r}$. The same holds for any milestone in $\{q\} \bigcup M'$. Let $V_k(p)$ and $V_k(q)$ be the visibility sets determined by the linking sequences of length k for the two milestones. By Lemma 4.2, both sets have volume at least $(3/4)\mu(\mathcal{F}_j)$ if we choose $k = 1.5/\beta$, and hence they must have a non-empty intersection with volume at least $(1/2)\mu(\mathcal{F}_j)$. We know that $\mu(\mathcal{F}_j) \geq \epsilon$, because in an ϵ-good space, the visibility region of any point in $\mathcal{F}_j$ must have volume at least ϵ. Since the n milestones in M'' are sampled independently at random, it follows that with probability at least $1-(1-\epsilon/2)^n \geq 1-e^{-n\epsilon/2}$, there is a milestone $x \in M''$ that lies in the intersection (see Figure 4.7). Note that both p and q have a path to x consisting of straight-line segments bending only at the linking sequence points, which of course belong to the set of milestones M_j. This means that there is a path from p to q through x using only the edges of the roadmap graph R_j.

Let B denote the event that p and q fail to be connected. Event B occurs if the sets in the linking sequences of p and q do not intersect or no point of M'' lies in the intersection, and therefore $\Pr(B) \leq 2re^{-(n-k)/r} + e^{-n\epsilon/2}$. Choosing $n \geq 2k$ and recalling $r = 1/\alpha\epsilon$,

[†]For clarity of exposition, we have chosen a slightly larger value of n than necessary. Using a more refined estimate of n will complicate the technical details in the following proof.

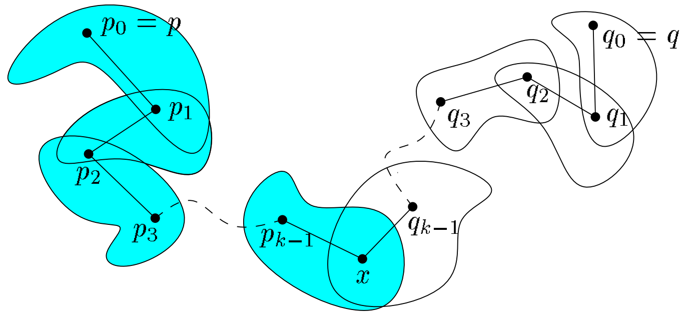

Figure 4.7. Linking sequences for p and q.

we have

$$\Pr(B) \leq 2re^{-n/2r} + e^{-n\epsilon/2} \leq 2re^{-n/2r} + e^{-n/2\alpha r} \leq 3re^{-n/2r}.$$

A graph R_j fails to be a connected graph if any pair of nodes $p, q \in M_j$ fail to be connected. The probability is at most

$$\begin{aligned} \binom{n}{2}\Pr(B) &= \binom{n}{2} 3re^{-n/2r} \\ &\leq 2n^2 re^{-n/2r} \\ &\leq 2re^{-(n-4r\ln n)/2r} \\ &\leq 2re^{-n/4r}, \end{aligned}$$

where the last inequality follows from the observation that $n/2 \geq 4r\ln n$ for $n \geq 8r\ln 8r$. Now requiring also that $n \geq 8r\ln(8r/\gamma)$, we have

$$\begin{aligned} 2re^{-n/4r} &\leq 2re^{-2\ln(8r/\gamma)} \\ &\leq 2r(\gamma/8r)^2 \\ &\leq \gamma. \end{aligned}$$

Clearly it is sufficient to choose $n \geq 8r\ln(8r/\gamma) + 2k$. Substituting $r = 1/\alpha\epsilon$ and $k = 1.5/\beta$ into the expression for n, we obtain the desired result. $\square$

Theorem 4.1 provides an upper bound on the number of milestones needed to build a good roadmap with high probability using uniform random sampling. Interestingly the

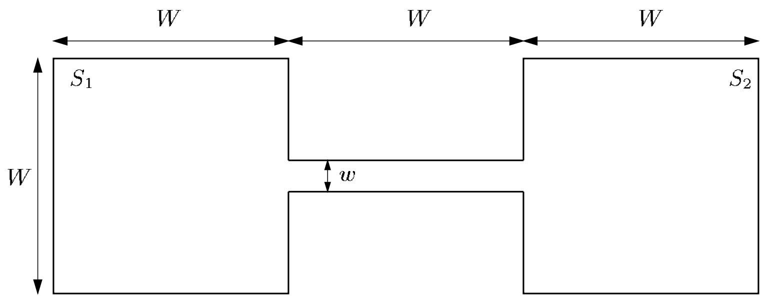

Figure 4.8. An $(\epsilon, \alpha, \beta)$-expansive free space with $\epsilon, \alpha, \beta \sim w/W$.

bound does not explicitly mention the dimension of the configuration space, because the definition of expansiveness is based solely on the visibility properties of the configuration space, which are stated in terms of volumes of subsets in $\mathcal{F}$. However, the dependence on the dimension of $\mathcal{C}$ is implicit in the size of the parameters ϵ, α, and β. To illustrate this point, consider the example in Figure 4.8. The free space consists of two squares, S_1 and S_2, connected by a narrow corridor. Each square has sides of length W, and the rectangular corridor has length W and width w with $w \ll W$. Up to a constant factor, each of the parameters ϵ, α, and β is on the order of w/W. Indeed, the points with the smallest visibility set are located in the corridor. Each such point has a visibility set of volume approximately $3wW$. Since the volume of the free space is $(2W + w)W$, $\epsilon \approx 3wW/((2W + w)W) \sim w/W$. Furthermore, only a small subset of S_1 with volume approximately wW, contains points, each of which sees a set of volume approximately $2wW$ in $S_1 \backslash \mathcal{F}$, and therefore $\alpha \approx wW/W^2 \sim w/W$ and $\beta \approx 2wW/((W + w)W) \sim w/W$. In the n-dimensional version of this example, two hyper-cubes, each having volume W^n, are connected by a hyper-parallelepipedic corridor that has size w in k dimensions ($1 \leq k \leq n - 1$) and size W in the rest $n - k$ dimensions. The parameters ϵ, α, and β are all on the order of $(w/W)^k$. Therefore the number of milestones needed to build a good roadmap is exponential in k for this example.

An alternative bound for the number of milestones needed can be obtained from the path-clearance assumption [BKL+97]. Consider a collision-free path between any two configurations q and q' in the same connected component of $\mathcal{F}$. Let L be the length of the path and δ be its clearance, which is defined as the minimum distance from all the points on the path to the boundary of $\mathcal{F}$.

Theorem 4.2 *Let γ be a constant in $(0, 1]$, and b be the constant $2^{-n}\mu(\mathcal{B}_1)/\mu(\mathcal{F})$, where $\mathcal{B}_1$ denotes the unit ball in R^n. With probability at least $1-\gamma$, a roadmap of $(1/b\delta^n)\ln(2L/\gamma\delta)$ milestones contains a connected component which has two milestones m and m' such that q sees m and q' sees m'.*

In the n-dimensional version of the example in Figure 4.8, the maximum clearance of a path going through the narrow passage is always $w/2$ for any integer k such that $1 \leq k \leq n-1$. The bound in Theorem 4.2 is always exponential in n, even if the passage is wide in most dimensions. Our new bound based on expansiveness yields a number of milestones that is only exponential in k.

Note also that a straight path between two configurations of $\mathcal{F}$ for one parameterization of the configuration space $\mathcal{C}$ may not be a straight path for another parameterization of $\mathcal{C}$. So the visibility properties in $\mathcal{F}$, hence the values of ϵ, α, and β, depend on how $\mathcal{C}$ is parameterized, though the connectivity of $\mathcal{F}$ does not depend on this parameterization. Choosing a parameterization of $\mathcal{C}$ yielding the largest values of ϵ, α, and β remains an open problem.

4.3 Single-Pair Problems

In Chapters 2 and 3, We looked at two single-query planners, one for the path planning problem (Section 2.4) and the other for motion planning under kinematic and dynamic constraints (Section 3.4). Neither of the two planners pre-computes a roadmap. Instead they try to build a small roadmap on the fly in order to answer the given query.

Our kinodynamic motion planner is a generalization of the path planner in Section 2.4. It picks new milestones by integrating randomly-sampled controls rather than connecting milestones with straight-line paths. If we restrict the set of available controls to generate straight-line paths only, these two algorithms are very similar. So we consider only the analysis of the more general case, but the result applies to the path planner as well.

4.3.1 Ideal Sampling

To simplify the presentation and focus on the most important aspects of the planner, let us assume for now that we have an ideal sampler IDEAL-SAMPLE that picks a point uniformly at random from the ℓ-reachability set of existing milestones. If it is successful, IDEAL-SAMPLE returns a new milestone p' and a trajectory from an existing milestone p to p'. With ideal sampling, the planning algorithm can be stated as follows:

Algorithm 4.1 Randomized expansion with IDEAL-SAMPLE.

```
1. Initialize a tree $T$ with $p_0 = s_{\text{init}}$; $R_0 \leftarrow \mathcal{R}_\ell(p_0)$.
2. **repeat**
3.   Invoke IDEAL-SAMPLE$(R_i)$, which samples a new milestone $p'$ and returns a trajectory from an existing milestone $p$ to $p'$ if the trajectory is admissible.
4.   **if** $p' \neq$ *nil* **then**
5.     Insert $p'$ into $T$.
6.     Create a directed edge $e$ from $p$ to $p'$, and store the trajectory with $e$.
7.     $R_{i+1} \leftarrow R_i \cup \mathcal{R}_\ell(p')$; $i \leftarrow i + 1$.
8.     **if** $p' \in$ ENDGAME **then** exit with SUCCESS.
```

This algorithm is similar to Algorithm 3.1, except that the use of IDEAL-SAMPLE replaces lines 3–5 in Algorithm 3.1.

4.3.2 Bounding The Number of Milestones

Let $\mathcal{X} = \mathcal{R}(s_{\text{init}})$ be the set of all points reachable from s_{init} under piecewise-constant controls. Our kinodynamic planner determines whether the goal lies in $\mathcal{X}$ by sampling a set of milestones; it terminates as soon as a milestone falls in the endgame region. The running time of the planner is thus proportional to the number of sampled milestones. In this subsection, we give a bound on the number of milestones needed in order to guarantee a milestone in the endgame region with high probability, if the intersection of the endgame region and $\mathcal{X}$ is non-empty.

Let M be a sequence of milestones $p_0 = s_{\text{init}}, p_1, p_2, \ldots$ generated by Algorithm 4.1.

We know from Lemma 4.1 that the ℓ-reachability set of M spans a large volume if M contains enough lookout points. Lemma 4.4 belows gives an estimate on the probability of this event. Combining the results from these two lemmas, we conclude that with high probability, the ℓ-reachability set of a relatively small number of milestones spans a large volume in $\mathcal{X}$.

The following results assume that $\mathcal{X}$ is (α, β)-expansive. For convenience, let us scale up all the volumes so that $\mu(\mathcal{X}) = 1$.

Lemma 4.4 *A sequence of n randomly-sampled milestones generated by Algorithm 4.1 contains k lookout points with probability at least $1 - ke^{-\alpha n/k}$.*

Proof. Let M be the sequence of milestones and L be the event that M contains k lookout points. We divide M into k groups of n/k consecutive milestones. Let L_i denote the event that the *i*th group contains at least one lookout point. Since the probability of M having k lookout points is greater than the probability of every group having at least one lookout point, we have

$$\Pr(L) \geq \Pr(L_1 \cap L_2 \ldots \cap L_k),$$

which implies

$$\Pr(\overline{L}) \;\leq\; \Pr(\overline{L}_1 \cup \overline{L}_2 \ldots \cup \overline{L}_k) \;\leq\; \sum_{i=0}^{k} \Pr(\overline{L}_i).$$

Each milestone picked by IDEAL-SAMPLE has probability α of being a lookout point, and thus $\Pr(\overline{L}_i)$, the probability of having no lookout point in the *i*th group, is at most $(1-\alpha)^{n/k}$. Hence

$$\Pr(L) = 1 - \Pr(\overline{L}) \geq 1 - k(1-\alpha)^{n/k}.$$

Note that $(1-\alpha)^{n/k} \leq e^{-\alpha n/k}$. So we have $\Pr(L) \geq 1 - ke^{-\alpha n/k}$. $\square$

The main result, stated in the theorem below, establishes a bound on the number of milestones needed in order to guarantee a milestone in the endgame region with high probability.

Theorem 4.3 *Let $g > 0$ be the volume of the endgame region in $\mathcal{X}$ and γ be a constant in $(0, 1]$. A sequence M of n milestones generated by Algorithm 4.1 contains a milestone in the endgame region with probability at least $1 - \gamma$, if $n \geq (k/\alpha)\ln(2k/\gamma) + (2/g)\ln(2/\gamma)$, where $k = (1/\beta)\ln(2/g)$.*

Proof. Let us divide the sequence M of milestones $p_0 = s_{\text{init}}, p_1, \ldots, p_n$ into two subsequences M_1 and M_2 so that M_1 contains the first n_1 samples and M_2 contains the rest $n_2 = n - n_1$ samples.

By Lemma 4.4, M_1 contains k lookout points with probability at least $1 - k(1-\alpha)^{n_1/k}$. If there are k lookout points in M_1, then by Lemma 4.1, the ℓ-reachability set $\mathcal{R}_\ell(M_1)$ has volume at least $1 - g/2$, provided

$$k \geq (1/\beta)\ln(2/g).$$

As a result, $\mathcal{R}_\ell(M_1)$ have a non-empty intersection of volume at least $g/2$ with the endgame region, and so does every set R_i for $i \geq n_1$, where R_i is the ℓ-reachability set of the first i milestones in M.

The procedure IDEAL-SAMPLE picks a milestone uniformly at random from the ℓ-reachability set of existing milestones, and therefore a milestone $p_i \in M_2$ falls in the intersection with probability $(g/2)/\mu(R_{i-1})$. Since $\mu(R_{i-1}) \leq \mu(\mathcal{X}) = 1$ for all i, and all the milestones are sampled independently, M_2 contains a milestone in the intersection with probability at least $1 - (1-g/2)^{n_2} \geq 1 - e^{-n_2 g/2}$.

If M fails to have a milestone in the goal, then either the ℓ-reachability set of M_1 does not have a large-enough intersection with the goal (event A) or no milestone of M_2 falls in the intersection (event B). From the preceding discussion, we know that $\Pr(A) \leq \gamma/2$ if $n_1 \geq (k/\alpha)\ln(2k/\gamma)$, and $\Pr(B) \leq \gamma/2$ if $n_2 \geq (2/g)\ln(2/\gamma)$. Choosing $n = n_1 + n_2 = (k/\alpha)\ln(2k/\gamma) + (2/g)\ln(2/\gamma)$ guarantees that $\Pr(A \cup B) \leq \Pr(A) + \Pr(B) \leq \gamma$. Substituting $k = (1/\beta)\ln(2/g)$ into the expression for n, we get the final result

$$n \geq \frac{\ln(2/g)}{\alpha\beta}\ln\frac{2\ln(2/g)}{\beta\gamma} + \frac{2}{g}\ln\frac{2}{\gamma}.$$

□

4.3.3 Approximating IDEAL-SAMPLE

The above analysis assumes the use of IDEAL-SAMPLE, which picks a new milestone uniformly at random from the the ℓ-reachability set of the existing milestones. One way to implement IDEAL-SAMPLE would be rejection sampling [KW86], which throws away a fraction of samples in regions that are more densely sampled than others. However, rejection sampling is not efficient: many potential candidates are thrown away in order to achieve the uniform distribution.

So instead, our planners try to approximate the ideal sampler. The approximation is much faster to compute, but generates slightly less uniform distribution. Recall that to sample a new milestone p', we first choose a milestone p from the existing milestones and then sample in the neighborhood of p. Every new milestone p' thus created tends to be relatively close to p. If we selected uniformly among the existing milestones, the resulting distribution would be very uneven; with high probability, we would pick a milestone in an already densely-sampled region and obtain a new milestone in that same region. Therefore the distribution of milestones tends to cluster around the initial configuration (or state). To avoid this problem, we associate with every milestone p a weight $w(p)$, which is the number of milestones in a small neighborhood of p, and pick an existing milestone to expand with probability inversely proportional to $w(p)$. So it is more likely to sample a region with a smaller number of milestones. The distribution $\pi_T(p) \propto 1/w(p)$ contributes to the diffusion of milestones over the free space and avoids oversampling.

To give a concrete example, let us put a uniform grid on the state space and assume that the neighborhood of p is the grid cell containing p. Due to the weighting, every cell covered by the ℓ-reachability set of existing milestones is sampled with the same probability (Figure 4.9a). Note that milestones inside the same cell are chosen with equal probability. This, however, causes the distribution to be slightly non-uniform, because the ℓ-reachability sets of close-by milestones may overlap. As a result, some regions are sampled more frequently than others. The grid cell shown in Figure 4.9b contains two milestones. Points in region C are more likely to be sampled than those in A or B. However, milestones with overlapping ℓ-reachability sets are more likely to be close to one another than milestones with no such overlapping. Therefore, it is reasonable to expect that weighting the milestones keeps the problem from worsening as the number of milestones grows. Furthermore we may limit the maximum number of milestones in each cell to avoid excessive overlapping of ℓ-reachability sets.

There is another issue that is specific to our planner for kinodynamic motion planning. In line 4 of Algorithm 3.1, we select u uniformly at random from $\mathcal{U}_\ell$ and integrate the equations of motion with u to obtain a new milestone p' in $\mathcal{R}_\ell(p)$. The distribution of p'

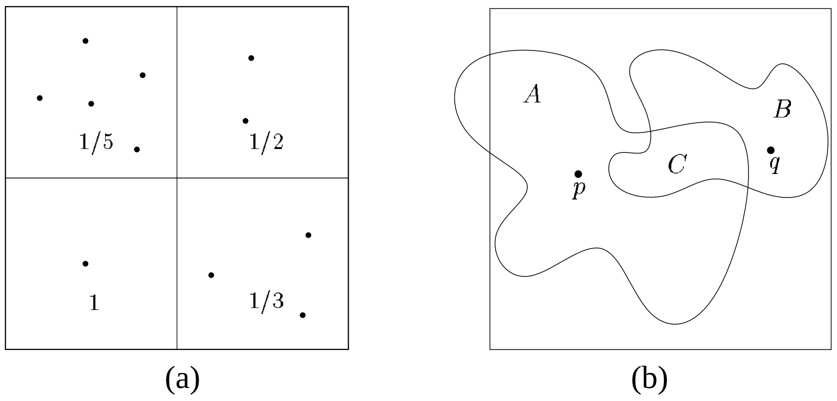

Figure 4.9. Approximating IDEAL-SAMPLE by weighting milestones. (a) Each of the four grid cells is chosen with equal probability regardless of the number of milestones in it. Every milestone is picked with probability proportional to the number in the cell containing the milestone. (b) All milestones within a cell are chosen with equal probability. So a point in region C, where the ℓ-reachability sets of p and q intersect, is sampled twice as likely as a point in A or B.

in $\mathcal{R}_\ell(p)$ is not uniform in general, because the mapping from $\mathcal{U}_\ell$ to $\mathcal{R}_\ell(p)$ may not be linear. In some cases, one may pre-compute a distribution $\pi_\mathcal{U}$ such that picking u from $\mathcal{U}_\ell$ with probability $\pi_\mathcal{U}(u)$ yields a uniform distribution of p' in $\mathcal{R}_\ell(p)$. In other cases, rejection sampling can be used locally. First pick several control functions $u_i, i = 1, 2, \ldots$ at random, and compute the corresponding p'_i. Then throw away some of them to achieve a uniform distribution among the remaining ones, and pick a remaining p'_i at random.

4.3.4 Choosing Suitable Control Functions

To sample new milestones, our kinodynamic planner (Algorithm 3.1) picks at random a piecewise-constant control function u from $\mathcal{U}_\ell$. Every $u \in \mathcal{U}_\ell$ has at most ℓ constant segments, each of which lasts for a time duration less than $\delta_{\max}$. The parameters ℓ and $\delta_{\max}$ are chosen according to the properties of each specific moving object.

In theory, ℓ must be large enough so that for any $p \in \mathcal{R}(s_{\text{init}})$, $\mathcal{R}_\ell(p)$ has the same dimension as $\mathcal{R}(s_{\text{init}})$. Otherwise, $\mathcal{R}_\ell(p)$ has zero volume relative to $\mathcal{R}(s_{\text{init}})$, and $\mathcal{R}(s_{\text{init}})$ cannot be expansive even for arbitrarily small values of α and β. This can only happen when some dimensions of $\mathcal{R}(s_{\text{init}})$ are not spanned directly by basis vectors in the control space Ω, but these dimensions can then be generated by combining several controls in Ω using Lie-brackets [BL93]. The mathematical definition of a Lie bracket can be interpreted as an infinitesimal "maneuver" involving two controls. Spanning all the dimensions of $\mathcal{R}(s_{\text{init}})$ may require combining more than two controls of Ω by imbricating multiple Lie brackets. At most $n-2$ Lie brackets are needed, where n is the dimension of the state space. Hence it is sufficient to choose $\ell = n-2$.

In general, the larger ℓ is, the greater α and β tend to be. So according to our analysis, fewer milestones are needed; on the other hand, the cost of integration and collision checking, when generating a new milestone, becomes more expensive. The choice of δ_{max} is somewhat related. A larger δ_{max} may result in greater α and β, but also lead the planner to integrate longer trajectories that are more likely to be inadmissible. Experiments show that ℓ and δ_{max} can be selected in relatively wide intervals without significant impact on the performance of the planner. However, if the values for ℓ and δ_{max} are too large, the approximation to IDEAL-SAMPLE becomes very poor.

4.4 Discussion

The bounds given in Theorems 4.1 and 4.3 provide a measure of the amount of work that the planners should do in order to build a good roadmap with high probability in an expansive space. Unfortunately we cannot compute this number effectively in advance, because it is difficult to calculate the values of ϵ, α, and β, except for very simple spaces. One may be tempted to use Monte Carlo techniques to estimate these values, but it seems that a reliable estimate would take as much time as building a satisfactory roadmap. Nevertheless these results are important. First, they tell us that the failure probability of our planners decreases exponentially with the number of milestones sampled. Second, the number of milestones needed increases only moderately when ϵ, α, and β decrease, *i.e.*, when the space becomes less expansive.

CHAPTER 5

Optimized Robot Placement in a Workcell

The efficiency of randomized motion planners makes them useful as a primitive, which we can use to accomplish more complex tasks. Although sometimes a direct invocation of motion planners does not solve the problem completely, the general idea of constructing the connectivity of configuration space via random sampling may nevertheless be helpful. In this chapter, we demonstrate the utility of randomized path planning in a practical application, the robot placement problem in the context of virtual prototyping.

Today robots are widely deployed in almost very manufacturing industry, but programming their motion remains a tedious task, because CAD systems usually have little support for dealing with objects in motion. To bridge the gap, virtual prototyping systems have been introduced to provide tools for checking collision among moving parts, computing and simulating motion, *etc.*, in order to help designers analyze how objects move or are moved during manufacturing and maintenance.

Randomized path planning has been used in virtual prototyping before as a tool for assembly maintainability studies [CL95]. Here we would like to consider a more complex problem: robot placement in a workcell. Specifically our goal is to find a base location for a robot manipulator so that specified tasks are executed as efficiently as possible. We present an algorithm that combines randomized motion planning with local iterative optimization to compute simultaneously a base location and a corresponding collision-free path that are optimized with respect to the execution time of tasks.

5.1 Overview

The robot placement problem considered here is motivated mainly by its applications in the manufacturing industry, where the base placement of a manipulator has a big impact on the cycle time of tasks executed, *e.g.*, spot welding, inspection, and part transfer. An automated means to determine the best placement can both increase the throughput of workcells and reduce set-up time. Our algorithm can also be used to position mobile manipulators, which are manipulators mounted on mobile bases. For many mobile manipulators, the base remains stationary while the mounted manipulator is in motion, because of operational constraints and increased control complexity [Ser95]. Thus positioning the mobile base for efficient operation of such a system is the same problem as that for a fixed-base manipulator.

A minimum requirement for the base placement of a manipulator is that the reachable workspace of the manipulator covers all the task points. Furthermore the robot should be placed to enable efficient task execution. Previous work on this problem usually considers the first and sometimes also the second criterion, but few systems take into account obstacles in the environment at the same time, partly because planning a collision-free path for a robot with many dofs is a difficult problem. However, robots share the space with part feeders and various other devices in a workcell, and must avoid collision with them while in motion (see Figure 5.1 for an example). We therefore believe that it is essential to consider the impact of obstacles on the placement of robots.

Our robot placement algorithm first computes a collision-free path for an initial base location, using the randomized path planner described in Chapter 2. It then deforms the computed path to obtain a locally optimal one and iteratively moves the base to better locations. At each iteration, it perturbs the base location and recomputes a new collision-free path. If the path planner were invoked every time, the computation would be prohibitively expensive. However, locally optimal paths for two different base locations are usually "close" to each other in the configuration space of the manipulator base and joints, if the two base locations are close. Exploiting this spatial "coherence", we use the path found in the previous iteration as a starting point for finding a new path in the current iteration. This allows us to save considerable computation time.

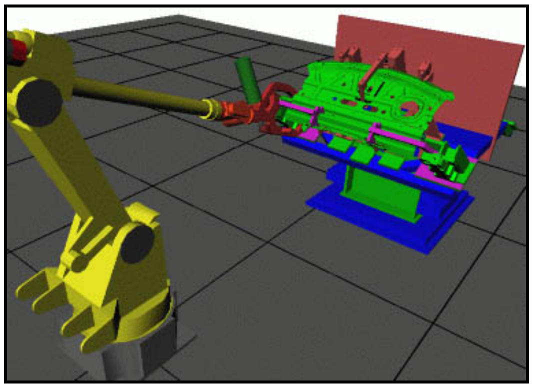

Figure 5.1. A robot spot-welding the side-panel of a car. It has to maneuver among fixturing devices and reach underneath the side-panel in order to access weld-points.

5.2 Related Work

The robot placement problem requires optimizing not only the motion of the robot but also its base location. Several variants of the problem have been proposed in the literature. Seraji considers the placement of a seven-dof Robotics Research arm by analyzing its reachability [Ser95]. Kolarov presents an algorithm that places a robot made up of telescoping links amid obstacles in a planar environment [Kol95]. Ozedou formulates the base placement problem as that of kinematic synthesis and solves it with generic optimization techniques [Oue97]. These work addresses the placement problem mainly from the reachability point of view. The algorithm proposed by Feddema places a manipulator for minimum-time joint-coordinated motion [Fed96], but it assumes an environment with no obstacles. Hwang and Watterberg's formulation of the placement problem [HW96] is more closely related to ours. Their algorithm discretizes the space of all base locations and uses a path planner to search the space exhaustively in order to find the optimal solution. They report that the exhaustive search took 50 hours on a grid of 175 base locations for an environment with relatively simple geometry.

5.3 Configuration-Space Formulation

Assume, for the moment, that the base of a manipulator M is fixed. The configuration of M is then its joint angles. Let $\mathcal{C}$ be the configuration space of M. A path in $\mathcal{C}$ is a

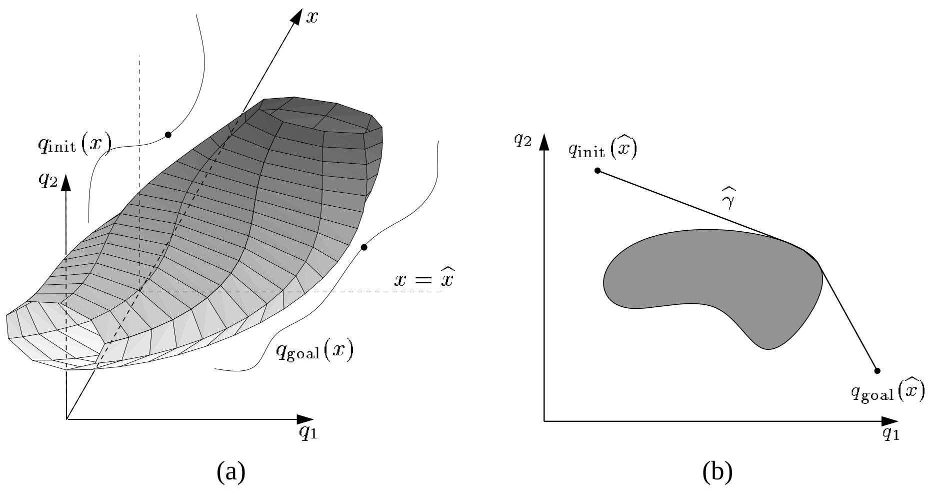

Figure 5.2. The base placement problem in the configuration space. The robot is a two-dof planar arm with joint angles q_1 and q_2. The base of the robot lies on a line parameterized by x. (a) Schematic drawing of the configuration space of the robot base and joints. Shaded parts indicate obstacles. (b) Cross section for the optimal base location $\hat{x}$. It contains the optimal path $\hat{\gamma}$.

continuous mapping $\gamma\colon [0,1] \mapsto \mathcal{C}$. The cost of a path can be measured in many ways, the most important ones being the time and energy that it takes for a manipulator to execute the path.

If the end-effector frame (position and orientation) of M is T, then for a given base location x, we can solve for a configuration of the manipulator joints that achieves T via inverse kinematics (IK). The solutions define a mapping $q\colon X \to \mathcal{C}$ from the space X of all base locations to the space $\mathcal{C}$ of manipulator joint configurations. Since the IK solution is not unique in general, the mapping can be one-to-many. However, for the simplicity of presentation, we assume that the IK solution is unique, when it exists. Our algorithm easily generalizes to deal with multiple IK solutions.

In a base placement problem, we are given an initial end-effector frame T_{init} and a goal end-effector frame T_{goal}. Given a base location x, let $q_{\text{init}}(x)$ and $q_{\text{goal}}(x)$ be the IK solutions for T_{init} and T_{goal}, respectively. Our objective is to find the best base location $\hat{x} \in X$ such that the path between the $q_{\text{init}}(\hat{x})$ and $q_{\text{goal}}(\hat{x})$ has the minimum cost. We can write this more compactly as

$$\min_{x \in X} \min_{\gamma \in \Gamma_x} L(\gamma),$$

where $L(\gamma)$ is the cost of a path γ, and Γ_x denotes the set of all collision-free paths such that for each $\gamma \in \Gamma_x$, $\gamma(0) = q_{\text{init}}(x)$ and $\gamma(1) = q_{\text{goal}}(x)$. Figure 5.2 illustrates the statement for a planar robot with two one-dof revolute joints. The base of the robot is assumed to be constrained on a line. So the configuration space of the robot base and joints is three-dimensional. For each fixed base location, there is a two-dimensional cross section of joint angles. Finding the best base placement $\widehat{x}$ is equivalent to finding the cross section that contains the optimal path.

5.4 The Base Placement Algorithm

Our algorithm for robot placement makes use of two sub-algorithms. The first one is a randomized path planner. Given a a fixed base location x, the planner generates a collision-free path γ between two configurations $q_{\text{init}}(x)$ and $q_{\text{goal}}(x)$. The path γ often contains many unnecessary turns because of the random steps taken by the planner and must be optimized. The second sub-algorithm takes γ as input and computes a locally optimal piecewise-linear path. The restriction to piecewise-linear paths is not severe, since any reasonable path can be well-approximated by a piecewise-linear one. Using these two building blocks, the robot placement algorithm iteratively searches for the best base location.

Our algorithm does not represent configuration-space obstacles (C-obstacle) explicitly. Instead, a collision checker determines whether a configuration is free or not. To determine whether a path is collision-free, we can discretize it into a sequence of configurations and regard the path free if all these configurations are free. Problems may arise if the discretization is not fine enough, but as explained in Section 2.5, they can be dealt with by making use of the distance information returned by the collision checker.

In the following, we first look at the overall algorithm and then discuss the choice of the two sub-algorithms.

Searching for the optimal placement In principle, given a path-planning and a path-optimization algorithm, we can search for a good base location by brute force. At each candidate base location, simply call the path planner to get a collision-free path and then optimize it. However, this would be very expensive computationally due to the repeated invocation of the path planner.

Notice, however, that if $\mathcal{B}_x$ is the C-obstacle for a manipulator placed at x, for sufficiently small Δx, $\mathcal{B}_{x+\Delta x}$ can be obtained from $\mathcal{B}_x$ by a small deformation. Therefore a collision-free path for a manipulator placed at x is likely to be collision-free in $\mathcal{C}_{x+\Delta x}$, the configuration space for the manipulator placed at $x + \Delta x$. If not, we may hope to transform the path into a collision-free one by a small deformation.

Using this observation and a fast path optimization routine, we can quickly recompute an optimal path at each new base location and iteratively move the base towards the best location. The algorithm is described in the pseudocode below. It selects new base locations by randomly sampling the neighborhood of the best one found so far.

Algorithm 5.1 Robot placement.

```
1.  Find a collision-free path for an initial base location $x_0$, and optimize it to obtain a
    piecewise-linear path $\gamma = (v_1, v_2, \ldots, v_n)$, where $v_1 = q_{\text{init}}(x_0)$, $v_2, \ldots, v_{n-1}, v_n =$
    $q_{\text{goal}}(x_0)$ are the vertices of $\gamma$.
2.  **if** a collision-free path $\gamma$ is found
3.    **then** $x \leftarrow x_0$.
4.    **else**  **return** FAILURE.
5.  **repeat**
6.     $x' \leftarrow x + \Delta x$ for some random $\Delta x$.
7.     Compute $q_{\text{init}}(x')$ and $q_{\text{goal}}(x')$ for the base location $x'$ using the manipulator IK. If
       $q_{\text{init}}(x')$ or $q_{\text{goal}}(x')$ is in collision, then continue to the next iteration (line 6).
8.     **if** the path $(q_{\text{init}}(x'), v_2, \ldots, v_{n-1}, q_{\text{goal}}(x'))$ is collision-free
9.       **then** $\gamma' \leftarrow (q_{\text{init}}(x'), v_2, \ldots, v_{n-1}, q_{\text{goal}}(x'))$.
10.      **else**  Sample $K$ new configurations in each of the neighborhoods of $v_2, v_3, \ldots, v_{n-1}$,
                for some constant $K$.
11.             Try to find a path $\gamma'$ between $q_{\text{init}}(x')$ and $q_{\text{goal}}(x')$ through the new config-
                urations. Continue to the next iteration (line 6) if no collision-free path is
                found.
12.    Optimize $\gamma'$.
13.    **if** $L(\gamma') < L(\gamma)$ **then** $x \leftarrow x'$; $\gamma \leftarrow \gamma'$.
14. **until** the termination condition is satisfied.
15. **return** $x$ and $\gamma$.
```

In lines 10–11 of the algorithm, we find a collision-free path at the new base location by randomly sampling in the neighborhood of the path for the previous base location. This deformation technique is simple to implement and usually quite effective. However, at certain critical locations, no local deformation is sufficient to obtain a new path. In this case, we call the planner to replan a new path, but replanning does not happen very frequently in practice.

The termination condition in line 14 can be implemented in various ways. A simple one is to keep track of the improvement between successive iterations and terminate if the improvement falls below some threshold. We also bound the maximum number of iterations.

Note that the outcome of the algorithm may depend on the initial location x_0 and the initial path computed by the randomized path planner. To alleviate the impact of this bias, one may run the algorithm several times with different values for x_0 and initial paths, and keep the best result.

Randomized path planning The goal here is to find a collision-free path between two given configurations q_{init} and q_{goal} for a fixed base location. Although the obstacles that we consider are stationary, the base location of the robot changes between iterations, and therefore pre-computing a roadmap is not useful. The path planner described in Section 2.4 is a good candidate for the solution. Unlike most other PRM planners, it is intended for single-query problems and builds only a small roadmap necessary to process the query. It is also very efficient in general. Of course, other fast path planners may be used as well. Note that if a manipulator has multiple IK solutions, we can easily extend our planner by considering multiple q_{init} and q_{goal}.

Optimizing a path The specific choice of a path-optimization algorithm depends on the robot and the cost measure used. To fit into the overall algorithm, path optimization must be very fast because it is performed at every candidate base location.

A straightforward method for path optimization is to start with some initial path γ, sample a large number of free configurations in the neighborhood of γ, find the best path going through these configurations, and repeat the same operation on the new path. This method is very general and can handle many different cost functions, but it can be quite slow.

Here we define the cost of a path to be the time that it takes a manipulator to execute the path, assuming that all joints can achieve maximum velocity in negligible amount of time. As shown in Section 2.6, under this definition, the cost L of a piecewise-linear path $\gamma = (v_1, v_2, \ldots, v_n)$ is $L(\gamma) = \sum_{i=1}^{n-1} \tau(v_i, v_{i+1})$, where $\tau(v_i, v_{i+1})$ is the time needed to travel a straight-line path between v_i and v_{i+1}. Since the function τ satisfies the triangle inequality, we can use ADAPTIVE-SHORTCUT for path optimization, which is much more efficient than the general technique described in the above paragraph. However, in practice, this definition of path cost does not take into account manipulator dynamics and is only an approximation to the time that it takes a manipulator to execute a path. It is a good approximation if the manipulator can reach the maximum speed quickly. If dynamics must be considered, we have to resort to the general technique or some other path optimization methods based on calculus of variations, but they are likely to be much slower than ADAPTIVE-SHORTCUT.

5.5 Computed Examples

We have implemented our algorithm in C++ and tested it on several data sets. Four examples in our test suite are shown in Figure 5.3. They vary in the complexity of workspace and motion needed to complete the specified tasks. Scene 1 is a simple blocks-world, and the motion required of the robot is straightforward. Scene 2 is much more complex in terms of both the workspace geometry and the motion required. The robot has to go through openings in the window and sun-roof of the car in order to reach the task-points. Scene 3 has relatively simple geometry, but in one of the tasks (p2), the robot has to execute complicated maneuvers in order to pull the big end-effector through a small rectangular hole. Scene 4 is a large CAD model containing about 72,100 triangles. The robot has to maneuver among fixturing devices and reach under the side-panel of a car in order to access the task-points. Scenes 1–3 were synthesized for testing; scene 4 was derived from CAD data provided to us by General Motors. For each of these scenes, we specify a number of tasks in the form of a pair of initial and goal end-effector frames (Cartesian frames at the wrist-point of the robot), labeled by p1, p2, or p3 in the figures.

Two types of robot are used in the tests. Scene 1 and scene 2 use a PUMA robot that has six dofs and about 460 triangles describing its geometry. Scene 3 and scene 4 use a modified version of a FANUC-200 robot. We have replaced the four-bar linkage of the

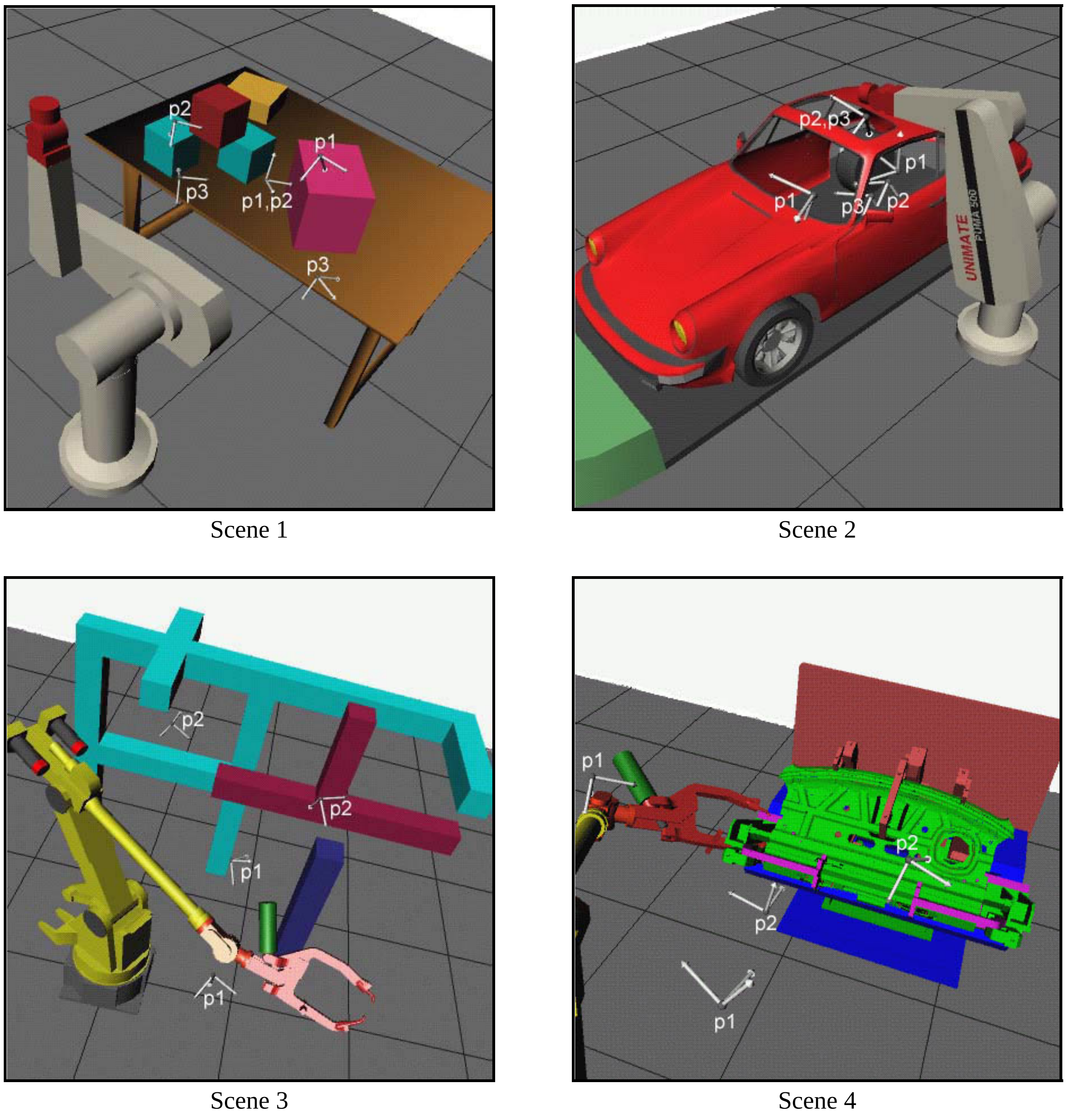

Figure 5.3. Four test scenes for the robot placement problem.

Table 5.1. Running time and cost of paths for the tested scenes.

Scene	N_{tri}	Task	T_{plan} (sec.)	N_{plan}	T_{opt} (sec.)	N_{opt}	C_{init} (sec.)	$C_{\text{i-opt}}$ (sec.)	C_{final} (sec.)
1	560	p1	0.3	309	15.1	9955	1.36	0.52	0.32
		p2	0.3	362	8.8	7016	1.08	0.48	0.39
		p3	0.3	353	9.0	6831	1.26	0.77	0.50
2	21400	p1	3.4	2831	39.8	17647	2.25	1.51	1.16
		p2	4.2	3806	31.7	10053	2.57	1.53	1.08
		p3	16.5	14794	34.1	10255	2.80	1.69	1.44
3	1368	p1	4.4	4331	25.5	17859	1.13	0.84	0.48
		p2	197.0	210829	55.4	23566	2.17	1.43	1.30
4	72100	p1	11.4	7425	107.4	24436	1.78	1.18	0.82
		p2	25.1	28263	133.6	19453	1.24	0.79	0.59

FANUC-200 by a simple revolute joint in order to simplify the IK solution. The geometry of the robot is mostly preserved. The modified robot has six dofs and about $1,260$ triangles.

The results of our experiments are shown in Table 5.1. The running times were collected on an SGI Indigo 2 workstation with an 195 MHz MIPS R10000 processor and 384 MB memory. In all the tests, our algorithm took only a few minutes to finish the computation, and less than a minute for simpler problems. The cost of the paths was reduced by about 50% by choosing a good base location and optimizing the path.

Column 2 of the table lists the total number of triangles contained in all the objects in the environment, including the robot. It gives an idea of the complexity of the environment. Columns 4 and 5 give the running time and the number of collision checks needed to find an initial collision-free path. For ease of implementation, we used the Scheme 2.2 with uniform sampling strategy for path planning. The algorithm performed reasonably well in these examples. All problems except for one were solved in less than 30 seconds. In the case where the planner took a long time, the robot must pull the welding-gun (end-effector) through a small hole, a very challenging situation for a randomized motion planner. Columns 6 and 7 show the running time and the number of collision checks spent searching for the best base placement and a corresponding collision-free path. The running times range from a few seconds to 2–3 minutes. Recall that at each new base location, the algorithm must find a new collision-free path and optimize it. Both operations require a large number of collision checks and are very costly. The computation time was significantly reduced by

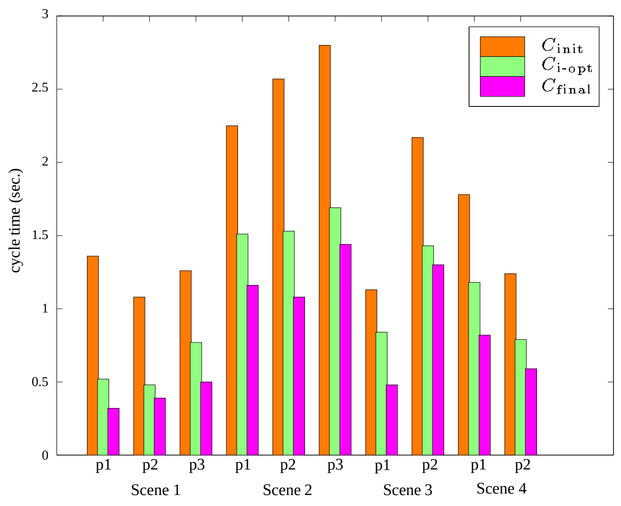

Figure 5.4. Reduction in path cost for the test cases.

exploiting the spatial coherence in the configuration space as we have discussed previously.

The last three columns of the table list respectively the cost of the initial path, the cost of the optimized path at the initial base location, and the cost of the path at the best base location found. The same data are shown in a bar graph (Figure 5.4). The horizontal axis of the graph represents the ten test cases, and the vertical axis represents the path cost. By comparing the costs in columns 8 and 9 of Table 5.1, we see that ADAPTIVE-SHORTCUT was able to reduce the cost of paths by about 25–60%. Finding a good base location further reduced the cost by additional 10–30%.

5.6 Discussion

This chapter presents an algorithm that computes a locally optimal base location and a corresponding collision-free path for a robot manipulator to move between two end-effector frames in minimum amount of time. We have tested this algorithm on both synthesized examples and real-life CAD data from the automotive industry. Experiments show that the algorithm can significantly reduce the cycle time by choosing a good base location and optimizing the path, and therefore improve the productivity of a workcell. In all our experiments, the computation was completed in a few minutes. The efficiency of our algorithm, we believe, derives from two factors. The first one is the use of a fast randomized path planner, which has enabled us to give a more realistic formulation of the robot placement problem that takes into account obstacles in the environment. The second factor is the spatial coherence of collision-free paths for base locations close to one another. By taking advantage of spatial coherence, we avoid paying the high computational cost of recomputing a collision-free path at each new base location from the scratch.

Currently our algorithm only considers a minimum-cost path between two end-effector frames. An immediate extension of the problem is to consider multiple end-effector frames. Given a set of end-effector frames and a partial order on them so that some frames have to be visited before others, we would like to find a path that visits each frame exactly once and obeys the partial order. This is closely related to the traveling salesman problem [L+85], which has been studied extensively. Despite its difficulty, many heuristic and approximation algorithms are available.

Our algorithm was originally developed for the base placement problem, but the approach can potentially be applied to kinematic synthesis problems, in which one tries to find a small set of parameters to configure a robot. Application of the algorithm to this more general class of problems is an interesting direction for future research.

CHAPTER 6

Conclusion

Motion planning is a provably hard computational problem; there is strong evidence that a complete algorithm will take time exponential in the number of dofs of a moving object [Rei79]. A fundamental challenge in motion planning is to design algorithms that are general and can efficiently handle a large number of dofs under complex motion constraints. In recent years, random sampling has emerged as an important framework for motion planning. In addition to being able to deal with many dofs, randomized motion planning algorithms are often simple to implement.

6.1 Summary of Main Results

We have presented efficient randomized algorithms for single-query motion planning. Our planners incrementally construct a roadmap in the free space by random sampling. Two specific planners have been discussed in detail. One considers the simpler problem of path planning, in which the only requirement is that the motion of moving objects remains collision-free. The other extends the basic idea to solve kinodynamic motion planning problems, which take into account non-holonomic and dynamic constraints on the motion. In the latter case, our algorithm encodes the motion constraints by a control system, a system of differential equations that characterizes all possible movements of an object M at each state. To sample new milestones, it first picks at random a control function u in the space of admissible control functions and maps u to a point in the state space of M by integrating the control system with u. Our planners have been tested extensively with both synthesized examples and real-life CAD data from the industry, and have demonstrated strong performance on rigid-body and articulated objects with up to 18 dofs. The typical running times of our planners range from less than a second in simple cases to tens of seconds in very difficult ones.

Unlike most PRM other methods, which are intended for processing many queries in a static environment, our planners do not perform expensive pre-computation and are able to handle environments that are frequently changing. This makes them particularly useful in practical applications. We have given three examples: assembly maintainability checking, motion synthesis for animated characters, and kinodynamic motion planning for an integrated real-time robot system in dynamic environments. In the first two cases, the environments have complex geometry with up to 200,000 triangles, and the final motion of the object requires delicate maneuvers in order to avoid collision with obstacles. To our knowledge, some of these problems are very difficult to solve, if they can be solved at all, with traditional PRM methods. In the third case, our planner for kinodynamic motion planning is integrated with a real-time robot system developed for testing space robotics technology. The robot operates in an environment with moving obstacles. Our planner computes an acceleration-bounded, collision-free trajectory for the robot under real-time constraints. Although the planner assumes that the obstacles move with constant linear velocities, a vision system continuously monitors the motion of the obstacles. If an obstacle deviates from its predicted trajectory, the planner re-computes the robot's trajectory on the fly.

The complexity of path planning problems is traditionally measured by the dimension of the input configuration space as well as the number and the maximum degree of polynomials describing obstacle boundaries. However, experiments in this work and elsewhere indicate that the most significant impact on the running time of randomized path planners comes from narrow passages in the free space. To explain the experimental results, we have introduced the notion of expansive spaces, which is based on the volumes of certain subsets in the free space. We have proven that in an expansive space, our planners find a solution with probability that converges to 1 at an exponential rate, if a solution exists. Also the number of milestones needed to process a query increases only moderately when the space becomes less expansive. The notion of expansive spaces is important, because it provides the analysis tools for us to understand under what conditions randomized motion planning algorithms work well.

An efficient motion planner can also be used as a building block for accomplishing more complex tasks. We have demonstrated this on the robot placement problem in workcell design. Although in this case, a direct invocation of path planners does not solve the problem, the basic idea of constructing the connectivity of the configuration space via random sampling remains useful. By combining our randomized path planner with local iterative optimization, our algorithm computes simultaneously a base location and a corresponding collision-free path that are optimized with respect to the execution time of tasks. The availability of a fast randomized path planner has enabled us to give a more realistic formulation of the robot placement problem, which takes into account both the motion of the robot and obstacles in the environment.

6.2 Future Work

The main outstanding issue with randomized motion planners is the narrow passage problem [Lat00]: if the free space $\mathcal{F}$ contains narrow passages, randomized motion planners, including ours, must sample a large number of milestones in order to have one fall into the narrow passage and establish connections between different parts of $\mathcal{F}$. One heuristic for dealing with this problem is to sample more densely on or near obstacle boundaries [ABD$^+$98, BOvdS99], because narrows passages often lie close to the obstacle boundaries. However, the boundaries may be very complex due to the geometric interaction between the moving object and the obstacles. The effectiveness of this idea has only been demonstrated in relatively simple examples. Another approach is to accept, as milestones, samples whose penetration distance into the obstacles is small, thus effectively widening the narrow passages and making it easier to connect different parts of $\mathcal{F}$. The difficulty with this approach is that in general, penetration distance is difficult to define and compute effectively. A satisfactory solution to the narrow passage problem has so far eluded us; better and maybe more specialized sampling strategies may be needed to address this issue.

The experiments also indicate that the running time distribution of our randomized motion planners has a long and thin tail, caused by a small number of runs taking time much longer than the average. It would be interesting to investigate techniques for reducing the variance of running times. This is especially important for our single-query planners, because they are intended to be used interactively or in real time. Large variances in the running times are highly undesirable in these settings.

To date, most of the work on randomized motion planning has been empirical. We have introduced the notion of expansive spaces as a new way to characterize the difficulty of motion planning problems. However, the parameters for an expansive space cannot be easily determined except for very simple spaces, and so we cannot decide, in advance, the number of milestones needed for a given query. It is important to continue looking for new tools to analyze the efficiency of randomized motion planners. If we cannot measure the performance of these algorithms quantitatively, we will not be able to compare, improve, and thus advance our understanding of them.

Today, more than ever before, computers are being used as a tool to interact with the physical world, either a real one or a simulated version of it. The rapid advancement of hardware technology allows us to represent more and more detailed models of complex physical objects in computers. Generating motion for them becomes an essential task in these environments. Random sampling has made motion planning less dependent on the number of dofs; it has opened up the possibility to study the motion of large systems under complex physical constraints. Many interesting questions are waiting to be explored. How can we synthesize realistic motion for life-like animated characters? How can we control and coordinate the motion of a large group of independent robots? Related to this question is motion planning of reconfigurable robots made up of hundreds or thousands of identical modules. Also, how can we generate motion for deformable objects? We need to develop representations that not only model the physics of deformation correctly, but also are compact enough for efficient motion generation. The complexity of these new problems stems from the interaction between a very large number of dofs and the desire to generate realistic motion under physical constraints. The presence of constraints may reduce the number of dofs; on the other hand, they make the structure of the space more intricate. Random-sampling techniques developed for many-dof articulated objects are a good starting point for answering these questions, but new representations and computational techniques are likely to be needed. As is usually the case with knowledge, “what we know is not much; what we do not know is immense.”[*]

[*] P.-S. Laplace.

Bibliography

[ABD$^+$98] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo. OBPRM: An obstacle-based PRM for 3D workspaces. In P. K. Agarwal et al., editors, *Robotics: The Algorithmic Perspective: 1998 Workshop on the Algorithmic Foundations of Robotics*, pages 155–168, Wellesley, MA, 1998. A. K. Peters.

[ALMR97] P. K. Agarwal, J.-C. Latombe, R. Motwani, and P. Raghavan. Nonholonomic path planning for pushing a disk among obstacles. In *Proc. IEEE International Conference on Robotics and Automation*, pages 3124–3129, 1997.

[ALS95] R. Alami, J.-P. Laumond, and T. Siméon. Two manipulation planning algorithms. In K. Goldberg et al., editors, *Algorithmic Foundations of Robotics (1994 Workshop on the Algorithmic Foundations of Robotics)*, pages 109–125. A. K. Peters, Wellesley, MA, 1995.

[AW96] N. Amato and Y. Wu. A randomized roadmap method for path and manipulation planning. In *Proc. IEEE International Conference on Robotics and Automation*, pages 113–120, 1996.

[BDG85] J. E. Bobrow, S. Dubowsky, and J.S. Gibson. Time-optimal control of robotic manipulators along specified paths. *International Journal of Robotics Research*, 4(3):3–17, 1985.

[BKL$^+$97] J. Barraquand, L. Kavraki, Jean Claude Latombe, Tsai-Yen Li, R. Motwani, and P. Raghavan. A random sampling scheme for path planning. *International Journal of Robotics Research*, 16(6):759–774, 1997.

[BL91] J. Barraquand and J. C. Latombe. Robot motion planning: A distributed representation approach. *International Journal of Robotics Research*, 10(6):628–649, 1991.

[BL93] J. Barraquand and J. C. Latombe. Nonholonomic multibody mobile robots: Controllability and motion planning in the presence of obstacles. *Algorithmica*, 10(2-4):121–155, 1993.

[BLL90] J. Barraquand, B. Langlois, and J. C. Latombe. Robot motion planning with many degrees of freedom and dynamic constraints. In H. Miura et al., editors, *Robotics Research: The Fifth International Symposium*, pages 435–44. MIT Press, Cambridge, MA, 1990.

[BOvdS99] V. Boor, M. H. Overmars, and F. van der Stappen. The gaussian sampling strategy for probabilistic roadmap planners. In *Proc. IEEE International Conference on Robotics and Automation*, pages 1018–1023, 1999.

[Bri89] A. J. Briggs. An efficient algorithm for one-step planar compliant motion planning with uncertainty. In *Proc. ACM Symposium on Computational Geometry*, pages 187–196, 1989.

[Can88a] J. F. Canny. *The Complexity of Robot Motion Planning*. MIT Press, Cambridge, MA, 1988.

[Can88b] J. F. Canny. Some algebraic and geometric computations in PSPACE. In *Proc. IEEE Symposium on Foundations of Computer Science*, pages 460–467, 1988.

[CB97] H. Choset and J. Burdick. Sensor based motion planning: The hierarchical generalized Voronoi graph. In J.-P. Laumond et al., editors, *Algorithms for Robotic Motion and Manipulation: 1996 Workshop on the Algorithmic Foundations of Robotics*. A. K. Peters, Wellesley, MA, 1997.

[CL95] H. Chang and Tsai-Yen Li. Assembly maintainability study with motion planning. In *Proc. IEEE International Conference on Robotics and Automation*, pages 1012–1019, 1995.

[CR87] J. Canny and J. Reif. New lower bound techniques for robot motion planning problems. In *Proc. IEEE Symposium on Foundations of Computer Science*, pages 49–60, 1987.

[CY99] A. Casal and M. Yim. Self-reconfiguration planning for a class of modular robots. In *Proc. SPIE Symposium on Intelligent Systems and Advanced Manufacturing*, volume 3839, pages 246–255, 1999.

[DK90] D. P. Dobkin and D. G. Kirkpatrick. Determining the separation of preprocessed polyhedra—A unified approach. In *Proc. 17th International Colloquium on Automata, Languages, and Programming*, volume 443 of *Lecture Notes Computer Science*, pages 400–413. Springer-Verlag, 1990.

[DK99] J. P. Desai and V. Kumar. Motion planning for cooperating mobile manipulators. *Journal of Robotic Systems*, 16(10):557–579, 1999.

[DXCR93] B. Donald, P. Xavier, J. Canny, and J. Reif. Kinodynamic motion planning. *Jounal of the ACM*, 40(5):1048–1066, 1993.

[ELP87] M. Erdmann and T. Lozano-Pérez. On multiple moving objects. *Algorithmica*, 2(4):477–521, 1987.

[Fed96] J. T. Feddema. Kinematically optimal robot placement for minimum time coordinated motion. In *Proc. IEEE International Conference on Robotics and Automation*, pages 3395–3400, 1996.

[Fer98] P. Ferbach. A method of progressive constraints for nonholonomic motion planning. *IEEE Transactions on Robotics and Automation*, 14(1):172–179, 1998.

[Fra99] T. Fraichard. Trajectory planning in a dynamic workspace: A 'state-time space' approach. *Advanced Robotics*, 13(1):75–94, 1999.

[Fuj95] K. Fujimura. Time-minimum routes in time-dependent networks. *IEEE Transactions on Robotics and Automation*, 11(3):343–351, 1995.

[GBGL$^+$98] H. H. González-Baños, L. J. Guibas, J. C. Latombe, S.M. LaValle, D. Lin, R. Motwani, and C. Tomasi. Motion planning with visibility constraints: Building autonomous observers. In Y. Shirai and S. Hirose, editors, *Robotics Research: The Eighth International Symposium*, pages 95–101. Springer, New York, 1998.

[GHZ99] L. J. Guibas, D. Hsu, and L. Zhang. H-walk: Hierarchical distance computation for moving convex bodies. In *Proc. ACM Symposium on Computational Geometry*, pages 265–273, 1999.

[Gla90] B. Glavina. Solving findpath by combination of goal-directed and randomized search. In *Proc. IEEE International Conference on Robotics and Automation*, pages 1718–1723, 1990.

[GLM96] S. Gottschalk, M. Lin, and D. Manocha. OBB-Tree: A hierarchical structure for rapid interference detection. In *SIGGRAPH 96 Conference Proceedings*, pages 171–180, 1996.

[GO97] J. E. Goodman and J. O'Rourke, editors. *Handbook of Discrete and Computational Geometry*. CRC Press, New York, 1997.

[GRS83] L. J. Guibas, L. Ramshaw, and J. Stolfi. A kinetic framework for computational geometry. In *Proc. IEEE Symposium on Foundations of Computer Science*, pages 100–111, 1983.

[HA00] L. Han and N. M. Amato. A kinematics-based probabilistic roadmap method for closed chain systems. In *Proc. The Fourth International Workshop on the Algorithmic Foundations of Robotics*, 2000.

[HJW84] J. E. Hopcroft, D. A. Joseph, and S. H. Whitesides. Movement problems for 2-dimensional linkages. *SIAM Journal on Computing*, 13(3):610–629, 1984.

[HKL$^+$98] D. Hsu, L. Kavraki, J. C. Latombe, R. Motwani, and S. Sorkin. On finding narrow passages with probabilistic roadmap planners. In P. K. Agarwal et al., editors, *Robotics: The Algorithmic Perspective: 1998 Workshop on the Algorithmic Foundations of Robotics*, pages 141–154. A. K. Peters, Wellesley, MA, 1998.

[HKLR00] D. Hsu, R. Kindel, J. C. Latombe, and S. Rock. Randomized kinodynamic motion planning with moving obstacles. In *Proc. The Fourth International Workshop on the Algorithmic Foundations of Robotics*, 2000.

[HLM97] D. Hsu, J. C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. In *Proc. IEEE International Conference on Robotics and Automation*, pages 2719–2726, 1997.

[HLS88] P. Hsu, Z. Li, and S. Sastry. On grasping and coordinated manipulation by a multifingered robot hand. In *Proc. IEEE International Conference on Robotics and Automation*, pages 384–389, 1988.

[HLS99] D. Hsu, J. C. Latombe, and S. Sorkin. Placing a robot manipulator amid obstacles for optimized execution. In *Proc. IEEE International Symposium on Assembly and Task Planning*, pages 280–285, 1999.

[HSS84] J. E. Hopcroft, J. T. Schwartz, and M. Sharir. On the complexity of motion planning for multiple independent objects; PSPACE-hardness of the “Warehouseman’s Problem”. *International Journal of Robotics Research*, 3(4):76–88, 1984.

[HST94] T. Horsch, F. Schwarz, and H. Tolle. Motion planning for many degrees of freedom — random reflections at C-Space obstacles. In *Proc. IEEE International Conference on Robotics and Automation*, pages 3318–3323, 1994.

[Hub96] P. M. Hubbard. Approximating polyhedra with spheres for time-critical collision detection. *ACM Transactions on Graphics*, 15(3):179–210, 1996.

[HW96] Y. K. Hwang and P. A. Watterberg. Optimizing robot placement for visit-point tasks. In *Proc. AAAI Workshop on Artificial Intelligence for Manufacturing*, pages 81–86, 1996.

[JBN94] M.-R. Jung, N. Badler, and T. Noma. Animated human agents with motion planning capability for 3D-space postural goals. *Journal of Visualization and Computer Animation*, 5(4):225–246, 1994.

[JC89] P. Jacobs and J. Canny. Planning smooth paths for mobile robots. In *Proc. IEEE International Conference on Robotics and Automation*, pages 2–7, 1989.

[Kar91] R. M. Karp. An introduction to randomized algorithms. *Discrete Applied Mathematics*, 34:165–201, 1991.

[Kin00] R. Kindel. *Motion Planning for Free-Flying Robots in Dynamic and Uncertain Environments*. PhD thesis, Dept. of Aeronautics & Astronautics, Stanford University, Stanford, CA, 2000.

[KKKL94] Y. Koga, K. Kondo, J. J. Kuffner, and J. C. Latombe. Planning motions with intentions. In *SIGGRAPH 94 Conference Proceedings*, pages 395–408, 1994.

[KL94a] L. Kavraki and J. C. Latombe. Randomized preprocessing of configurations space for fast path planning. In *Proc. IEEE International Conference on Robotics and Automation*, pages 2138–2139, 1994.

[KL94b] Y. Koga and J. C. Latombe. On multi-arm manipulation planning. In *Proc. IEEE International Conference on Robotics and Automation*, pages 945–952, 1994.

[KL99] J. J. Kuffner and J. C. Latombe. Fast synthetic vision, memory, and learning models for virtual humans. In *Proc. Computer Animation 1999*, pages 118–127, 1999.

[KLH98] L. Kavraki, F. Lamiraux, and C. Holleman. Towards planning for elastic objects. In P. K. Agarwal et al., editors, *Robotics: The Algorithmic Perspective: 1998 Workshop on the Algorithmic Foundations of Robotics*, pages 313–325. A. K. Peters, Wellesley, MA, 1998.

[Kol95] K. Kolarov. Algorithms for optimal design of robots in complex environments. In K. Goldberg et al., editors, *Algorithmic Foundations of Robotics (1994*

Workshop on the Algorithmic Foundations of Robotics), pages 347–369. A. K. Peters, 1995.

[KRVM98] K. Kotay, D. Rus, M. Vona, and C. McGray. The self-reconfiguring robotic molecule: Design and control algorithms. In P. K. Agarwal et al., editors, *Robotics: The Algorithmic Perspective: 1998 Workshop on the Algorithmic Foundations of Robotics*, pages 375–386. A. K. Peters, Wellesley, MA, 1998.

[KW86] M. H. Kalos and Paula A. Whitlock. *Monte Carlo Methods*, volume 1. John Wiley & Son, New York, 1986.

[KZ86] K. Kant and S. W. Zucker. Toward efficient trajectory planning: The path-velocity decomposition. *International Journal of Robotics Research*, 5(3):72–89, 1986.

[L$^+$85] E. L. Lawler et al., editors. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. John Wiley & Son, 1985.

[Lat91a] J. C. Latombe. A fast path planner for a car-like indoor mobile robot. In *Proc. 9th National Conference on Artificial Intelligence*, pages 659–665, 1991.

[Lat91b] J. C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, MA, 1991.

[Lat00] J. C. Latombe. Motion planning: A journey of robots, molecules, digital actors, and other artifacts. *International Journal of Robotics Research*, 18(11):1119–1128, 2000.

[Lau86] J.-P. Laumond. Feasible trajectories for mobile robots with kinematic and environmental constraints. In *Proc. Int. Conf. on Intelligent Autonomous Systems*, pages 346–354, 1986.

[LC91] M. C. Lin and J. F. Canny. A fast algorithm for incremental distance calculation. In *Proc. IEEE International Conference on Robotics and Automation*, pages 1008–1014, 1991.

[LCH89] Z. Li, J. F. Canny, and G. Heinzinger. Robot motion planning with nonholonomic constraints. In H. Miura et al., editors, *Robotics Research: The Fifth International Symposium*, pages 309–316. MIT Press, Cambridge, MA, 1989.

[LJTM94] J-P. Laumond, P. E. Jacobs., M. Taïx., and R. M. Murray. A motion planner for nonholonomic mobile robots. *IEEE Transactions on Robotics and Automation*, 10(5):577–593, 1994.

[LK99] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. In *Proc. IEEE International Conference on Robotics and Automation*, pages 473–479, 1999.

[LL96] F. Lamiraux and J. P. Laumond. On the expected complexity of random path planning. In *Proc. IEEE International Conference on Robotics and Automation*, pages 3014–3019, 1996.

[LM96] K. M. Lynch and M. T. Mason. Stable pushing: Mechanics, controllability, and planning. *International Journal of Robotics Research*, 15(6):533–556, 1996.

[LM97] A. D. Lewis and R. M. Murray. Configuration controllability of simple mechanical control systems. *SIAM Journal on Control and Optimization*, 35(3):766–790, 1997.

[LP83] T. Lozano-Pérez. Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, C-32(2):108–120, 1983.

[LPMT84] T. Lozano-Pérez, M. T. Mason, and R. H. Taylor. Automatic synthesis of fine-motion strategies for robots. *International Journal of Robotics Research*, 3(1):3–24, 1984.

[LPW79] T. Lozano-Pérez and M. A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, 1979.

[LRDG90] J. Lengyel, M. Reichert, B. R. Donald, and D. P. Greenbert. Real-time robot motion planning using rasterizing computer graphics hardware. In *SIGGRAPH 90 Conference Proceedings*, pages 327–335, 1990.

[LS87] V. J. Lumelsky and A. A. Stepanov. Path planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica*, 2(4):403–430, 1987.

[LS89] J.-P. Laumond and T. Siméon. Motion planning for a two degrees of freedom mobile robot with towing. Technical Report LAAS/CNRS No. 89-148, LAAS, Toulouse, France, 1989.

[LS95] S. M. LaValle and R. Sharma. A framework for motion planning in stochastic environments: Applications and computational issues. In *Proc. IEEE International Conference on Robotics and Automation*, pages 3063–3068, 1995.

[Lyn99] K. Lynch. Controllability of a planar body with unilateral thrusters. *IEEE Transactions on Automatic Control*, 44(6):1206–1211, 1999.

[MC92] B. Mirtich and J. Canny. Using skeletons for nonholonomic path planning among obstacles. In *Proc. IEEE International Conference on Robotics and Automation*, pages 2533–2540, 1992.

[Mir98] B. Mirtich. *V-Clip*: Fast and robust polyhedral collision detection. *ACM Transactions on Graphics*, 17(3):177–208, 1998.

[MS90] R. M. Murray and S. S. Sastry. Steering nonholonomic systems using sinusoids. In *Proc. IEEE Conference on Decision and Control*, volume 4, pages 2097–3101, 1990.

[NGY00] A. Nguyen, L. J. Guibas, and M. Yim. controlled module density helps reconfiguration planning. In *Proc. The Fourth International Workshop on the Algorithmic Foundations of Robotics*, 2000.

[NSL99] C. Nissoux, T. Siméon, and J.-P. Laumond. Visibility based probabilistic roadmaps. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1316–1321, 1999.

[OLP89] P. A. O'Donnell and T. Lozano-Pérez. Deadlock-free and collision-free coordination of two robot manipulators. In *Proc. IEEE International Conference on Robotics and Automation*, pages 484–489, 1989.

[Oue97] F. B. Ouezdou. General method for kinematic synthesis of manipulators with task specifications. *Robotica*, 15(6):653–661, 1997.

[Ove92] M. H. Overmars. A random approach to motion planning. Technical Report RUU-CS-92-32, Dept. of Computer Science, Utrecht University, Utrecht, The Netherlands, 1992.

[PEUC97] A. Pamecha, I. Ebert-Uphoff, and G. S. Chirikjian. Useful metrics for modular robot motion planning. *IEEE Transactions on Robotics and Automation*, 13(4):531–545, 1997.

[PTVP92] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Plannery. *Numerical Recipes in C.* Cambridge University Press, second edition, 1992.

[QK93] S. Quinlan and O. Khatib. Elastic bands: Connecting path planning and control. In *Proc. IEEE International Conference on Robotics and Automation*, pages 802–807, 1993.

[Qui94] S. Quinlan. Efficient distance computation between non-convex objects. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pages 3324–3329, 1994.

[Rei79] J. H. Reif. Complexity of the mover's problem and generalizations. In *Proc. IEEE Symposium on Foundations of Computer Science*, pages 421–427, 1979.

[Rim97] E. Rimon. Construction of C-space roadmaps from local sensory data. What should the sensors look for? *Algorithmica*, 17(4):357–379, 1997.

[RS85] J. Reif and M. Sharir. Motion planning in the presence of moving obstacles. In *Proc. IEEE Symposium on Foundations of Computer Science,* pages 144–154, 1985.

[RS90] J. A. Reeds and L. A. Shepp. Optimal paths for a car that goes forwards and backwards. *Pacific Journal of Mathematics*, 145(2):367–393, 1990.

[SD91] Z. Shiller and S. Dubowsky. On computing the global time-optimal motions of robotic manipulators in the presence of obstacles. *IEEE Transactions on Robotics and Automation*, 7(6):785–797, 1991.

[SDS96] E. J. Stollnitz, T. D. DeRose, and D. H. Salesin. *Wavelets for Computer Graphics*. Morgan Kaufmann, San Francisco, CA, 1996.

[Ser95] H. Seraji. Reachability analysis for base placement in mobile manipulators. *Journal of Robotic Systems*, 12(1):29–43, 1995.

[Sho85] K. Shoemake. Animating rotation with quaternion curves. In *SIGGRAPH 85 Conference Proceedings*, pages 245–254, 1985.

[SL98] S. Sekhavat and J.-P. Laumond. Topological property for collision-free nonholonomic motion planning: The case of sinusoidal inputs for chained form systems. *IEEE Transactions on Robotics and Automation*, 14(5):671–680, 1998.

[SLB99] A. P. Singh, J. C. Latombe, and D. L. Brutlag. A motion planning approach to flexible ligand binding. In *Proc. Intelligent Systems for Molecular Biology*, pages 252–261, 1999.

[ŠO95a] P. Švestka and M. Overmars. A probabilistic learning approach to motion planning. In K. Goldberg et al., editors, *Algorithmic Foundations of Robotics (1994 Workshop on the Algorithmic Foundations of Robotics)*, pages 19–37. A. K. Peters, Wellesley Massachusetts, 1995.

[ŠO95b] P. Švestka and M. H. Overmars. Coordinated motion planning for multiple car-like robots using probabilistic roadmaps. In *Proc. IEEE International Conference on Robotics and Automation*, pages 1631–1636, 1995.

[SS83a] J. T. Schwartz and M. Sharir. On the ‘piano movers’ problem: II. general techniques for computing topological properties of real algebraic manifolds. *Advances in Applied Mathematics*, 4:298–351, 1983.

[SS83b] J. T. Schwartz and M. Sharir. On the piano movers’ problem: III. coordinating the motion of several independent bodies: The special case of circular bodies moving amidst polygonal barriers. *International Journal of Robotics Research*, 2(3):46–75, 1983.

[SŠLO97] S. Sekhavat, P. Švestka, J.-P. Laumond, and M. Overmars. Multi-level path planning for nonholonomic robots using semi-holonomic subsystms. In J.-P. Laumond et al., editors, *Algorithms for Robotic Motion and Manipulation: 1996 Workshop on the Algorithmic Foundations of Robotics*, pages 79–96. A. K. Peters, Wellesley, MA, 1997.

[Udu77] S. Udupa. *Collision Detection and Avoidance in Computer Controlled Manipulators*. PhD thesis, Dept. of Electrical Engineering, California Institute of Technology, Pasadena, CA, 1977.

[WAS99] S. A. Wilmarth, N. M. Amato, and P. F. Stiller. MAPRM: A probabilistic roadmap planner with sampling on the medial axis of the free space. In *Proc. IEEE International Conference on Robotics and Automation*, pages 1024–1031, 1999.