RANDOM NETWORKS IN CONFIGURATION SPACE FOR FAST PATH PLANNING

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

By
Lydia E. Kavraki
December 1994

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Jean-Claude Latombe
(Principal Adviser)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Leonidas J. Guibas

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Oussama Khatib

Approved for the University Committee on Graduate Studies:

Dean of Graduate Studies

Abstract

In the main part of this dissertation we present a new path planning method which computes collision-free paths for robots of virtually any type moving among stationary obstacles. This method proceeds according to two phases: a preprocessing phase and a query phase. In the preprocessing phase, a probabilistic network is constructed and stored as a graph whose nodes correspond to collision-free configurations and edges to feasible paths between these configurations. These paths are computed using a simple and fast local planner. In the query phase, any given start and goal configurations of the robot are connected to two nodes of the network; the network is then searched for a path joining these two nodes. The method is general and easy to implement. It can be applied to virtually any type of holonomic robot. It requires selecting certain parameters (e.g., the duration of the preprocessing phase) whose values depend on the considered scenes, that is the robots and their workspaces. These values turn out to be relatively easy to choose. Increased efficiency of the method can also be achieved by tailoring some of its components (the local planner, for example) to the considered robots. In this thesis the method is applied to articulated robots with many degrees of freedom. Experimental results show that path planning can be done in a fraction of a second on a contemporary workstation ($\approx$ 150 MIPS), after relatively short preprocessing times (a few dozen to a few hundred seconds for the most difficult examples we have treated).

In the second part of this dissertation, we present a new method for computing the obstacle map used in motion planning algorithms. The method, which is practical only for two-dimensional workspaces, computes a convolution of the workspace

and the robot with the use of the Fast Fourier Transform (FFT). It is particularly promising for workspaces with many and/or complicated obstacles, or when the shape of the robot is not simple. Furthermore, it is an inherently parallel method that can significantly benefit from existing experience and hardware on the FFT.

In the third part, we consider a problem from assembly planning. In assembly planning we are interested in generating feasible sequences of motions that construct a mechanical product or take it apart to its individual parts. The problem addressed is the following: given a planar assembly of non-overlapping polygons, decide if there is a proper subcollection of them that can be moved in the plane as a rigid body, and separated from the rest of the assembly without colliding with or disturbing the other parts of the assembly. We prove that this problem is NP-complete when the polygons of the assembly can translate, and when they can both translate and rotate. Several other variants of the above partitioning problem are shown to be NP-complete.

Acknowledgements

I am deeply grateful to my advisor, Professor Jean-Claude Latombe, for his strong guidance and encouragement during my studies. He generously helped me start on my research and always directed my explorations with patience and insight. He has been accessible and very committed to his work, and has created a scholarly but relaxed environment in the Robotics Laboratory at Stanford.

The other members of my reading committee, Professor Leonidas Guibas and Professor Oussama Khatib, helped me greatly with their comments and helpful discussions, throughout my studies here. Professor Anoop Gupta has also been very supportive since my early years at Stanford.

The work in this dissertation has been funded by DARPA contract DAAA21-89-C002, ARPA grant N0014-92-J-1809, ONR grant N0014-94-1-0721, a fellowship from GTE, and a fellowship from Rockwell Foundation.

I consider myself lucky to have had the opportunity to work among exceptional people at the Robotics Lab. They have made my years at Stanford an exciting period in my life. Many thanks to Craig Becker, Pierre Ferbach (who also proof-read part of my thesis), Danny Halperin, Yotto Koga, Anthony Lazanas, James Kuffner, Tsai-Yen Li, Mark Yim, David Zhu, and to the people with whom I worked more closely: Randy Wilson, Achim Schweikard, and Rhea Tombropoulos. Jutta McCormick provided wonderful support to the Lab. Carolyn Tajnai, the Director of the Computer Forum at Stanford, was most helpful to me at the end of my studies. In the Computer Science Department I found support and understanding in my colleagues Marcia Derr and Liz Wolf and I enjoyed my conversations with Adnan Darwiche and Aaron Goldberg.

I also had the chance to collaborate with Petr Švestka and Professor Mark Overmars of Utrecht University and this has been a very interesting experience for me.

I have met many people at Stanford who influenced my life. Among them, Sarah Jones has been an invaluable friend. Stefanos Sidiropoulos, Lena Tsakmaki, Anthony Lazanas, Vasso Roti, Yiannis Petridis, and George Hadjidakis have made the time spent at Stanford very enjoyable.

As an undergraduate, I attended the University of Crete, a small but vibrant school. During this most formative period in my life, I had excellent teachers who provided me with a solid background in Computer Science and encouraged my early involvement in research. I am indebted to Professors Yannis Vassiliou, Panos Constantopoulos, Manolis Katevenis, Stelios Orfanoudakis and Costas Courcoubetis.

I would not have been able to make it so far without the help of my parents, Mary and Vangelis Kavrakis. My father's insistence (and help) that I learn how things work, in nature and in technology, along with my mother's diverse interests and active involvement, have been my most decisive education. I am fortunate to have another role model in my life, Soula Karafoulidou. She has taught me to look at things from the right perspective, and helped me pursue them, baring their essence for me. I am also fortunate to have Eliza as my sister. Eliza has affected me deeply with her personality and interests. I have missed her considerably during my long studies, as I have missed my grandmother Zafiroula Dritsa, who cared for me for many years, and Ninon Karafoulidou who influenced my life so much when I was younger.

Last but not least, my husband Mihalis Kolountzakis has been a constant source of inspiration for me. His quest for expanding his horizons together with his balanced and consistent attitude in life, have given a unique meaning to our relationship. This work would never have been possible without his love and support.

Αφιερωνεται

στους γονεις μου Μαιρη και Βαγγελη

στη Σουλα

στην Ελιζα

και στο Μιχαλη.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 On Robotics and Planning

Automation plays an ever more important role in our society. From the machine that dispenses refreshments to a mechanical arm that paints an automobile along the assembly line, tasks that are considered tedious, dangerous or physically demanding are delegated to machinery. Thus, on the one hand, humans are free to deal with work that requires intelligence (or free not to work at all) and, on the other, these tasks are performed much more efficiently and reliably by machines than by humans.

The need to have machines take over an ever larger part of our work creates the need to have *intelligent* machines. It takes the combination of mechanisms, sensors, dynamics, control and reasoning about the physical world to arrive to systems that can synthesize some aspects of human function.

What we usually call a *robot* today, consists of two parts: a) a mechanical device which, properly controlled, can perform a useful task (e.g. open the door of the can dispenser and let one can out), and b) a computer which controls the mechanical part and interacts with the outside world (e.g. senses that the amount of coins dropped in the can dispenser is sufficient and orders the door to open).

A large class of robots have the ability to move in their surrounding space. The robot can move as a whole (e.g. the robots that carry meals to the patients of Stanford Hospital), or may rest on a fixed base and move parts of it (e.g. a painter arm along the assembly line). In either case it is necessary for the robot to be able to avoid hitting the obstacles that surround it while it is performing some useful task.

A principal problem in robotics, which will concern us in the main part of this dissertation, is that of *motion planning*. We want to devise algorithms that will enable a robot to move from one position to another without any collisions. Indeed, this capability is most important in applications. The robot of Stanford Hospital may have a map of the building and know at any given moment its exact position (or *configuration*), but when asked to go to a certain point in the building it will need to find a collision-free path from its current position to its target.

Since the robot interacts with its environment it needs to have some knowledge about the environment. This can either be given to the robot off-line as a map or can be obtained by the robot itself using certain sensors like cameras, sonars and lasers. For the purposes of this thesis we will assume that the knowledge about the surrounding space, the *workspace*, is complete and will not discuss how it was acquired.

A fundamental task that is performed invariably by all motion planners is the task of answering questions of the form "does this position of the robot give rise to a collision?" This seems conceptually a very easy problem, but its importance is such that any gain in the speed of processing such queries is reflected almost intact in the speed of the motion planner. Indeed, most planners ask this question a great many times. And many planners treat collision checking as a black box subroutine. This separation of the planning task from the collision checking task creates the need for an efficient, general purpose collision checker, that can be used by any planner.

Finally, we deal with a problem from *assembly planning*, an important generalization of the motion planning problem. In assembly planning we are interested to plan the simultaneous motion of a collection of objects, from an initial configuration in space to a final one. (Incidentally, assembly planning can be put in the framework of robot motion planning if we consider the system of objects as our robot, whose set of degrees of freedom consists of the degrees of freedom of the individual objects.) The importance of being able to perform this task is enormous for automatic assembly, maintenance and repair of mechanical parts, as well as for the fast manufacturing evaluation of assembly designs. The problem has been shown to be computationally hard, as have many special cases of it, which one might initially expect to be easier to handle. We discuss such a special case here, that of *assembly partitioning* into *two* subassemblies.

1.2 Contributions of this Dissertation

There are three distinct parts in this dissertation.

Motion Planning

In the main part of this dissertation we present a new path planning method which computes collision-free paths for robots of virtually any type moving among stationary obstacles (static workspaces). This method proceeds according to two phases: a preprocessing phase and a query phase. In the preprocessing phase, a probabilistic network is constructed and stored as a graph whose nodes correspond to collision-free configurations and edges to feasible paths between these configurations. These paths are computed using a simple and fast local planner. In the query phase, any given start and goal configurations of the robot are connected to two nodes of the network; the network is then searched for a path joining these two nodes. The method is general and easy to implement. It can be applied to virtually any type of holonomic robot. It requires selecting certain parameters (e.g., the duration of the preprocessing phase) whose values depend on the considered scenes, that is the robots and their workspaces. But these values turn out to be relatively easy to choose.

Increased efficiency of the method can also be achieved by tailoring some of its components (e.g., the local planner) to the considered robots. In this thesis the method is applied to articulated robots with many degrees of freedom. Experimental results show that path planning can be done in a fraction of a second on a contemporary workstation ($\approx$ 150 MIPS), after relatively short preprocessing times. These times range from a few dozen seconds to a few hundred seconds for the most difficult examples we have treated.

We also attempt a theoretical analysis of an abstracted model of our algorithm. Our results give us a good idea of how the performance of the planning method depends on certain characteristics of the space in which the probabilistic network is constructed. Unfortunately these characteristics are hard to measure or estimate a priori.

Collision Checking

In the second part of this dissertation, we present a new method for computing the obstacle map used in motion planning algorithms including the one described above. The method, which is practical only for calculating low-dimensional maps, computes a convolution of the workspace and the robot with the use of the Fast Fourier Transform (FFT) algorithm. It is particularly promising for workspaces with many and/or complicated obstacles, or when the shape of the robot is not simple. It is an inherently parallel method that can significantly benefit from existing experience and hardware on the FFT.

Assembly Partitioning

In the last part of this dissertation, we consider a problem from assembly planning. In assembly planning we are interested in generating feasible sequences of motions that construct a mechanical product or take it apart to its individual parts. The problem addressed is the following: given a planar assembly of polygons, decide if there is a subassembly that can be moved in the plane as a rigid body, and separated from the rest of the assembly without disturbing the other parts of the assembly.

We prove that this problem is NP-complete when the polygons of the assembly can translate, and when they can both translate and rotate. The same complexity result is also shown for a planar assembly consisting of parts whose vertices are constrained to lie on a grid, whose angles are right, and which are allowed to translate in the horizontal and vertical directions only by grid increments. We finally discuss some variants of the problem that are also NP-complete. These include: assemblies with parts of constant complexity, assemblies that can be completely separated to their parts by repeated partitioning and assemblies of polyhedra.

The work described in the first part of this dissertation has appeared in [Kavraki and Latombe, 93b, Kavraki and Latombe, 94a, Kavraki and Latombe, 94b] and in collaboration with Petr Švestka and Mark Overmars in [Kavraki et al, 94]. The work in the second part of this dissertation is also described in [Kavraki, 93]. The work in the third part has appeared in [Kavraki and Latombe, 93a] and in collaboration with Randy Wilson in [Kavraki et al, 93, Wilson et al 93].

1.3 Outline

The organization of this dissertation is as follows:

- Chapters 2, 3, and 4 are devoted to the presentation and evaluation of the path planning algorithm.

  In Chapter 2 we define the motion planning problem using the configuration space formalization. We also discuss the complexity of the problem as well as previously developed motion planners that work well in practice. The rest of the chapter describes our path planning algorithm. The presentation is general and appropriate for virtually any holonomic robot. The parameters of the method are clearly identified and summarized at the end of the chapter.

In Chapter 3 our method is applied to articulated robots moving in two and three dimensional workspaces. The robots used have up to 16 degrees of freedom. We describe specific techniques which can be substituted for more general ones in the method to handle articulated robots more efficiently (especially when these have many degrees of freedom). Extensive experiments help evaluate the performance of the method and explain how the choice of the parameters should be done.

In Chapter 4 a theoretical analysis of a simplified version of our algorithm is carried out. The analysis sheds some light on how the performance of the method depends on certain features of the space in which the robot moves.

- Chapter 5 describes an algorithm for computing the obstacle map used in many planning algorithms. Some background is given on different algorithms that are available for computing the above-mentioned map. Our approach uses the FFT algorithm to compute the convolution of the workspace and the robot. We briefly compare the FFT-based approach with other existing techniques and note its advantages and disadvantages. In the path planning method described in this dissertation, obstacle maps arising from two-dimensional workspaces were calculated with the FFT-based algorithm.

- In Chapter 6 we present some lower bounds on the complexity of the planar assembly partitioning problem. This problem is not directly related to motion planning. It is an important problem from assembly sequencing and this line of research aims at identifying restrictions that should be placed on assemblies so that they can be handled efficiently by automated robotic systems. We define the problem, discuss some previous work and proceed to reduce a well known NP-complete problem (3-Satisfiability) to it. Several variants of planar partitioning are also shown NP-complete.

- We conclude in Chapter 7 by stressing the contributions and the limitations of the work described in this dissertation.

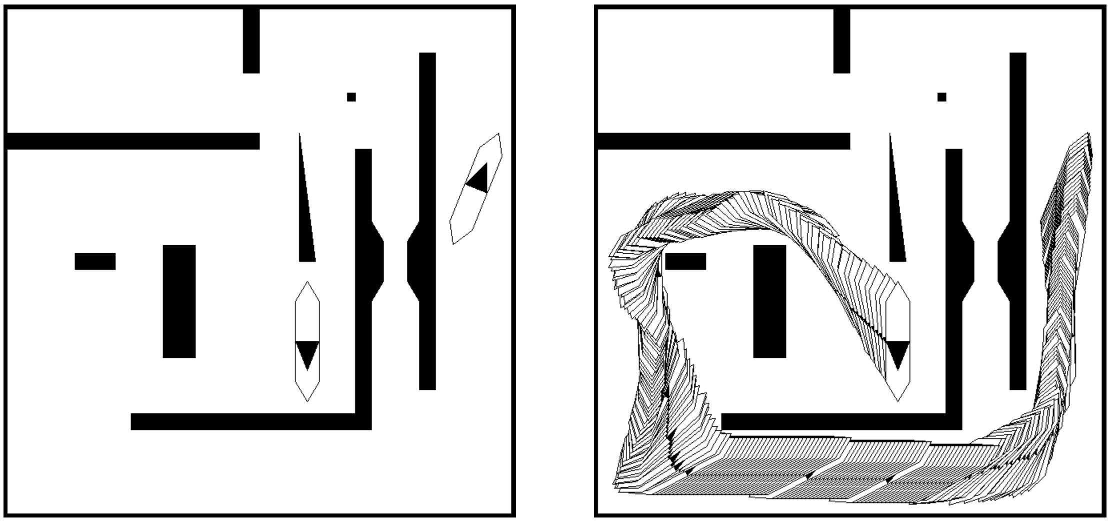
Chapter 2

Random Networks for Path Planning

2.1 Collision-Free Path Planning

In its simplest form path planning can be defined as follows: Let $\mathcal{A}$ be a robot which is either a single rigid body or a collection of rigid subparts (some of which may be attached to each other at certain joints while others may move independently) and suppose that $\mathcal{A}$ can move in a two or three dimensional workspace $\mathcal{W}$. The workspace is populated by obstacles which are known to the robot. Given an initial position I and a final position F of the robot, *path planning* consists of determining whether there exists a continuous collision-free motion that takes $\mathcal{A}$ from I to F, and if so, to find such a motion. Figure 2.1(b) shows a path for a planar robot. The initial and final positions of the robot are drawn in Figure 2.1(a).

An alternative formulation of the problem is obtained through the abstraction of the *configuration space* (C-space) of the robot [Lozano-Pérez, 81]. The C-space "encodes" the set of all possible placements (configurations) of the robot. Each point in C-space is a d-tuple with values for the parameters that correspond to the d degrees of freedom (dof) of the robot. Specification of the values of the dof of $\mathcal{A}$ completely



(a) (b)

Figure 2.1: In (b) we see a collision-free path that joins the two configurations of the planar robot shown in (a)

determines the position of $\mathcal{A}$ in its workspace.

For example, assume that $\mathcal{A}$ is a rigid body (of any shape) that is free to translate and rotate in the plane. Fix two points A and B on the robot as shown in Figure 2.2(a). In order to describe an arbitrary configuration of the robot we need to specify the coordinates (with respect to a fixed coordinate system) of the point A, which we call x, y and θ which is the angle that vector AB forms with the x-axis. So in this case the C-space is the set of all possible triples (x, y, θ), with x, y being real numbers and $\theta \in [0, 2\pi)$. Suppose now that $\mathcal{A}$ can move freely in space. To specify an arbitrary position of the robot, we have to give the coordinates (x, y, z) of the point A, describe the orientation of the vector AB and any possible rotation about the AB axis. The orientation of a vector in space requires two angles: the angle θ that the projection of AB onto the xy-plane forms with the x-axis, as well as the angle ϕ formed by the vector AB and its above-mentioned projection. Additionally, each rotation about the AB axis requires a single angle. Thus the dimension of the C-space is 6. As a final example of C-space, consider a planar articulated robot consisting of 7 links serially connected by revolute joints (Figure 2.2(b)). Any configuration of that robot can be specified if we give the values of the angle that each link forms with the previous one,

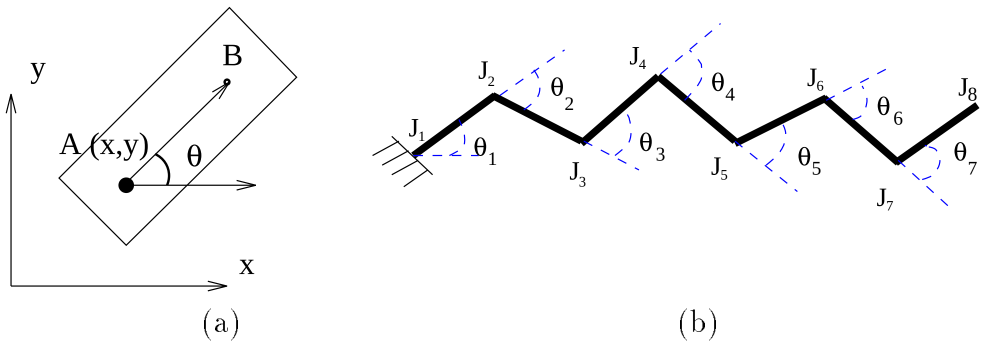


Figure 2.2: (a) A rigid body moving freely in the plane and (b) a planar articulated robot with fixed length links

except from the first link for which we give the angle it forms with the global x-axis. In this case the C-space consists of all 7-tuples $(\theta_1, \ldots, \theta_7)$ with $\theta_1, \ldots, \theta_7 \in [0, 2\pi)$ and is thus 7-dimensional.

For any specific robot, a large part of the C-space is not feasible. For example, the planar robot of Figure 2.2(a) may move in a cluttered workspace and a large part of the C-space will become infeasible because of collisions of the robot with the surrounding obstacles. The robot itself may impose constraints on the feasible configurations. The articulated robot of Figure 2.2(b) will collide with itself for most assignments of values of the 7-tuple $(\theta_1, \ldots, \theta_7)$. There may also be mechanical stops in the joints of the robot and these further restrict the free C-space.

The points of the C-space can be classified into three categories: those which give rise to *autocollision* or are due to the mechanical design of the device (e.g., mechanical stops), those at which the robot collides with the surrounding obstacles, and the rest of the points which we call *free* or *feasible*. For all practical purposes the kind of collision (autocollision, mechanical design restrictions, or collision with the surrounding obstacles) is immaterial when planning the motion of a robot. So we group the points of the first two categories in one set (the forbidden set) which we usually call the *C-obstacle*. A detailed discussion on C-space and its formal representations can be found in [Latombe, 91a].

Hereafter, we denote the C-space by $\mathcal{C}$. The C-obstacle, $\mathcal{O}$ can be regarded as an arbitrary subset of $\mathcal{C}$ of which we only demand that there exists an effective procedure that answers the question

"Is the point $P \in \mathcal{O}$?"

The free part of the C-space $\mathcal{F}$ is the complement of the C-obstacle, that is $\mathcal{F} = \mathcal{C} \backslash \mathcal{O}$.

In the above setting the robot $\mathcal{A}$ can now be identified with a point moving freely in $\mathcal{F}$, the (high-dimensional) free part of the C-space. The path-planning task now becomes the task of finding a continuous path in $\mathcal{F}$ which joins two given points I (the initial configuration) and F (the final configuration) without intersecting the C-obstacle (the points I and F are of course assumed to be in $\mathcal{F}$), or answer that such path does not exist.

In the rest of this chapter we will discuss the complexity of the path-planning problem and present existing algorithms for solving it. Then we will describe a method for planning paths for robots with many degrees of freedom. The difficulty of path planning rises sharply as the number of dof goes up (see Section 2.3). It is important to have algorithms that can be used effectively in the increasing number of practical problems which involve many-dof robots. The work presented in this dissertation is directed towards this end.

2.2 Other Aspects of Path Planning

In practice we are not only interested in obtaining a path of the robot from the initial to the final position [Latombe, 91a]. Because of limitations on the ability to control a mechanical device, the path that we give as a solution to a path-planning problem should obey several restrictions. For example, it should be smooth since following an irregular path would mean the exertion of strong forces on the mechanical parts of the robot. Also, the length of the produced path may be an important consideration.

It will be apparent from the next section of this chapter that the mere finding of *any* path joining I and F is already very difficult when the dimension of the C-space is high (6 or more is considered high), or when the workspace (and consequently the C-space) is very cluttered with obstacles. Let us mention here that typically the C-obstacle takes up almost all of the C-space in high-dimensional spaces. In the examples we consider later in this dissertation the free part of the C-space is approximately 0.1% of the C-space.

We adopt the point of view in this thesis, that for many-dof robots most of the effort should be spent on constructing any path from I to F. This path will then be subjected to a (heuristic) "smoothing" operation that will make it conform more to the requirements mentioned above. Looking ahead, let us also mention that the path-planning method that we shall propose inherently produces paths that are quite satisfactory in terms of smoothness.

2.3 The Complexity of Path Planning

In this section we present some of the most important results that concern the complexity of the path-planning problem. The upper and lower bounds we give are a function of the *size of the problem.* What exactly we consider as the size of the problem is usually irrelevant if we only care for membership of the problem in classes such as P, NP, NP-hard, etc. In these cases the size should be thought of as the size of the most economical description of a problem instance, or any polynomial function of that. For example, if we are planning the motion of a polygonal robot in a polygonal workspace, then it is reasonable to take as the size of the problem the total number of vertices of the robot and all the obstacles. In cases where the bounds (lower or upper) are polynomial, the size of the problem is usually considered to be the number of vertices of the polygons involved, the degree of the polynomials involved in describing subsets of the space, etc, and is explicitly defined in each case. For the complexity classes mentioned see for example [Garey and Johnson, 79].

2.3.1 Lower Bounds

We highlight some of the most important results which describe lower bounds on the complexity of several problems. Most of them are hardness results proved via some efficient reduction from known hard problems.

Polyhedral World

Planning the motion of a robot which consists of a set of polyhedra in the three-dimensional space connected together at some joint vertices in a workspace of polyhedral obstacles is PSPACE-hard [Reif, 79].

Multiple Rectangular Robots in a 2D Box

We have multiple independent robots each of which is a rectangle aligned with the axes of the plane and can only move along the x or y direction and only within an aligned rectangular workspace which is otherwise empty of obstacles. It is then PSPACE-complete to decide whether two different configurations can be joined by a collision-free path [Hopcroft et al, 84, Hopcroft and Wilfong, 86].

Planar Arm Among Polygonal Obstacles

The robot is a planar arm with arbitrarily many non-extensible links serially connected by revolute joints. The obstacles are arbitrary polygons in the plane. Path planning is PSPACE-hard [Joseph and Plantiga, 85].

Moving Obstacles and Translating Robot in 3-Space

The robot is an arbitrary polyhedron free to translate in 3-space and the obstacles are arbitrary polyhedra which can both translate and rotate along known trajectories. If the speed of the robot is bounded by a constant then the problem of avoiding the obstacles while the robot is moving to its target position is PSPACE-hard [Reif and Sharir, 85].

Line Segment Moving Freely Among Polygons

The robot is a freely moving line segment (zero thickness) and is surrounded by polygonal obstacles with total number of vertices equal to n. Then it is known [O'Rourke, 85] that there exists a workspace and two placements of the robot which are reachable from each other, but are such that any motion from one to the other has "complexity" at least $\Omega(n^2)$. That is the robot has to alternate moving forwards and backwards at least as many times during its motion. As a result, any algorithm that finds such a motion needs $\Omega(n^2)$ time.

Shortest Paths in Space Among Polyhedra

It is an NP-hard problem [Canny and Reif, 87] to find the shortest path between two given points in space that avoids a collection of polyhedral obstacles. In the plane this problem admits a polynomial time solution (see below).

2.3.2 Upper Bounds

We present the complexity of some known algorithms for solving various kinds of path-planning problems.

Point in High Dimension

Planning the motion of a point in dimension d when the obstacles are described by a set of n polynomial constraints of maximum degree m can be done in time doubly exponential in the dimension d and polynomial in both n and m [Schwartz and Sharir, 83b]. The method is based on the Collins decomposition algorithm. Canny [Canny, 88] reduced this upper bound to simply exponential in d. Both methods reduce path planning to the satisfiability of a sentence of the first order theory of the real numbers.

Line Segment Moving Freely Among Polygons

There is an algorithm [Leven and Sharir, 87a] for the computation of the path of a line segment among polygonal obstacles of total number of vertices equal to n which takes time $O(n^2 \log n)$.

Disk Moving Among Polygons

Planning for a disk that moves freely among polygons can be done in time $O(n \log n)$ [Ó'Dúnlaing and Yap, 82] where n is the total number of vertices of the obstacles.

Convex Polygon Translating Among Polygons

When the robot is a convex polygon of a sufficiently simple shape translating among polygons with total number of vertices equal to n then planning can be done in time $O(n \log n)$ [Leven and Sharir, 87b].

Polygon Moving Freely Among Polygons

If the robot is a polygon (in the plane) with a constant number of sides which is free to translate and rotate among polygonal obstacles with total number of sides equal to n, then path planning can be done in $O(n^{2+\epsilon})$ time, for any fixed $\epsilon > 0$ [Halperin and Sharir, 93].

Shortest Paths in the Plane Among Polygons

If the obstacles are polygons in the plane with total number of vertices equal to n, then the shortest free path between two placements of a point robot can be found [Hershberger and Suri, 93] in time $O(n \log n)$.

2.4 Practical Approaches to Path Planning for Many Degrees of Freedom

The algorithms of the previous section assume that the environment is available in a clean analytic representation. They return a path, or declare that there is no feasible path, in time that is known a priori.

The complexity of these path-planning methods in high-dimensional configuration spaces has led researchers to seek methods that embed weaker notions of completeness (e.g., probabilistic completeness) and/or can be partially adapted to specific problem domains in order to boost performance in those domains. The main characteristic of such methods is that they are quite successful for non-pathological problems, and can be expected to work well in "practice", but they may take inappropriately long time in some cases. We refer to them as practical approaches to path planning.

Considerable attention is directed towards the creation of practical planners for many-dof robots. Indeed, while such robots are becoming increasingly useful in industrial applications, there are very few planners that can efficiently plan their motions. New emerging applications also motivate that trend, e.g., computer graphics animation, where motion planning can drastically reduce the amount of data input by human animators, and molecular biology, where motion planning can be used to compute motions of molecules (modeled as spatial linkages with many dof) docking against other molecules.

In recent years, some of the most impressive results in planning for robots with many degrees of freedom were obtained using potential-field planning methods. Such methods are attractive, since the main heuristic function they use to guide the search for a path, the potential field, can easily be adapted to the specific problem to be solved, in particular the workspace and the final configuration. Two main lines of research are given below:

- A method which uses the idea of "dynamic" potential fields is proposed in [Faverjon and Tournassoud, 87] for planning the paths of robots with many dof. The potential function depends not only on the distance between the robot and the obstacles, but also on the rate of variation of this distance along the current direction of motion of the robot. The method can be very fast on rather simple examples, but it may get stuck at local minima of the potential function on more difficult ones. It was used to compute paths of an 8-dof manipulator among vertical pipes in a nuclear plant, with interactive human assistance to escape local minima. In [Faverjon and Tournassoud, 90] the same authors present a learning scheme to avoid falling into local minima. During the learning phase, probabilities of moving between neighboring configurations without falling into a local minimum are accumulated in an r^n array, where n is the number of dof and r is the number of intervals discretizing the range of each dof. During the planning phase, these probabilities are used as another heuristic function (in addition to the potential function) to guide the robot away from the local minima. This learning scheme was applied with some success to robots with up to 6 dof. However, the size of the r^n array becomes impractical when n grows larger.

- Techniques for both computing potential functions and escaping local minima in high-dimensional C-spaces are presented in [Barraquand and Latombe, 91, Barraquand et al, 92]. The Randomized Path Planner (RPP), which is described in [Barraquand and Latombe, 91], escapes local minima by executing random walks. It has been successfully experimented on difficult problems involving robots with 3 to 31 dof. It has also been used in practice with good results to plan motions for performing riveting operations on plane fuselages [Graux et al, 92]. Recently, RPP has been embedded in a larger "manipulation planner" to automatically animate graphic scenes involving human figures modeled with 62 dof [Koga et al, 94]. However, several examples have also been identified where RPP behaves poorly [Chalou and Gini, 93, Zhu and Gupta, 93]. In these examples, RPP falls into local minima whose basins of attraction are mostly bounded by obstacles, with only narrow passages to escape. The probability that any random walk finds its way through such a passage is almost zero. In fact, once one knows how RPP computes the potential field, it is not too difficult to create such examples. One way to prevent this from happening is to let RPP randomly use several potential functions, but this solution is rather time consuming.

Other interesting lines of work include the following: A promising method based on variational dynamic programming is presented in [Barraquand and Ferbach, 94] and that method can tackle problems of similar complexity to the problems solved by RPP. In [Gupta and Gou, 92, Gupta and Zhu, 94] a sequential framework with backtracking is proposed for serial manipulators and in [Chen and Hwang, 92] a motion planner with performance proportional to task difficulty is developed for many-dof robots operating in cluttered environments. The planner in [Kondo, 91] finds paths for 6-dof manipulators using heuristic search techniques that limit the part of the C-space that is explored. Parallel processing techniques are investigated in [Chalou and Gini, 93, Lozano-Pérez and O'Donnell, 91] and in [Ahuactzin et al, 92] where a powerful planner is described. This planner utilizes genetic algorithms to help search for a path in high-dimensional C-spaces.

The planning method that we shall present in the rest of this chapter differs significantly from the methods referenced above, which are for the most part based on potential field or cell decomposition approaches. Our method applies a roadmap approach [Latombe, 91a], that is it constructs a network of paths in the free C-space. Previous roadmap methods include the visibility graph [Lozano-Pérez and Wesley, 79], Voronoi diagram [Ó'Dúnlaing and Yap, 82], and silhouette [Canny, 88] methods. All these three methods compute in a single shot a roadmap that completely represents the connectivity of the free C-space. An exception is the planner in [Canny and Lin, 90] which gradually builds a skeleton of the free C-space by using a local opportunistic strategy. However, all the visibility graph and Voronoi diagram methods are limited to low-dimensional C-spaces. In theory the silhouette method applies to C-spaces of any dimension, but its complexity makes it impractical.

In contrast, our method builds a roadmap *incrementally* using probabilistic techniques. These techniques apply to C-spaces of any dimension and produce a roadmap, or a network of connections in the free C-space, in any amount of time allocated to them. Of course, if this time is too short, the computed network may not represent the connectivity of free C-space well. Actually, in our planner, the network is never guaranteed to fully represent free C-space connectivity, though if we let our techniques run long enough it eventually will. However, while building the network, our method heuristically identifies "difficult" regions in free C-space and generates additional configurations in those regions to increase network connectivity. Therefore, the final distribution of configurations in the network is not uniform across free C-space; it is denser in regions considered difficult by certain heuristic criteria. This feature helps to construct networks of reasonable size that represent the connectivity of the free C-space well. In particular, it allows our implemented planner to efficiently solve tricky problems requiring choices among several narrow passages, i.e., the kind of problems that potential field techniques tackle poorly.

A very similar method to what we shall present has been discussed in [Overmars, 92, Overmars and Švestka, 94]. A single-shot random planner was described in [Overmars, 92] and was subsequently expanded into a learning approach in [Overmars and Švestka, 94]. In these papers the emphasis was on robots with a rather low number of dof. Similar techniques have been applied both to car-like robots that can move forward and backward (symmetrical nonholonomic robots) and car-like robots that can only move forward [Švestka, 93, Švestka and Overmars, 94]. Independently, a preprocessing scheme similar to the learning phase in the above papers was introduced in [Kavraki and Latombe, 93b] for planning the paths of many-dof robots. This scheme also builds a probabilistic network in free C-space, but focuses on the case of many-dof robots. The need to expand the network in "difficult" regions of C-space was noted there and addressed with simple techniques. Better expansion techniques were introduced in [Kavraki and Latombe, 94a, Kavraki and Latombe, 94b]. A combination of the ideas developed independently by the two above teams has been done in [Kavraki et al, 94].

Finally, it should be noted that another planner which bares similarities with our planning approach is proposed in [Horsch et al, 94]. Also, planners that have used in some way the idea of intermediate random subgoals can be found in [Glavina, 89, Eldracher, 94].

2.5 Overview of the Method

In the rest of this chapter, we present a new planning method which computes collision-free paths for robots moving among stationary obstacles (static workspaces).

The method can be applied to a wide range of robots, such as mobile robots, car-like vehicles and articulated linkages moving in the plane or in space. However, it is particularly interesting for robots with many dof, say five or more, where it exhibits a very good performance. Indeed, an increasing number of practical problems involve many-dof robots, while very few effective motion-planning methods are available to solve them (see Section 2.4). The method proceeds according to two phases: a *preprocessing phase* and a *query phase*.

In the preprocessing phase a probabilistic *network* or *roadmap* is constructed by repeatedly generating random free configurations of the robot and trying to connect these configurations using some simple, but very fast motion planner. We call this planner the *local planner*. The random network thus formed in the free C-space of the robot is stored as an undirected graph G. The configurations are the nodes of G and the paths computed by the local planner are the edges of G. The preprocessing phase is concluded by steps that aim to improve the network's connectivity. Typically, the networks produced have a large number of nodes (order of thousands). They may have one or more connected components, depending on the robot's free C-space and the time spent on preprocessing.

Following the preprocessing phase, multiple queries can be answered. A *query* asks for a path between two given free configurations of the robot. To process a query the method first attempts to connect the given start and goal configurations to two nodes of the random network, with paths that are feasible for the robot. Next, a graph search is done to find a sequence of edges connecting these nodes in the network. Concatenation of the successive path segments transforms this sequence, if one has been found, into a feasible path for the robot. Any standard smoothing algorithm can be used to improve the path.

The input to our method is a *scene*, i.e., a workspace and a robot. For example, the input can be a set of polyhedra in space and an articulated robot which consists of 7 serially connected links and has a fixed base. To run our planning method the values of several parameters must first be selected, e.g., the time to be spent in the preprocessing phase. While these values depend on the considered scene, it has been our experience that good results are obtained with values spanning rather large intervals. Thus, it is not difficult to choose one set of satisfactory values for a given scene or family of scenes, through some preliminary experiments. Moreover, increased efficiency can be achieved by tailoring several components of our planning method, in particular the local planner, to the considered robots. Overall, we found the method quite easy to implement and run. Many details can be engineered in one way or another to fit better the characteristics of an application domain.

In Chapter 3 we shall demonstrate the efficiency of the proposed method on a variety of difficult examples involving many-dof articulated linkages moving in the plane or in space. Our experimental results show that the preprocessing times required for the construction of adequate networks, i.e., networks that capture well the connectivity of the free C-space, are low. On a DEC Alpha workstation, they range from a few dozen of seconds for relatively easy problems to a few dozen of minutes for the most difficult problems we have treated. Once a good network has been constructed, most path planning queries are processed in a fraction of a second.

The very small query times make our planning method particularly interesting for many-dof robots which perform several point-to-point motions in known static environments. Examples of tasks meeting these conditions include maintenance of cooling pipes in a nuclear plant, point-to-point welding in car assembly, and cleaning of airplane fuselages. In such tasks, many dof are needed to achieve successive desired configurations of the end-effector while avoiding collisions of the rest of the arm with the complicated workspace. Explicit programming of such robots is tedious and time consuming. An efficient path planner can considerably reduce the programming burden.

2.6 Detailed Description of the Method

We now describe our path-planning method in general terms for a holonomic robot without focusing on any specific type of robot. During the preprocessing phase a data structure called the roadmap, or the network, is constructed in a probabilistic way for a given scene, i.e., a given robot and a given workspace. In the query phase, the network is used to solve individual path-planning problems in this scene. Each problem is specified by a start configuration and a goal configuration of the robot.

The network is stored as an undirected graph $G = (V, E)$. The nodes in V are properly generated free configurations of the robot and the edges in E correspond to connections between configurations: an edge (a, b) corresponds to a feasible path connecting configurations a and b. The vast majority of these paths are computed by an extremely fast, though not very powerful planner, called the local planner. The paths are not explicitly stored in the network, since recomputing them is very inexpensive. This saves considerable space, but requires the local planner to succeed and fail deterministically. There are few paths in the above network which are not computed by the local planner, but by a more powerful (but slow) path planner which we call the *auxiliary* planner. We use the Randomized Path Planner (RPP) [Barraquand and Latombe, 91] as the auxiliary planner. However, any other effective planner can be substituted here. Paths produced by the auxiliary planner are explicitly stored in the corresponding network edge, since we want to avoid spending time to recompute them. We will explain later in this chapter when the auxiliary planner is used. Throughout our description of the algorithm, we assume that the preprocessing phase is entirely performed before any path-planning query is processed. However, the preprocessing and query phases can also be interwoven and this is discussed later in the chapter.

In the query phase, given a start configuration I and a goal configuration F, the method first tries to connect I and F to some two nodes $\tilde{I}$ and $\tilde{F}$ in V. The connection is done using the local planner, or some simple randomized variant of that planner. If the connection is achieved, our method searches G for a sequence of edges in E connecting $\tilde{I}$ to $\tilde{F}$. Finally, it transforms this sequence into a feasible path for the robot by recomputing the corresponding local paths and concatenating them. The way the search of G is done can affect some properties of the path, i.e., its length. Failure is declared at this stage if we either cannot connect I or F to V, or we cannot connect $\tilde{I}$ and $\tilde{F}$ along the edges of the network (disconnected graph).

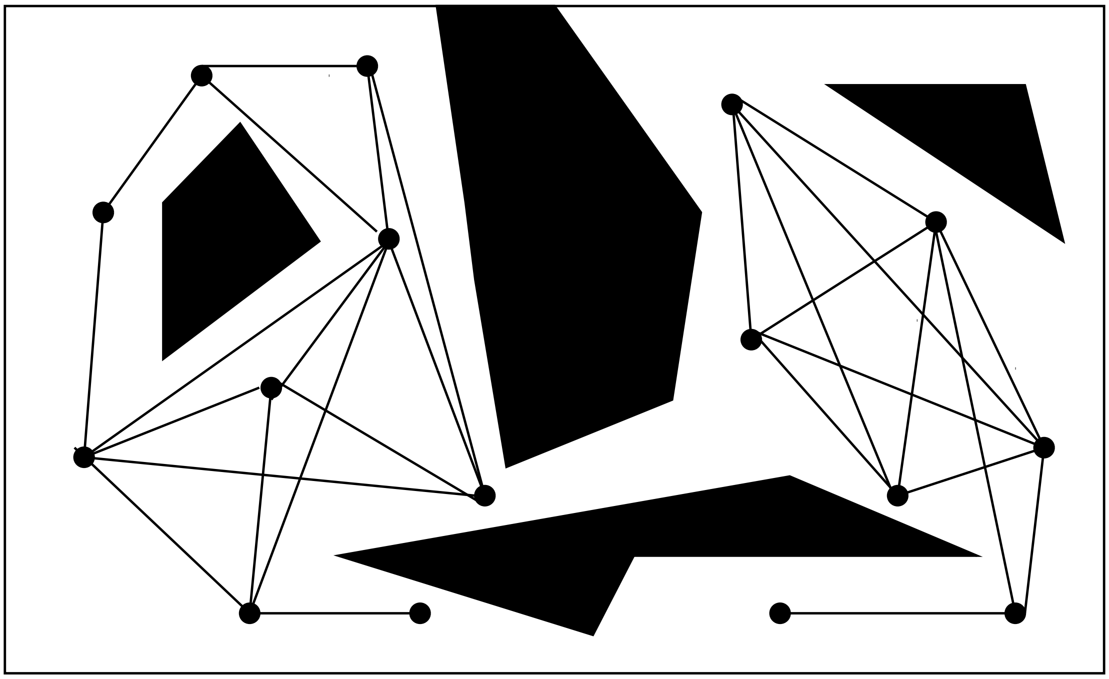


Figure 2.3: The network construction step

2.7 The Preprocessing Phase

The preprocessing phase of the method consists of three successive steps which are outlined below:

1. **Network construction step**
   During this step random configurations of the robot are generated over the free C-space $\mathcal{F}$ of the robot and are interconnected by calling the local planner. The objective of this step is to obtain a reasonably connected network in $\mathcal{F}$, with enough vertices to provide a rather uniform covering of the free C-space (see Figure 2.3).

2. **Network enhancement step**
   This step is aimed at further improving the connectivity of the network produced with the network construction step. It selects nodes of G which, according to some heuristic evaluator, lie in difficult regions of C-space and expands the

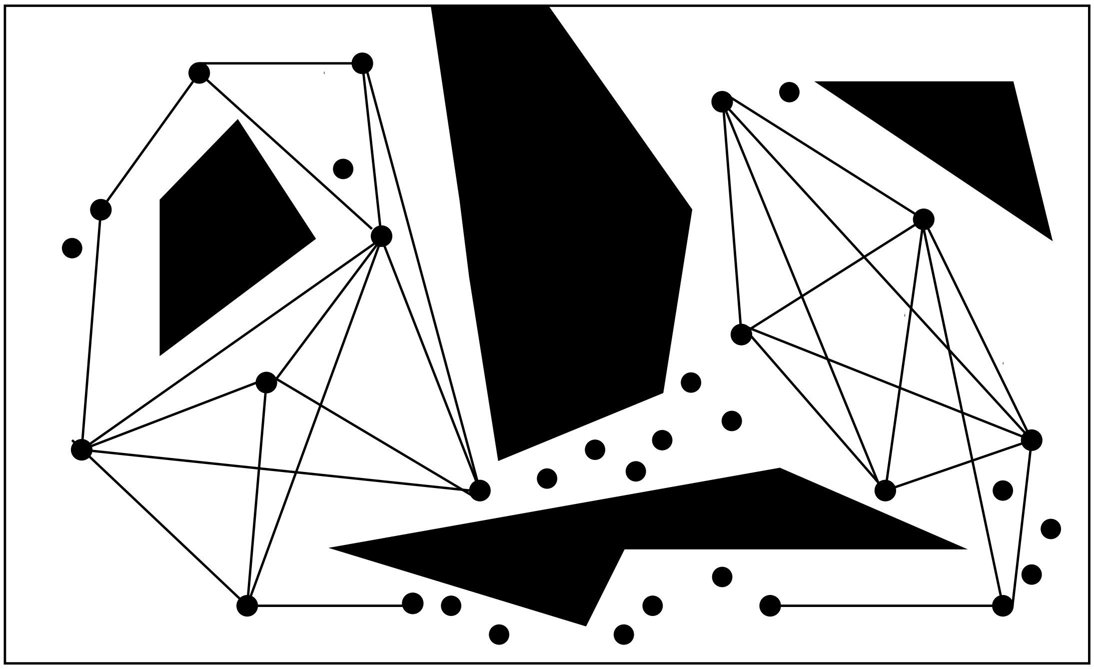


Figure 2.4: The network enhancement step

network around these nodes by generating additional nodes in their neighborhoods (see Figure 2.4). Hence, the covering of free C-space by the final network depends on the local intricacy of that space. The local planner is still used in this step.

3. **Further component reduction step**
   In the case where more than one components (of significant size) are present in the network produced at the end of the enhancement step, the auxiliary planner (RPP in our case) is used to connect nodes belonging to different components. The auxiliary planner is more powerful than the local planner. If this planner fails to connect disconnected components in our network, it is assumed that any remaining components correspond to disconnected regions of the free C-space, or to regions whose interconnection is a task above the capabilities of current planners.

The use of the last step of preprocessing is a worst-case scenario and is required only for very difficult examples involving robots with many dof. For most cases the application of the first and second steps suffices.

Preprocessing aims at generating a network that captures as well as possible the connectivity of the free C-space. Evaluation of the quality of the produced network is done in the query phase. Multiple failures or long planning times are an indication that more time should be spent in the preprocessing phase. Let us note here, that the preprocessing and the query phase can be interwoven to adapt the size of the network. For instance, a small network can be first constructed. Then it can be augmented using data generated by queries, or by reapplication of the preprocessing steps. The incremental nature of the method is one of its big advantages.

We proceed to describe in detail the three steps of the preprocessing phase.

2.7.1 Network Construction Step

This is the simplest but most time consuming part of the preprocessing. Initially the undirected graph $G = (V, E)$ is empty, i.e., $V = E = \emptyset$. Then, repeatedly, a random free configuration is generated and added to V. For every such new node x, we select a number of nodes from the current V and we try to connect x to each of them using the local planner. Whenever this planner succeeds to compute a feasible path between x and a selected node n, the edge (x, n) is added to E. The actual local path is not memorized.

The selection of the nodes to which we try to connect x is done as follows: First, a set N_x of candidate neighbors is chosen from V. This set is made of nodes within a certain distance of x, for some metric D. Then we pick nodes from N_x in order of increasing distance from x and we try to connect x to each of the selected nodes.

In our description of the algorithm we consider that the local planner is a symmetrical function $\Delta : \mathcal{F} \times \mathcal{F} \to \{0, 1\}$, which returns whether a feasible path can be obtained between the two free configurations given as its arguments. The distance function D is a function $\mathcal{C} \times \mathcal{C} \to R^+ \cup \{0\}$, which defines a pseudo-metric in $\mathcal{C}$. We

```
1   V = ∅
2   E = ∅
3   loop N times
4       Select a random free configuration x and put V = V ∪ {x}
5       Select the neighbors N_x of x from V
6       For each n ∈ N_x in order of increasing D(x,n) :
7         If Δ(x,n) then E = E ∪ {(x,n)}
8   endloop
```

Figure 2.5: Algorithm for the network construction step

only require that D be symmetrical and non-degenerate.

The algorithm for the network construction step is shown in Figure 2.5. To clarify further this description we need to explain how random nodes are generated (line 4), what it means for a node to be a neighbor of another (line 5), how the distance function is specified (line 6) and what are the requirements for the local planner.

Generation of Random Nodes

During the network construction step, a total of N nodes are generated in $\mathcal{F}$, the free part of the C-space. We try to produce a rather uniform distribution of these N nodes in $\mathcal{F}$. One way to approximate this is by sampling the value of each dof of the robot uniformly and independently from the interval of the allowed values of the corresponding dof. The resulting configuration must be checked for collision with the obstacles. For most robots that are not simple rigid bodies (e.g., articulated robots) we should also check for autocollision, that is make sure that distinct bodies of the robot do not intersect, and for other limitations imposed by the design of these robots (e.g., mechanical stops). A configuration is declared valid only if it passes these tests.

Collision checking can be done using a variety of existing general techniques [Giezeman, 93, Gilbert et al, 88, Newman and Branicky, 91, Quinlan, 94]. In two-dimensional workspaces we can use a faster but more specific collision checker [Kavraki, 93] (see Chapter 5). The use of iterative collision checkers, like the one in [Quinlan, 94], may be especially beneficial in three-dimensional workspaces. This checker automatically generates successive approximations of the objects and the robot. It can determine efficiently if collisions occur at a coarse but reasonable resolution. Configurations that are collision-free with that resolution are the only ones checked further for collisions.

It must be emphasized that for complex robots, only a very small percentage of the generated configurations are collision-free. For example, if the robot is an articulated serial linkage with 7 dof moving in the plane, it is typical that less than 0.1% percent of the generated configurations are valid configurations. However, the potentially large number of configurations that have to be generated is not the most expensive step of the algorithm. It is the interconnection of these configurations that takes most of the preprocessing time.

The Local Path Planner

Our choice for the planner used to achieve connections between configurations of the robot is a simple and fast deterministic planner. We emphasize here simplicity versus effectiveness. It is to be expected that the local planner will fail most of the time to connect two configurations which can nevertheless be connected to each other. What *is* expected from the local planner though is that it will succeed to connect most pairs of configurations that are close to each other. The closeness is measured in terms of the heuristic distance D in C-space.

Concerning how fast the local planner should be, there is clearly a tradeoff between the time spent in each individual call of this planner and the number of calls. If a powerful local planner was used, it would often succeed in finding a path when one exists. Hence, relatively few nodes would be required to build a network capturing the connectivity of the free C-space sufficiently well to reliably answer path-planning queries. Such a local planner would probably be rather slow, but this could be somewhat compensated by the small number of calls needed. On the other hand, a very fast planner is likely to be less successful. It will require more configurations to be included in the network ; so, it will be called more often, but each call will be less expensive.

We have four arguments in favor of using a fast and deterministic local planner instead of a more powerful (and slow) one:

1. The free part of the C-space in high dimension is very complicated. It is frequently the case that in order to move from the initial to the final position, the point-robot has to go through a narrow passage. But the only way this can be achieved with our method (despite of the enhancement step which we have not yet described) is to be lucky enough to place several of our random points in the neighborhood of the narrow passage. For this to have a significant probability of happening, the total number of random points that we initially throw in the free C-space *has* to be large. Thus, we do not need a powerful local planner since all the pairs of nodes that we are going to try it on will be very close to each other, and thus have a high chance of getting connected with the simple and fast planner.

2. The choice of the local planner also affects the query phase. The purpose of having a preprocessing phase is to make it possible to answer path-planning queries quasi-instantaneously. It is thus important to be able to connect any given start and goal configurations to the network, or to detect that no such connection is possible, very quickly. This requires that the network is dense enough, so that it always contains a few nodes (at least one) to which it is easy to connect each of the start and goal configurations. It thus seems preferable to use a very fast local planner, even if it is not too powerful, and build large networks with configurations widely distributed over the free C-space. Furthermore, the same local planner that is used in the preprocessing phase can be used in the query phase to connect the start and goal configurations to the network.

3. Parallelization of the preprocessing phase of the planner can make the method very attractive as discussed at the end of this chapter. In the network construction step, the tasks of generating random nodes in C-space and trying to

connect pairs of them can be carried out independently. So the task of generating a larger set of nodes which are to be connected using a simple local planner is easier to parallelize. In addition to that, a simple local planner is easier to implement in specialized hardware which would significantly improve the performance of the method. We adopt here the philosophy that it is better to have a large number of small tasks than have a smaller number of larger tasks.

4. Perhaps the clearest argument of having a fast deterministic planner is that we do not have to store the paths that are computed by the planner but just the fact that they were found. As a result, the storage requirements for the produced networks are low. During the query phase, whenever we need a certain path, we run the fast local planner with the original arguments and we retrieve it. The query times are not significantly affected by the recomputation of these paths.

A quite general local planner which satisfies the above requirements and is applicable to all holonomic robots, could work as follows. Given any two configurations of the robot in d-dimensional space, that is two d-tuples

$$a = (a_1, \ldots, a_d), \text{ and } b = (b_1, \ldots, b_d), \text{ with } a_1, \ldots, a_d, b_1, \ldots, b_d \in \mathbf{R},$$

the planner tries to connect a and b by moving along the straight-line segment ab in C-space ($\mathbf{R}^d$). In other words a sufficiently dense set of points on the set

$$\{(1-t)\cdot a + t\cdot b \ : \ t \in [0,1]\}$$

is checked for collision. This can be done as follows [Barraquand and Latombe, 91]: First, discretize the line segment (more generally, any path generated by the local planner) into a number of configurations $c_1, \ldots, c_m$, such that for each pair of consecutive configurations (c_i, c_{i+1}) no point on the robot, when positioned at configuration c_i, lies further than some eps away from its position when the robot is at configuration c_{i+1} (eps is an input positive constant). Then, for each configuration c_i, test whether the robot, when positioned at c_i and "grown" by eps, is collision-free. If none of the m configurations yield collision, conclude that the path is collision-free. Since eps is constant, the computation of the robot bodies grown by *eps* is done only once. In the following we will refer to the above local planner as the *general local planner*.

Let us note that by changing the parametrization of C-space, we can change the actual motion of the robot in the workspace (the one that corresponds to a straight line in C-space) in many ways. Nevertheless we have found that with a reasonable parametrization the above technique works well. If the method is applied to a specific class of robots (i.e., articulated robots), it is possible to write a local planner that takes advantage of the geometry and the structure of these robots in a more sophisticated way. In Chapter 3 we give some examples of local planners that are engineered towards specific classes of robots. Such planners can further boost the performance of the method.

Neighbors of a Node

The most time-consuming part of the network construction step is the interconnection of the random configurations, that is the cumulative cost of the invocations of the local planner. To reduce this cost, it is important to avoid calling the local planner for configurations which are unlikely to get connected with the technique used. This is the motivation behind the definition of the *neighborhood* N_x of a node x. We try to avoid calls of the local planner that are likely to return failure, by submitting only pairs of configurations whose relative distance (according to the distance function D) is smaller than some constant threshold *max_distance*. Thus

$$N_x = \{y \in V : D(x,y) \leq max\_distance\}.$$

Sometimes the neighborhood of a node may consist of too many nodes. This is for example the case when the node lies in a large "easy" part of the C-space with very little C-obstacle around it. In our experiments we found it useful to bound the size of the set N_x by some constant *max_neighbors* (typically in the order of 30). This additional criterion guarantees that, in the worst case, the running time of each iteration of the main loop of the construction step algorithm is independent of the current size of V. Thus, the above choice makes the relation between N and the running time of the network construction step almost linear. We can safely ignore the time it takes for each node to determine its neighborhood, which is quadratic, since this affects only a tiny fraction of the running time (essentially because this operation does not involve any collision checking).

Distance of Two Nodes

The distance function $D(\cdot,\cdot)$ is used to both construct and sort the set N_x of candidate neighbors of each new node x. Nodes which, with respect to $D(\cdot,\cdot)$, lie further away than *max_distance* from x are discarded as potential neighbors. We are not attempting to connect them with x, since we believe that there is little chance that our local path planner will be able to make this connection.

Thus to improve our chances of obtaining a good network in C-space, the distance function $D(x,y)$ should be defined in a way that reflects the likelihood that the local planner will fail to connect the two nodes x and y. In other words, $D(x,y)$ should be large if nodes x and y are unlikely to be connected by the local path planner.

If, for example, we are using the general local planner that joins two configurations via a straight line in d-dimensional C-space, then it is reasonable to define $D(x,y)$ to be the Euclidean distance of x and y in $\mathbf{R}^d$. $D(x,y)$ may also be defined as the longest Euclidean distance that any point on the robot travels in the workspace, when the robot moves along the line segment joining x and y in the configuration space, i.e.,

$$D(x,y) = \max_{z \in \text{robot}} |z(x) - z(y)|,$$

where z denotes a point on the robot, $z(x)$ is the position of z in the workspace when the robot is at configuration x, and $|z(x) - z(y)|$ is the Euclidean distance between $z(x)$ and $z(y)$.

2.7.2 Network Enhancement Step

At the end of network construction step we have a graph $G = (V, E)$, where the node set V consists of N random configurations in the free C-space and two nodes are joined by an edge if we have managed to connect them using the local path planner. If the number of nodes generated during the construction step is large enough, the set N gives a fairly uniform covering of the free C-space $\mathcal{F}$. In easy scenes G is then well connected. But in more constrained ones where the free C-space is actually connected, G often consists of a few large components and several small ones. It therefore does not effectively capture the connectivity of $\mathcal{F}$.

The enhancement step is intended to improve the connectivity of the graph G generated by the construction step. Typically, if the graph is disconnected in a place where $\mathcal{F}$ is not, this place corresponds to some narrow, hence difficult region of the free C-space. The idea underlying the enhancement step is to select a number of nodes among those generated by the construction step which are likely to lie in such regions and to "expand" them. By expanding a configuration x, we mean selecting a new free configuration in the neighborhood of x, adding this configuration to V, and trying to connect it to other nodes of V, in the same way as in the construction step. So, the enhancement step increases the density of the network configurations in regions of $\mathcal{F}$ that are believed to be difficult. Since the "gaps" between components of the graph G are typically located in these regions, the connectivity of G is likely to increase.

We propose the following probabilistic scheme for the enhancement step: With each node x generated in the construction step, we associate a positive weight $w(x)$ that is a heuristic measure of the "difficulty" of the region around x. Thus, $w(x)$ is large whenever x is considered to be in a difficult region. We normalize $w(\cdot)$ so that all weights together (for all N nodes) add up to one. Then, we repeat the following step M times: we choose a node from among those N nodes that the construction step generated with probability

$$Pr(x \text{ is selected}) = w(x),$$

and we expand this node. Suppose that y is the node resulting from the expansion. y is tried for connection with the nodes in V as in the network construction step. The detailed algorithm for the enhancement step is shown in Figure 2.6. The total number of nodes in V at the end of the enhancement step is $N + M$.

Given (V, E), compute the function $w(x), x \in V$
loop M times
    Select a configuration x, x generated in the construction step,
      with probability $Pr(x \text{ is selected}) = w(x)$
    Expand x and obtain configuration y
    $V = V \cup \{y\}$
    Select the neighbors N_y of y from V
    For each $n \in N_y$ in order of increasing $D(y, n)$:
      If $\Delta(y, n)$ then $E = E \cup \{(y, n)\}$
endloop

Figure 2.6: Algorithm for the network enhancement step

Suppose for a moment that the function $w(\cdot)$ adequately identifies the difficult parts of the C-space by having a large value at those parts and small value in the "easy" parts. Then, if x lies in a difficult region, $w(x)$ should have a relatively large value. As a result x will have a large probability of being selected and many random nodes will be placed in its neighborhood. What our heuristic achieves is to fill areas with large $w(\cdot)$ value more than others.

It now remains to define the heuristic weight $w(\cdot)$. It is obvious that $w(\cdot)$ is the essential parameter in our enhancement scheme. It is important that this function is a good measure of "difficulty" in C-space. Additionally, $w(\cdot)$ must be an easily computable function. We should note that the identification of the difficult parts of the C-space is no simple matter and the choices that we propose below clearly go only a certain distance in this direction.

One possibility for $w(x)$ is to count the number of nodes of V lying within some predefined distance of x. If this number is low, the obstacle region probably occupies a large subset of x's neighborhood. This suggests that $w(x)$ could be defined inversely proportional to the number of nodes within some distance of x. Another possibility is to look at the distance $dist_x$ of x from the nearest connected component not containing x. If this distance is small, then x lies in a region where two components failed to connect, which indicates that this region might be a difficult one (it may also be actually obstructed). This idea leads to defining $w(x)$ inversely proportional to $dist_x$. Alternatively, rather than using the nodes of G to identify difficult regions, we could define $w(x)$ according to the behavior of the local planner. For example, if the local planner often failed to connect node x to other nodes, this is also an indication that x lies in a difficult region. The number of connections that the local planner has obtained for a node is also a measure of the difficulty of the area in which the node lies. Which particular heuristic function should be used depends, to some extent, on the input scene.

Nevertheless, the following function, which is based on the idea of using the behavior of the local planner to determine difficult parts, has produced good results whenever we tried it. We define the *degree* d_x of a node x as the number of connections that x has with other nodes at the end of the network construction step. In other words, d_x denotes the number of successful calls to the local path planner when we attempted to connect the node x to the nodes in its neighborhood N_x (or the first $max\_neighbors$ of the nodes in N_x, in case N_x contained more than $max\_neighbors$ configurations). The numbers d_x can be computed very easily. It is a reasonable assumption that a node x which is in a "difficult" area of free C-space will have a smaller d_x than a node which is away from the C-obstacle. Our strategy will put configurations around nodes with small d_x so that we have better chances of going through "narrow passages" and thus create larger components.

The function $w(x)$ can be computed as follows

$$w(x) = \frac{1}{d_x + 1} \Big/ \sum_{t=1}^{N} \frac{1}{d_t + 1}.$$

Clearly, $w(x)$ gives small weight to the nodes which have large d_x and larger weight to those with smaller d_x. The division by the sum $\sum_{t=1}^{N} \frac{1}{d_t+1}$ normalizes the sum so that

$$\sum_{x=1}^{N} w(x) = 1.$$

The addition of 1 in the denominator serves to avoid trouble in cases when $d_x = 0$. There is no argument on why we choose $w(x)$ as this specific function of d_x and not, say, $w(x) = (d_x + 1)^{-2}$ appropriately normalized so as to have $\sum_{x=1}^{N} w(x) = 1$. Indeed, it seems that any decreasing function of d_x could be used in the place of the one we chose and it is true that many such functions give comparable results. Our choice should be viewed as perhaps the simplest one.

The above choice of $w(\cdot)$ has been shown to work well in a large number of experiments that we have performed (Chapter 3). These experiments indicate that the enhancement step indeed helps

1. to obtain non trivial connections between sizable components by placing many nodes between them and
2. to connect more of the initial randomly generated nodes to the main components.

From the experimental results of Chapter 3 another important observation is made. The addition of M nodes to the N nodes created by the network construction step using the enhancement step that we just described, as opposed to just throwing in the C-space $N+M$ random nodes and doing no enhancement at all, gives much better results in terms of effective covering of the free C-space.

Let us now discuss how to perform the expansion of a node. Again here, several techniques give good results. One way to accomplish the expansion is to choose each of the parameters that describe the configuration uniformly at random from a small interval centered at the current value of the corresponding parameter of x. Another way, which is the one we use in our implementations, makes use of what we call random-bounce walks (or rbw). For holonomic robots, an rbw consists of repeatedly picking at random a direction of motion in C-space and moving in this direction until an obstacle is hit. When a collision occurs, a new random direction is chosen. And so on. To expand a node x, we compute a rbw starting from x. We limit the length of the rbw to a small number of steps which we call $rand\_bounce\_length$ (say, 100 steps). The final configuration n reached by the rbw and the edge (x, n) are included into G. The path computed between x and n is explicitly stored, since it was generated by a non-deterministic technique. Then we try to connect n to the other nodes of the network in the same way as in the construction step. The enhancement step thus never creates new components in G. At worst, it fails reducing the number of existing components.

To complete our discussion of the algorithm of Figure 2.6, let us discuss how large is the fraction of nodes that is created with the enhancement process. We have found experimentally that $M = N/2$ gives very good results. The number N cannot be made too small: we cannot hope to cover a narrow passage during enhancement if initially we do not manage to put *any* points near that passage that will be used to guide the enhancement in that area.

Once the enhancement step is over, the remaining small components of G, if any, are discarded. Here, a component is considered small if its number of nodes is less than some *min_component* percent (typically 1%) of the total number of nodes in V. The graph G that remains after discarding small components may contain one or several components.

In case G contains one major component, which in our examples happened very often, this component can be used directly for planning and there is no need for the third step of the preprocessing. If, on the other hand, at the end of the enhancement step we end up with more than one connected components of significant size, we can elect to spend some more time in trying to merge them into one component, or, at any rate, reduce the number of components. This of course is impossible if the number of components reflects the reality in C-space, that is it may well be the case that $\mathcal{F}$ *is* disconnected. And there is unfortunately no simple way to tell whether this is the case or not. The way we choose to view the situation therefore is the following: if at the end of the second step we do not have one major component, we spend some extra, but a priori fixed time, in trying to merge existing components using either (a) a repetition of the network construction and enhancement steps, or (more often) (b) a more powerful technique for connecting configurations from different components, which will be aided from the fact that it is now up to us to choose these configurations, as explained in the next section.

2.7.3 Further Reduction of the Number of Components

In case multiple major components remain at the end of the enhancement step we may decide that it is worth spending some more time in trying to reduce the number of these components. For this we invoke a more sophisticated path planner than the local planner that we have used so far, the auxiliary planner. In our experiments we used the Randomized Path Planner (RPP) as the auxiliary planner (described in Section 2.4).

We proceed as follows: We select any pair of connected components, say A and B. Our purpose is to use the auxiliary planner in order to connect some node, say a, from component A to some node b from component B. The most important choice that we have to make is to select nodes a and b so that the auxiliary planner will have a good chance to connect them in a short time. A reasonable strategy for this selection seems to be to select that pair of nodes a and b for which the heuristic distance $D(a, b)$ becomes minimum over all possible choices of a and b.

Our strategy follows this spirit but is slightly different in order to be less prone to failure. Assume that component A is larger than component B. We then select a node b at random from B (with equal probability of selecting any node in B) and order the nodes of component A in increasing distance D from the node b. We then invoke the auxiliary planner to connect b with the first node in the list (i.e., the closest to b node of A) and we allow the auxiliary planner to run for some predefined interval of time. If it fails we try to connect b with the next closest node from A and so on, until we either succeed or try a certain prespecified number of nodes of A for connection with b. If we have not succeeded by then we declare failure and components A and B will remain unconnected, unless they get connected indirectly through a sequence of connections of components that succeed in this step. If components A and B *do* get connected by the auxiliary planner, we merge them into a new component.

The above procedure is repeated for different components until there is either one connected component only, or a prespecified amount of time has elapsed. This prespecified amount of time is, in our examples, the sum of the times spent in the network construction and enhancement steps. The philosophy behind our choice not to try to connect the pair (a, b) which achieves minimum distance over all pairs in $A \times B$ is that we want to allow a certain randomness in the process and avoid bad choices. The minimum-distance selection may give us a pair of nodes that is very difficult to connect with the auxiliary planner. Failure to obtain one connected component at the end of this step is a strong indication that $\mathcal{F}$ is not connected, or that the connection of any remaining components is beyond the capabilities of existing planning techniques.

2.8 Query Phase

During the query phase, paths are to be found between arbitrary input initial and final configurations, using the network constructed in the preprocessing phase. Assume for the moment that the free C-space is connected and that the network consists of a single connected component G. A query is answered in the following way: Given an initial configuration I and final configuration F, we try to connect I and F to some two nodes of G, respectively $\tilde{I}$ and $\tilde{F}$, with feasible paths P_I and P_F. If this fails, the query fails. Otherwise, we compute a sequence P of nodes in G connecting $\tilde{I}$ to $\tilde{F}$. A feasible path from I to F is eventually constructed by concatenating P_I, the local paths recomputed by the local planner when applied to pairs of consecutive nodes in P, and P_F reversed. Figure 2.7 illustrates the above idea.

The main question is how to compute the paths P_I and P_F. The queries should preferably terminate quasi-instantaneously, so no expensive algorithm is desired here. Our strategy for connecting I to G is to consider the nodes in G in order of increasing distance from I (according to D) and try to connect I to each of them with the local planner, until one connection succeeds. We ignore nodes located further than *max_distance* away from I, because we consider that the chance of success of the local planner is too low. If all connection attempts fail, we perform one or more random-bounce walks, as described in Subsection 2.7.2. But, instead of adding the node at the end of each such rbw to the network , we now try to connect it to G with the local

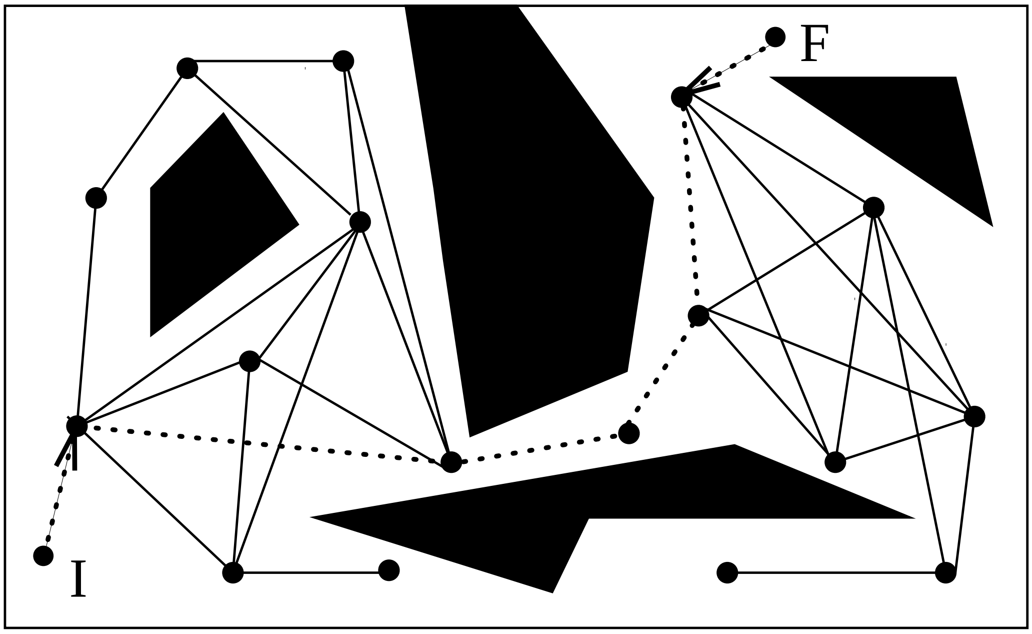


Figure 2.7: Answering a path planning query

planner. As soon as I is successfully connected to G, we apply the same procedure to connect F to G.

If we did not succeed to connect either I or F to the network then our planner declares failure. In practice, we bound the time allocated to each query by a user defined constant *query_time* and we declare failure if P_I and P_F have not been found within that time. Assume now that I was connected to node $\tilde{I}$ and F to node $\tilde{F}$ of the network (remember that our network is connected). Our task is then completed by searching the network for a path (along the edges of the network now) connecting $\tilde{I}$ to $\tilde{F}$.

This can be done, for example, in a very straightforward way if we start propagating a "wave" from node $\tilde{I}$ until it reaches node $\tilde{F}$ (breadth-first search). In this way we are going to find a path from $\tilde{I}$ to $\tilde{F}$ which is minimal in length (one network edge having "length" 1). But we can also choose to use standard graph algorithms to find paths from $\tilde{I}$ to $\tilde{F}$ that optimize other quantities. We could, for example, assign a weight $\rho(a, b)$ to the edge (a, b), where $\rho(a, b)$ is the heuristic distance used in the preprocessing phase, or any other function that measures the difficulty of moving the specific robot from configuration a to configuration b. It should be obvious that the quality of the paths produced in this manner (or at least of the parts of the paths that connect nodes on the network) is quite good.

Having found a path that joins I to F, we can now forget the network and invoke any path smoothing algorithm that takes as input a collision-free path from I to F and returns one with the same endpoints but with improved characteristics as far as the total length, smoothness or other quantities of interest are concerned.

In general, however, the network may consist of several connected components G_i, $i = 1, \ldots, p$. This is always the case when the free C-space is itself not connected. It may also happen when the free C-space is connected if, for instance, the network generated is not dense enough. If the network contains several components, we first try to connect both I and F to two nodes in the *same* component. To do this, we consider the components of the network in order of increasing distance from $\{I, F\}$; for each component we proceed as we did above with the single component G. We define the distance between $\{I, F\}$ and a component G_i as follows: Let the distance $D(x, G_i)$ between a configuration x and G_i be the minimum of $D(x, n)$ for all $n \in G_i$. The distance between $\{I, F\}$ and G_i is the maximum of $D(I, G_i)$ and $D(F, G_i)$. If the connection of I and F to some component G_i succeeds, a path is constructed as in the single-component case. The method returns failure whenever it fails to connect both I and F to the same network component. Since in most examples the network consists of rather few components, failure is rapidly detected.

Finally, we should note that certain kinds of local planners render unnecessary the recomputation of collisions along the network edges when the corresponding paths are reconstructed. This makes the planning stage even faster. For example, the general local planner of Subsection 2.7.1 aborts when a collision is detected. During planning time, intermediate configurations on a path induced by this planner have to be recomputed, since they have not been stored, but we do not need to check each of them for collision. The situation is different if the local planner does not abort when a collision is detected but performs a certain action. Then, in the planning stage

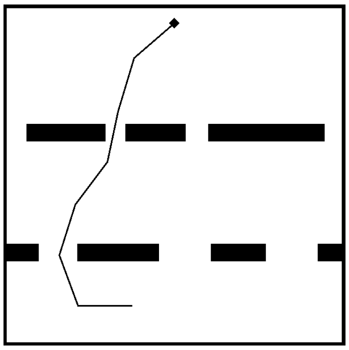


Figure 2.8: An example: a serial robot with 7 links and a fixed base

collision must be checked along the recomputed path so that the same action can be repeated just after the collision is detected.

2.9 The Numerical Parameters of the Method

Having completed the description of the method, we summarize here all its numerical parameters. The values of these parameters should be specified by the user. For most of them there is no well-defined method of selecting them, and in our tests we chose them more or less by experimentation. The situation is partly remedied by the fact that, according always to our observations, the performance of the algorithm does not depend on the value of any of them in an unstable manner. That is, there is usually a quite large range of values for each parameter in which the method performs well.

The typical values of the parameters that we mention here concern the range of scenes that we present in Chapter 3. As an example, we show in Figure 2.8 a 7-dof robot moving in a planar workspace. The workspace is the unit square $[0,1]^2$ and the length of each of the seven links of the robot is 0.16. The fixed base of the robot is denoted with a thick square. A more detailed picture of this robot can be found in Figure 2.2(b).

1. Number of random nodes N

   This is the number of nodes that will be thrown at random in the free C-space during the network construction step. It is in the order of a few thousand (2,000-3,000). The running time of the network construction step of the preprocessing is roughly linear with N.

2. Number of nodes during enhancement M

   This is the number of nodes that will be added during the enhancement step of the preprocessing. We have found that the choice $M = N/2$ works very well in practice. Again, the running time of the network enhancement step is roughly linear with M.

3. Time spent in the third step of preprocessing (further component reduction)

   We bound the time allocated to this step by the time we have spent in constructing the network so far, that is the sum of times spent in the network construction and enhancement steps. We note however that the third step of the preprocessing is rarely used.

4. Time spent per individual planning query $query\_time$

   We require our queries to finish quasi-instantaneously and we usually limit this time to a few seconds (2-5 seconds). Most queries finish within a fraction of a second, once a good network has been constructed. $query\_time$ provides an upper bound on the time the query will take before declaring failure.

5. $max\_distance$, the distance that defines a neighborhood

   We remind the definition for the neighborhood of a node x

$$N_x = \{y \in V : \ D(x,y) \le max\_distance\}.$$

   For an example of how large $max\_distance$ should be, let us consider the robot of Figure 2.8. Suppose we define the distance D of two configurations by

$$D(x,y) = \left(\sum_{k=1}^{7} |J_k(x) - J_k(y)|^2 \right)^{1/2},$$

   where $J_k(x)$ denotes the position in the workspace of point J_k of configuration x (see Figure 2.2(b)), and $|a - b|$ denotes the Euclidean distance of the points

a and b of the plane. Good results are obtained if we choose

$$max\_distance = 0.427,$$

which, translates into a (mean square) displacement of 0.161 for each joint of the robot.

6. $max\_neighbors$, **the maximum number of neighbors examined for connection**
   If the neighborhood N_x of a node x has more than $max\_neighbors$ configurations in it, we keep only the first $max\_neighbors$ of them when we are trying to connect x with its neighbors. The remainder configurations of N_x are not examined for connection with x. This makes the running time of the network construction and enhancement steps roughly linear with the number of nodes added in the graph during these steps. A typical value for this parameter is 30.

7. $rand\_bounce\_length$, **the random bounce path length**
   Whenever we perform a random bounce walk in the free part of the C-space we let it run for a certain fixed number of steps which we call $rand\_bounce\_length$. We use consistently the value 100 for the number of these steps.

8. **Small components**
   Components with less than $min\_component$ percent of the total nodes at the end of the network enhancement step are regarded small and are not considered further in the third step of the preprocessing, or in the query phase. A typical value of $min\_component$ is 1%, that is components with less than $0.01 \cdot |V|$ nodes are discarded from the network.

The values given above for the last three parameters have been kept fixed across a large range of examples (see Chapter 3). The fact that the performance of the method does not crucially depend on any of them is a very good feature. Overall, we have not found it difficult to obtain good values for the numerical parameters of the method with little experimentation.

2.10 Some Remarks

We have described a two-phase method to solve robot motion-planning problems in static workspaces. In the preprocessing phase, the method constructs a probabilistic network as a collection of configurations properly selected in the free C-space. In the query phase, it uses this network to quickly process path-planning queries, each specified by a pair of configurations. The method was described for a general holonomic robot. Several aspects of this method deserve further investigation.

One of these aspects is if the performance of the method can be further increased when the method is customized to a particular application. Customization is the substitution of some of the components of the method that were designed for a general robot (i.e., local planner) with techniques that fit better the characteristics of the considered scenes. In the next chapter we answer this question in the affirmative. We also show the easiness with which the method can be applied to a specific class of robots (i.e., articulated robots) and experiments which confirm that the customized implementation exhibits a better performance than the general method.

An issue not further explored in this dissertation is the incremental nature of this method. The preprocessing and the query phases do not have to be executed sequentially. Instead, they can be interwoven to adapt the size of the network to difficulties encountered during the query phase, thus increasing the learning flavor of our method. For instance, a small network could be first constructed; this network could then be augmented (or reduced) using intermediate data generated while queries are being processed. If query times are not satisfactory for the application considered, more nodes can be added to the network. This is simply done by resuming the construction step algorithm and/or the enhancement step algorithm, starting with the current network. We can also take advantage of the incremental nature of the method to conduct trial-and-error experiments in order to decide how much computation time should be spent in the preprocessing phase. Let us note here that the sizes of the produced networks are very small since all the paths produced by the local planner are not stored. Thus, given enough preprocessing time, a very large network that has good chances of capturing well the connectivity of the free C-space can be gradually obtained.

We conclude this chapter by emphasizing that the randomized method we described lends itself to parallelism. It escapes the sequential way of searching for a path from the initial to the final configuration of the robot. Instead, it looks at the whole C-space from the beginning and tries to acquire information about the connectivity of this space. The preprocessing phase, which is the computationally expensive phase, can be massively parallelized. Take the network construction step of the preprocessing for example. The random nodes can be generated in different processors and the interconnection of each of these nodes with its neighbors can be done independently in different processors. Having a fast local planner permits to balance the workload among the processors used easily. There is virtually no communication in the network construction step. The same arguments are valid for the network enhancement step of the preprocessing. Once each processor has a copy of the probability distribution function used in this step, it can start selecting nodes among those generated by the construction step and expanding them. The third step of the preprocessing phase requires calling a sequential planner, the auxiliary planner. Here different calls of this planner can be made for different pairs of nodes belonging to the components that we wish to connect. These calls can be initiated at independent processors and all of them will be aborted once one of them succeeds. This step involves more communication among the processors involved. However, the third step of the preprocessing is hardly ever used in the one processor case, and we believe that the increased computational power resulting from a parallel implementation of the method will further eliminate the need of this step. Parallelization will make the method even more interesting since it will radically reduce preprocessing time.

Chapter 3

Application of the Planning Method to Articulated Robots

3.1 Introduction

This chapter describes the application of our planning method to articulated robots with fixed or free bases moving in the plane or in space. We present techniques specific to these robots that can be substituted for more general techniques in the planning method in order to increase its efficiency. Experimental results reported in this chapter confirm that robot-specific components can further improve the performance of the method, although the general technique remains quite powerful even for difficult path-planning problems.

The purpose of presenting customized versions of our method is to illustrate the easiness with which the general method for holonomic robots of Chapter 2 can be engineered to better suit the needs of a particular application. Sections 3.2, 3.3, and 3.4 report extensive experiments with articulated robots which show the effectiveness of the method, its dependence on the numerical parameters and the value of the network enhancement step in difficult examples.

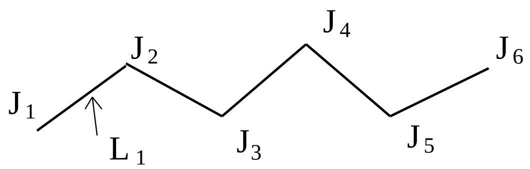


Figure 3.1: A planar articulated robot

3.2 Articulated Robots in the Plane

In this section we consider planar articulated robots with revolute and prismatic joints having one or multiple kinematic chains. Our examples involve robots whose links are line segments. In practice of course the links of robot arms are not line segments. Our simplification does not affect in any way the description of our method since the shape of the links is information that we can always incorporate into the collision checker.

Consider for a moment the robot shown at Figure 3.1. The links of that robot are denoted by L_1 through L_q (in the figure, $q = 5$). Points J_2 through J_q designate revolute joints. Point J_1 denotes the base of the robot; it may, or may not, be fixed relative to the workspace. If it is fixed, then J_1 is also a revolute joint. If it is not, then J_1 can translate freely in the plane and the robot is said to have a free base. The point J_{q+1} (J_6 in the figure) is called the endpoint of the robot; actually, it is any point on the last link, preferably the one located the furthest away from J_q. Similarly, if the robot's base is free, J_1 can be any point on L_1, preferably the one located the furthest away from J_2. Each revolute joint J_i ($i = 1$ or 2 to q) has defined certain internal joint limits, denoted by low_i and up_i, with $low_i < up_i$, which constrain the range of the possible orientations that L_i can take relative to L_{i-1}. If the robot's base is free, the translation of J_1 is bounded along the x and y axes of the Cartesian coordinate system embedded in the workspace by low_x and up_x, and low_y and up_y, respectively. Joints can also be prismatic. If joint J_2 of the robot in Figure 3.1 is prismatic, the relative displacement of J_2 and J_3 can be a translation, sometimes called the joint offset. In other words, when a prismatic joint is present, the corresponding link, L_2 in our case, can be thought of as an extensible link.

Let us make our discussion more precise by considering three examples of planar articulated robots:

1. **Serial robot with 7 links and with a fixed base (7 dof)**
   Figure 3.2 shows a robot with 7 links of the same length L moving in a planar workspace. J_1 in this robot has a fixed position and the parameters of the C-space are the seven orientations of the links. The obstacles are in black and the robot's fixed base is marked with a thick square.

2. **Serial robot with 5 links and without a fixed point (7 dof)**
   The parameters of the C-space of the robot shown in Figure 3.3 are the two coordinates x and y of J_1, plus the five orientations that define the position of the five links. One of the ends of the robot in the above-mentioned figure is marked with a square in order to distinguish it from the other. We do not want to identify two configurations of the robot which do not differ in the geometry but only in the order in which the links are placed in the workspace. Again, all the links of the robot are of the same length.

3. **Multiple-chain (hand-like) robot with a fixed base and 7 links of which the first 3 are extensible (10 dof)**
   The hand-like robot of Figure 3.4 has a fixed base and its first 3 links are connected by prismatic and revolute joints. The length of these links can vary between L and $2L$. Then the chain of links branches into two sequences of two links of length L each. The parameters of the C-space are the seven orientations of the links plus the lengths of the first three extensible links. The fixed base of the robot is again denoted by a thick square.

For all the robots described above, we do not allow any two consecutive links to form an angle smaller than 10 degrees. This restriction was imposed to simulate mechanical stops.

To customize the general method of Chapter 2 to a particular class of robots, we tailor the computationally expensive components of the method to these robots.

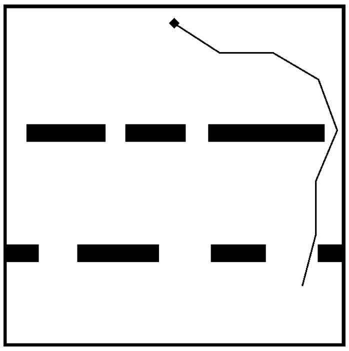


Figure 3.2: Serial robot with 7 links and a fixed base (7 dof)

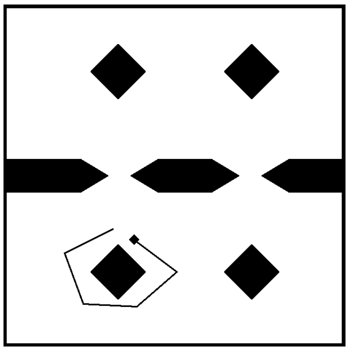


Figure 3.3: Serial robot with 5 links and no fixed point (7 dof)

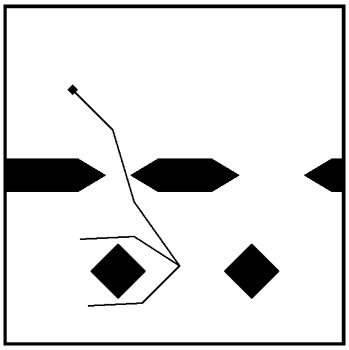


Figure 3.4: Multiple chain (hand-like) robot with a fixed base and 7 links of which the first 3 are extensible (10 dof)

Three are the components that deserve special attention: the local planner, the distance definition and the way collision checking is done. The local planner is called thousands of times to connect configurations and its rate of success affects the density of the produced network. The distance of two nodes should reflect the likelihood that the local planner will fail to connect two nodes and should be compatible with the definition of the local planner. Finally, the collision checker is invoked frequently and its performance is directly reflected in the time spent in the preprocessing phase.

In the next three sections we describe specific techniques that can be substituted for the more general ones described in Chapter 2 and which give better results for planar articulated robots.

3.2.1 The Local Planner for Planar Articulated Robots

Let a and b be any two given configurations that we wish to connect with the local planner. Let also $J_i(a)$ denote the position in workspace of point J_i of the robot, when the latter is in configuration a. The local planner we use constructs a path as follows: it translates at constant relative velocity all the movable J's with an odd index, i.e., all J_{2i+1}'s, along the straight lines in the workspace that connect their positions at configuration a to their positions at configuration b. So, for example, joint J_3 will move on the straight line segment (in the workspace, not in C-space) joining $J_3(a)$ to $J_3(b)$. During this motion the planner adjusts the position of every other joint J_{2i} using the simple inverse kinematic equations of this point relative to J_{2i-1} and J_{2i+1}. A small detail here: the inverse kinematics give two possible positions for J_{2i} for any given position of J_{2i-1} and J_{2i+1} of which we choose the one closest to the previous position of J_{2i}.

Thus, the J_{2i}'s "follow" the motion led by the J_{2i+1}'s. Of course it may be the case that the intermediate nodes J_{2i} are not able to follow the motion dictated by the odd numbered nodes. If all the links are L long, this can happen if the two enclosing nodes have moved further than $2L$ apart. In this case the planner fails.

If q (the number of the last joint) is odd, the position of J_{q+1} is not determined by the above rule; it is then computed by rotating joint J_q at constant revolute velocity relative to the linear velocity of point J_{q+1}. In our first example (Figure 3.2) where $q = 7$ this means that the last parameter of the C-space, θ_7, moves with constant velocity from $\theta_7(a)$ to $\theta_7(b)$, where $\theta_7(a)$ and $\theta_7(b)$ are the values of that parameter at configurations a and b correspondingly. Since θ_7 is an angular degree of freedom we always choose the direction of motion, from $\theta_7(a)$ to $\theta_7(b)$ in the clockwise or counterclockwise direction, that results in the shortest distance covered.

Recall that any local path is discretized into a sequence of configurations for collision checking and that the motion is aborted if a collision occurs (see Section 2.7.1). We have observed that in cases when the above motion does not manage to connect configurations a and b, it nevertheless brings the robot to a configuration b' very close to b. It then pays off to try to connect b' and b with a straight line in C-space and, only after this fails, to declare failure of the local planner to connect a and b. In the following we will refer to the above planner as the *specific local planner* for planar articulated robots. Our experiments have shown that the local paths generated by the specific planner are more likely to be collision-free than those generated by the general planner. On the other hand, the specific planner, though still very fast, is not as fast as the general planner.

3.2.2 Distance Computation

In association with the above local planning technique we use the following distance function $D(\cdot, \cdot)$ in C-space

$$D(a, b) = \left(\sum_{i=1}^{q+1} |J_i(a) - J_i(b)|^2 \right)^{1/2},$$

where $|J_i(a) - J_i(b)|$ is the Euclidean distance between the points $J_i(a)$ and $J_i(b)$. Again, $J_i(a)$, $i = 1, \ldots, q+1$ denotes the position of the point J_i in the workspace, when the robot is at configuration a. It is reasonable to expect that this function reflects well the chances of failure of the specific local planner since it approximates the area swept by the robot along the path induced by this planner.

3.2.3 Collision Checking

The 2D workspace allows for a very fast collision checking technique. In this technique each link of the robot is regarded as a distinct robot with two dof of translation and one dof of rotation. A bitmap representing the 3D configuration space of this robot (in sufficiently high resolution; we used $128 \times 128 \times 128$) is precomputed. The 0's describe the free subset of the C-space and the 1's describe the subset where the link collides with an obstacle. When a configuration is checked for collision, the 3D configuration of each link is computed and tested against its C-space bitmap, which is a constant-time operation. The configuration of a link is particularly fast to compute when the specific local planner is used, since this planner directly provides the coordinates of two points in the link. Note that we need not always create one bitmap for each link of the robot. For example, when all the links are line segments (as in all our examples), a single bitmap can be computed for the shortest link. We then model the longer links as two or more short line segments. As a result, collision checking for a long link may require multiple accesses to the bitmap.

The 3D bitmap for one link can be computed as a collection of 2D bitmaps, each corresponding to a fixed orientation of the link. If the link and the obstacles are modeled as collections of possibly overlapping convex polygons, the construction of a 2D bitmap can be done as follows [Lengyel et al, 90]: First use the algorithm in [Lozano-Pérez, 83] to produce the vertices of the obstacles in the link's C-space. This algorithm takes linear time in the number of vertices of the objects. Then draw and fill the obstacles into the 2D bitmap. On many workstations, this second operation can be done very quickly using raster-scan hardware originally designed to efficiently display filled polygons on graphics terminals. Each 2D bitmap may also be computed using the FFT-based method given in [Kavraki, 93] and described in detail in Chapter 5 of this dissertation. The complexity of this method depends only on the size of the produced bitmap. When using the FFT-based algorithm we can with equal ease compute the parts of the C-obstacle that correspond to links whose shape is not a straight-line segment but *any* other shape. This FFT-based method is also advantageous when the obstacles are originally input as bitmaps. In any case, a 3D bitmap with a size on the order of $128 \times 128 \times 128$ can be computed in a few seconds. The computation of the 3D bitmap(s) needed for collision checking is performed only once, prior to the preprocessing phase of our randomized algorithm.

Clearly, the above collision-checking technique is not yet practical for 3D workspaces, since it requires the generation of 6D bitmaps.

Apart from checking for collision with the obstacles we should also check for collision of the robot with itself. This is done by checking each line segment of the robot against each other, checking first those that are more likely to give rise to a collision. These depend on the geometry of the linkage and are usually the ones furthest from the fixed base, if there is one. The situation gets more complicated if we consider the links to have arbitrary polygonal shape rather than just line segments. Then, the time spent in autocollision checking increases.

3.2.4 Experiments with Articulated Robots in the Plane

Our planner was implemented in the C programming language and we used a DEC Alpha workstation (Model Flamingo) running under DEC OSF/1 for our experiments. This machine is rated at 121.5 SPECmark89. In this section, we analyze in depth the performance of our customized method for 2D robots with some examples. We also show how these results compare with the results obtained when using the general local planner of Section 2.7.1 and not the specific local planner of Section 3.2.1.

Since our algorithm depends on randomness any measure of its performance is a random variable, which may change value from one run to another. To give more reliable results we average all measures of performance over 40 independent runs of our algorithm.

Let us also point out that because of the way path-planning queries are answered (by joining the initial and the final position to the network) the essential measure of the performance of our algorithm at the query phase is whether a given configuration can be easily connected to the network. Searching the network for a path that joins two nodes takes little (and importantly predictable) time compared to connecting a configuration to the network.

That is why in our experimental results we are not concerned with pairs of configurations but only with connecting single configurations to the network. For each robot and workspace in question we have chosen (manually) 8 configurations $C_1, \ldots, C_8$, in a manner that reflects the different parts of the C-space. We refer to these configurations as the test set of the scene. We then measure (by averaging over 40 independent runs of the method) the probability that each of those configurations can be connected to the network produced each time by preprocessing. We consider that we have failed to connect a configuration to the network if we have not succeeded in doing so within 10 seconds for all our examples. Notice that it is important to choose the configurations in the test set manually. For obvious reasons, a random generation similar to the one used during the preprocessing phase tends to produce configurations that are very easily connected to the network. Instead, proceeding manually we can select "interesting" configurations, for example configurations where the robot lies in narrow passages between workspace obstacles.

Our experiments are shown for various values of the parameters N and M, where N is the size of the network before enhancement and $N + M$ is its size after enhancement. So for each such pair of numbers N, M we perform 40 times the following: we run our preprocessing phase, we discard all components but the largest, and then we try to connect $C_1, \ldots, C_8$ to the network (which is thus always connected). We report: the average size of the largest connected component that is kept at the end of preprocessing, the average time spent on preprocessing, and the frequency with which each of the configurations of the test set was connected to that component. All the 40 runs of the method are executed independently of each other. In this way, we believe that we present a quite realistic characterization of the performance of our planner. In particular, we ensure that the results do not reflect just a lucky run, or a bad one.

The same quantities (average size of the largest component, average preprocessing time and success rates for connecting the test set to the network) are also reported after averaging over a different set of 40 runs of the program. These new 40 runs correspond to the previous 40 in the sense that for each pair $N = n$, $M = m$ that we tried in the first set, we are now trying the pair $N = n + m$, $M = 0$. This experiment is done to illustrate the fact that the performance we get using enhancement is better than the performance obtained by just distributing nodes randomly over the C-space.

Finally let us mention the values of the numerical parameters used in our examples. A comprehensive list of these parameters can be found in Section 2.9. We assume that the workspace is the unit square $[0, 1]^2$. $max\_neighbors$ and $rand\_bounce\_length$ have the values 30 and 100 respectively, for all three planar robots that we examined. eps is consistently 0.01. The value of $query\_time$, which provides an upper bound in the time spent to connect each of the configurations of the test set to the networks produced, is 10 seconds. All the lengths of non-extensible links are 0.16 and the length of each extensible link can vary from 0.16 to 0.32. Finally, the parameter $max\_distance$ has the value 0.42 for the fixed-base serial robot, 0.54 for the free serial robot, and 0.47 for the hand-like robot. Notice that only the value of $max\_distance$ changes across the different examples. The above numbers clearly support our claim that the performance of the method does not depend on its numerical parameters in an unstable way. Further tuning of the parameters may be possible, but overall we have found that with little experimentation we can obtain a satisfactory set of values for the parameters involved.

Fixed-base Serial Robot in the Plane (7 dof)

Figure 3.5 shows 8 different configurations of the fixed-base 7-dof robot described in Section 3.2. The workspace consists of two sequences of narrow gates. The fixed base of the robot is shown with a thick black square.

For every row in Table 3.1 we generated 40 independent networks, each with the indicated number of nodes. N is the number of nodes created in the network construction step and M are the nodes added during the network enhancement step. We use $M = N/2$ in our experiments. As in all similar tables that follow, in columns 1-3 we give $N + M$ (total number of nodes), N and M. In column 4 we give the average size (over all 40 runs) of the largest component at the end of preprocessing.

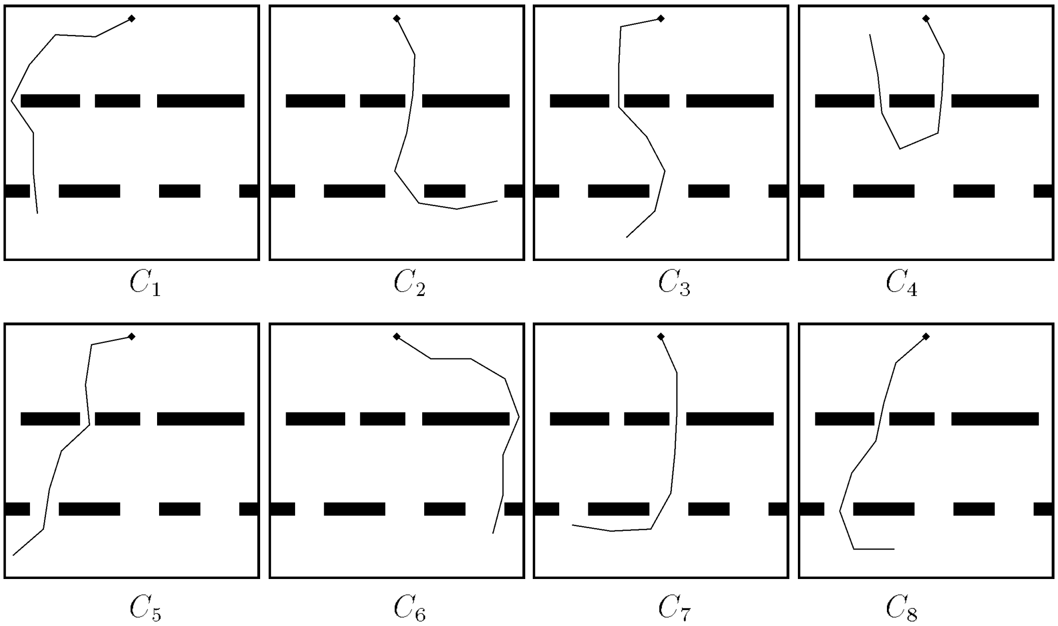


Figure 3.5: Scene 1, with 7-revolute-joint fixed-base robot

In this example we have not used the third step of the preprocessing phase: we have observed that even after short preprocessing times we obtain one major component in the free C-space, thus there is no need for that step. In column 5 we report the average time in seconds for preprocessing. And in columns 6-13 we give an estimate of the probability that each of the configurations $C_1, \ldots, C_8$ of Figure 3.5 was connected to the network. This estimate is nothing more than what fraction of the 40 times we were able to make the connection in less than 10 seconds. Note that we discard all but the largest component when answering path-planning queries.

From Table 3.1 we observe that some configurations like C_1 and C_6 are easily connected to the network even when the number $N + M$ is very small. The success rates are consistently larger than 90% with $N + M = 2700$ which corresponds to about 90 seconds of preprocessing time.

In Table 3.2 we present the results that were obtained by performing a similar set of experiments as in Table 3.1. In each row of Table 3.2 $N + M$ is preserved the same as in the corresponding row of Table 3.1 but now $M = 0$. In other words,

	Initial	Enhanc	Avg	Avg time	Success Rate (%)							
N+M	N	M	size	(sec)	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
1200	800	400	787	29	100.0	30.0	65.0	27.5	62.5	100.0	27.5	55.0
1500	1000	500	1089	40	100.0	45.0	67.5	40.0	77.5	100.0	40.0	72.5
1800	1200	600	1480	52	100.0	77.5	82.5	75.0	92.5	100.0	75.0	85.0
2100	1400	700	1833	64	100.0	85.0	97.5	87.5	97.5	100.0	82.5	97.5
2400	1600	800	2130	78	100.0	87.5	95.0	87.5	95.0	100.0	87.5	95.0
2700	1800	900	2451	90	100.0	97.5	95.0	97.5	92.5	100.0	97.5	90.0
3000	2000	1000	2771	104	100.0	100.0	97.5	97.5	97.5	100.0	97.5	97.5
3300	2200	1100	3091	118	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
3600	2400	1200	3382	132	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
3900	2600	1300	3665	146	100.0	100.0	100.0	97.5	100.0	100.0	97.5	100.0
4500	3000	1500	4284	176	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0

Table 3.1: Success rates with customized planner for Scene 1 (with enhancement)

	Initial	Enhanc	Avg	Avg time	Success Rate (%)							
N+M	N	M	size	(sec)	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
1200	1200	0	697	31	100.0	5.0	40.0	2.5	32.5	100.0	5.0	20.0
1500	1500	0	954	43	100.0	17.5	40.0	17.5	35.0	100.0	17.5	35.0
1800	1800	0	1246	55	100.0	30.0	57.5	30.0	57.5	100.0	27.5	52.5
2100	2100	0	1560	69	100.0	40.0	70.0	42.5	70.0	100.0	42.5	72.5
2400	2400	0	1891	83	100.0	65.0	62.5	62.5	67.5	100.0	62.5	65.0
2700	2700	0	2285	96	100.0	77.5	82.5	77.5	82.5	100.0	80.0	87.5
3000	3000	0	2593	110	100.0	80.0	92.5	77.5	92.5	100.0	77.5	92.5
3300	3300	0	2933	125	100.0	92.5	92.5	90.0	90.0	100.0	90.0	90.0
3600	3600	0	3257	139	100.0	90.0	100.0	92.5	100.0	100.0	90.0	100.0
3900	3900	0	3571	154	100.0	95.0	100.0	95.0	97.5	100.0	95.0	97.5
4500	4500	0	4173	185	100.0	97.5	100.0	97.5	100.0	100.0	97.5	100.0

Table 3.2: Success rates with customized planner for Scene 1 (without enhancement)

enhancement is not performed. Here, the success rates are consistently larger than 90% when $N + M \geq 3300$ which corresponds to preprocessing time of roughly 125 seconds. This preprocessing time is significantly higher than the preprocessing time required to arrive to the same result but with the use of the enhancement step.

In general, the percentages of successful connections are lower in Table 3.2 (no enhancement) than in Table 3.1 (enhancement). The difference shows more clearly when the preprocessing time is small. The version of the method that uses enhancement starts giving high rates of success already from the third row ($N + M = 1800$ or 52 seconds preprocessing time) while the same is not true for the plain (no-enhancement) method. If we are interested in obtaining a solution to a path-planning problem as fast as possible, it is thus better to spend part of the time allocated to the preprocessing phase on the enhancement step rather than spend it completely on the construction step. As mentioned above, the ratio $M = N/2$ gives good results over a wide range of problems. This roughly corresponds to spending 2/3 of the preprocessing time in the network construction step and 1/3 of the preprocessing time in the network enhancement step.

To convey more the intuition behind the enhancement step we show in Figure 3.6 the four major components of the network that are produced during the graph construction step of our algorithm with $N = 1600$ and $M = 0$. To be more precise we show the "projection" of the configurations in each component onto the workspace, that is for each configuration belonging to a component we draw the position of the robot in the workspace. Thus, every two configurations that are in the same picture can be joined to each other with a series of applications of our specific local planner.

We can see that none of the configurations of the major component (Figure 3.6(b)) has gone through any of the doors in its workspace. Each of the other three large components goes through one door of the first level. Several very small connected components are not shown at all. They all had fewer than 10 nodes. The major component here comprises 875 out of a total of 1600 nodes, or 55% of the total.

In Figure 3.7 we see the connected components of the network after we have enhanced our initial network with 800 more nodes. We notice that the three large components of Figure 3.6 have been merged into one comprising 2182 out of a total of 2400 nodes. This is a percentage of 91% of the nodes, much higher than before adding the 800 nodes. The two smaller components shown in Figure 3.7(b) and (c) were not large enough to be visible in the Figure 3.6 before the enhancement. Notice also that the large component goes through all doors of the first level and does quite well at the second level too.

It is clear from Figure 3.6 that we would like to be able to characterize those configurations that are close to the narrow gates. Indeed, adding configurations there instead of anywhere else in the C-space will help the large components of Figure 3.6 merge. This is what we try to do with our enhancement heuristic. We are rather

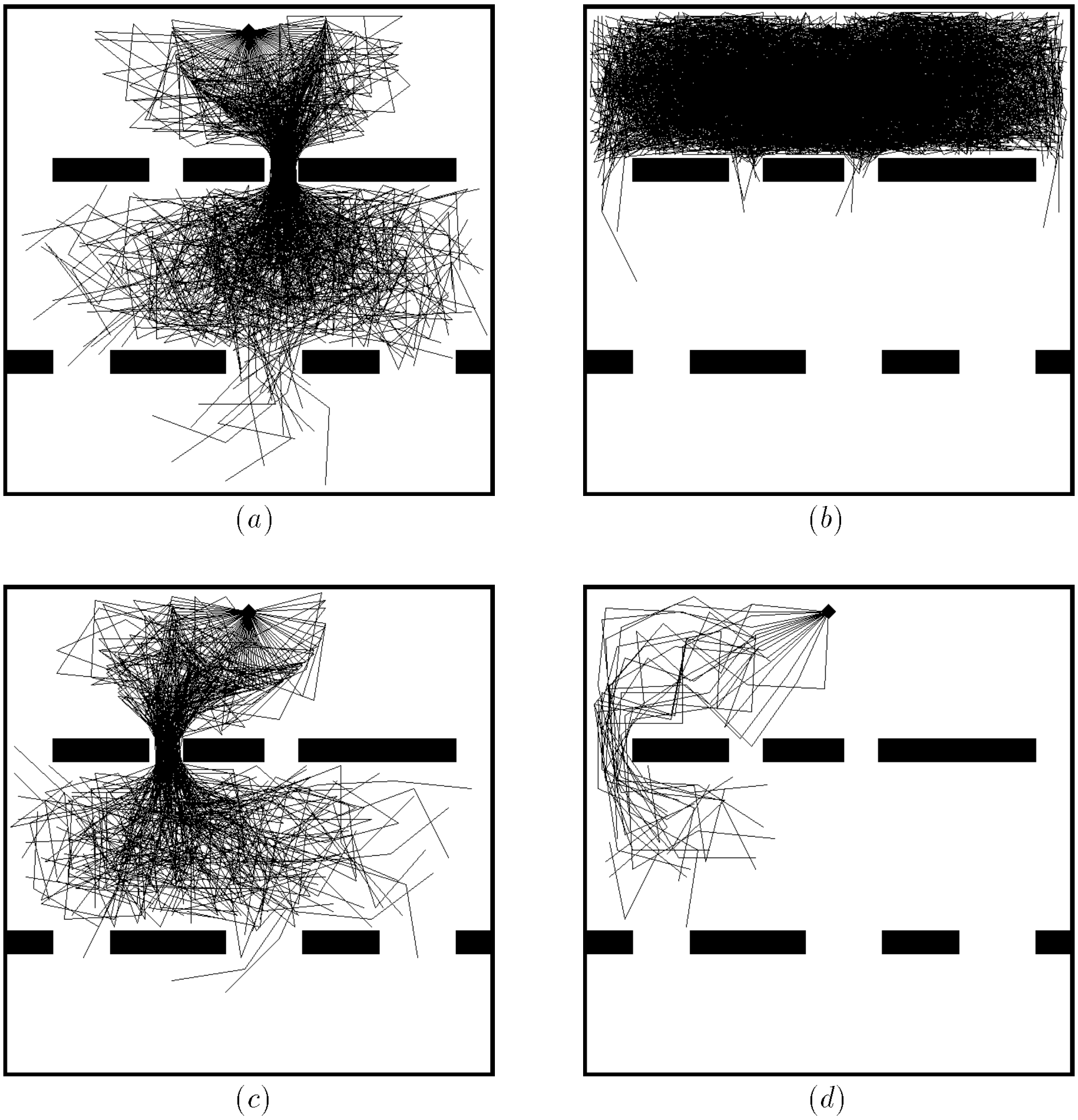


Figure 3.6: Connected components for $N = 1600$, $M = 0$ with sizes 272(a), 875(b), 163(c), 22(d)

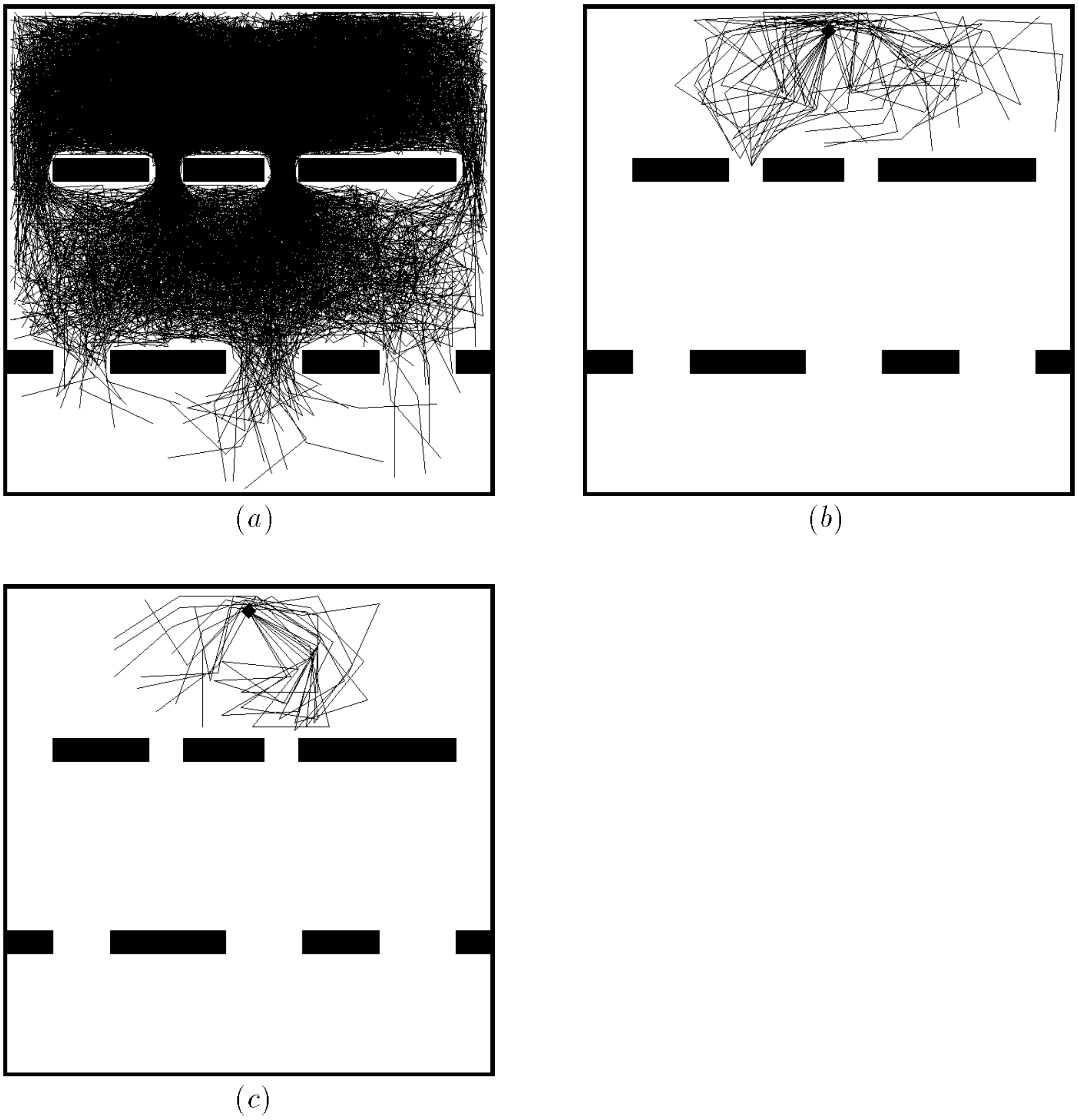


Figure 3.7: Connected components for $N = 1600$, $M = 800$ with sizes 2182(a), 25(b), 15(c)

Initial N	Enhanc M	Comp (num)	Size of components	Prepr (sec)	Time to connect to largest component (sec)							
					C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
800	400	67	683, 207, 149	29	0.00	F	F	F	F	0.63	F	F
1000	500	73	1327	41	0.03	0.00	0.02	0.17	0.00	0.12	0.00	0.02
1200	600	87	1302, 273, 15	51	0.00	0.02	F	0.18	8.37	4.63	0.02	F
1400	700	58	1897, 43, 15	66	0.00	0.02	0.00	0.17	0.00	0.30	0.00	0.02
1600	800	50	2182, 25, 15	77	0.02	0.00	0.02	0.00	0.02	0.05	0.02	0.00
1800	900	53	2515, 39, 16	92	0.00	0.02	0.00	0.00	0.02	0.07	0.00	0.02
2000	1000	36	2841, 31	106	0.00	0.00	0.02	0.00	0.02	0.02	0.02	0.00
2200	1100	43	3157, 50	118	0.00	0.02	0.00	0.02	0.00	0.02	0.00	0.02
2400	1200	41	3370, 66, 61	132	0.00	0.00	0.00	0.02	0.00	0.02	0.02	0.02
2600	1300	44	3675, 93, 38	144	0.02	0.02	0.02	0.00	0.02	0.02	0.02	0.02
3000	1500	56	4156, 110, 86	173	0.00	0.02	0.02	0.02	0.00	0.00	0.02	0.00

Table 3.3: Timings for connecting to the networks for Scene 1 (with enhancement)

Initial N	Enhanc M	Comp (num)	Size of components	Prepr (sec)	Time to connect to largest component (sec)							
					C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
1200	0	144	704, 197, 132	31	0.02	F	F	F	F	0.42	F	F
1500	0	172	873, 246, 176	41	0.02	F	F	F	F	0.00	F	F
1800	0	187	1577	54	0.00	0.02	0.00	0.18	0.02	0.13	0.05	0.02
2100	0	176	1873, 21	69	0.02	0.00	0.00	0.18	0.02	0.30	0.17	0.02
2400	0	169	2172	80	0.02	0.00	0.02	0.13	0.00	0.28	0.00	0.00
2700	0	187	1953, 496, 22	96	0.02	F	0.02	F	0.00	0.15	F	0.00
3000	0	186	2213, 511, 34	109	0.02	F	0.00	F	0.52	0.02	F	0.10
3300	0	191	3037, 29, 20	123	0.02	0.02	0.00	0.02	0.02	0.03	0.00	0.02
3600	0	196	3360	141	0.02	0.02	0.02	0.22	0.02	0.00	0.00	0.00
3900	0	181	3660, 26	152	0.02	0.02	0.00	0.20	0.02	0.00	0.02	0.00
4500	0	217	4159, 49, 47	182	0.02	0.02	0.00	0.25	0.02	0.02	0.00	0.02

Table 3.4: Timings for connecting to the networks for Scene 1 (without enhancement)

successful in this effort. Table 3.1 shows that for $N = 1600$, $M = 800$ we arrive to a connected component which goes through all the doors in the workspace with probability 0.875 (Table 3.1, row 5), while for $N = 2400$, $M = 0$ the corresponding probability is only 0.625 (Table 3.2, row 5).

Overall, both the plain and the enhanced version of our method with the specific local planner give very good results. Moreover, the connection of the configurations in the test set to the networks built during preprocessing is performed very quickly. To show this, we run our algorithm *once* (not 40 times) for the same values of N, M, and we report the time it takes to connect each of $C_1, \ldots, C_8$ to the largest component of the network produced. We perform experiments both with enhancement (Table

3.3) and without enhancement (Table 3.4). An F in the above tables denotes failure to connect the configuration to the corresponding network within 10 seconds.

Column 3 of Tables 3.3 and 3.4 shows the total number of components present in the network at the end of the first two steps of preprocessing. Isolated nodes count for one component here. Column 4 reports the sizes of those components with more than 10 nodes. For example, for $N = 800$, $M = 400$ (Table 3.3) there were 67 components remaining at the end of preprocessing (row 1, column 3). Of them only 3 had more than 10 nodes and the sizes of these components were 683, 207 and 149 nodes respectively (row 1, column 4). Notice that few major components remain at the end of the preprocessing phase. When $N + M$ increases, the largest component has significantly more nodes than any other. Also, for large values of $N+M$, the total number of components remaining at the end of preprocessing decreases. Subsequently, isolated nodes become less and less frequent. Both these facts are an indication that the networks capture well the connectivity of the free C-space for large $N + M$. Also, we observe that when $N + M$ is reasonably large, we get one major connected component from the first two steps of preprocessing. This is the reason why the last step of the preprocessing is never invoked for this robot.

The time needed to connect $C_1, \ldots, C_8$ to the largest component of the networks is shown in columns 6 to 13 in Tables 3.3 and 3.4. Notice that whenever the planner succeeded to connect a configuration to the largest component, it took much less than a second. The only two exceptions in this rule are C_5 and C_6 in row 3 of Table 3.3. The timings of about 8 and 4 seconds shown there are due to the fact that one or more random-bounce walks were performed before the local planner was able to connect C_5 and C_6 to the network (see Section 2.8).

Before ending this section and for comparison purposes let us show how the method performs if we use the generic straight-line local planner instead of the specific local planner which is customized for planar articulated robots. Table 3.5 repeats the experiments in Table 3.1 but uses the straight-line generic local planner and its corresponding distance measure (Section 2.7.1). The fact that customization pays is plain from the comparison of Tables 3.1 and 3.5: for the same preprocessing time

	Initial	Enhanc	Avg	Avg time	Success Rate (%)							
N+M	N	M	size	(sec)	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
1200	800	400	626	24	100.0	12.5	30.0	5.0	35.0	100.0	10.0	27.5
1500	1000	500	841	31	100.0	17.5	22.5	12.5	25.0	100.0	15.0	25.0
1800	1200	600	1131	39	100.0	25.0	45.0	22.5	60.0	100.0	25.0	50.0
2100	1400	700	1418	48	100.0	37.5	65.0	35.0	62.5	100.0	35.0	55.0
2400	1600	800	1764	57	100.0	52.5	65.0	55.0	70.0	100.0	52.5	65.0
2700	1800	900	2128	66	100.0	72.5	80.0	67.5	80.0	100.0	67.5	77.5
3000	2000	1000	2368	76	100.0	70.0	67.5	70.0	70.0	100.0	70.0	70.0
3300	2200	1100	2709	86	100.0	72.5	87.5	70.0	90.0	100.0	75.0	92.5
3600	2400	1200	3112	96	100.0	85.0	97.5	85.0	95.0	100.0	85.0	95.0
3900	2600	1300	3433	106	100.0	90.0	100.0	90.0	100.0	100.0	90.0	97.5
4500	3000	1500	4052	128	100.0	95.0	95.0	95.0	95.0	100.0	95.0	95.0

Table 3.5: Success rates with straight-line planner for Scene 1 (with enhancement)

the success rates for connecting the test set to the network are higher when the customized planner is used (Table 3.1). Nevertheless, the generic local planner does quite well although it does not use any knowledge about the structure of the robot.

Free-flying Serial Robot in the Plane (7 dof)

Figure 3.8 shows 8 different configurations of the free-flying 7-dof robot described in Section 3.2. The thick black square at one end of the robot is not part of the body of the robot; it just helps us to distinguish the one end of the robot from the other.

In Tables 3.6 and 3.7 we show the success rates for connecting each of these 8 configurations to the network, for enhancement and no enhancement respectively. The third step of the preprocessing again has not been used in these calculations, since when $N + M$ is sufficiently large, one major component remains at the end of enhancement. As was the case with the fixed-base serial linkage that we examined before, we also have very good running times for both versions of the algorithm with the enhanced version doing clearly better. At 34 seconds preprocessing time ($N + M = 1800$) the enhanced version gives success rates of 90% or more, while the same rates are achieved after 50 seconds of preprocessing without enhancement ($N + M = 2400$). The difference (in favor of enhancement) shows again more clearly for small values of $N + M$.

In Table 3.8 we give the time required for connecting $C_1, \ldots, C_8$ of Figure 3.8 to the

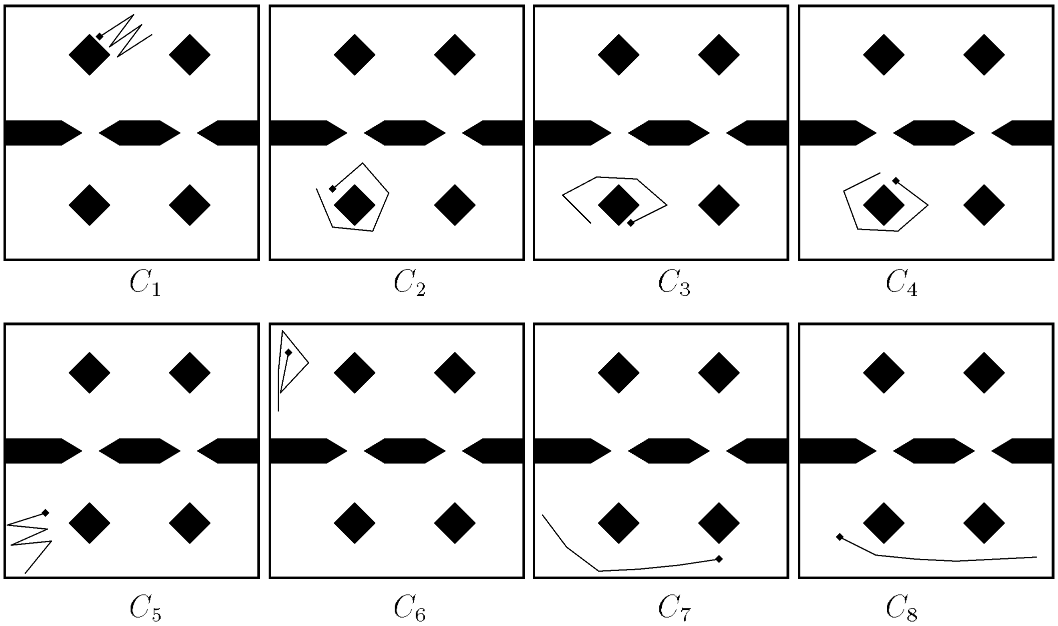


Figure 3.8: Scene 2, with 5-revolute-joint free-base robot

network. We retain only the largest component at the end of the preprocessing phase. The size of that component is reported in column 4 of the above table, together with the sizes of any other components that remain at the end of the network enhancement step and have more than 10 nodes. Again in Table 3.8 an F denotes failure to connect to the network. Subsecond connection to network times are achieved after 48 seconds of preprocessing ($N + M = 2400$). Connection to network timings larger than 1 second denote that the configuration was connected to the network after one or more random-bounce walks.

Note that the long connection to network timings (and failures) of row 5 are due to the fact that some two major components did not get connected in the end, which resulted in the use of the next-level-of-sophistication planner (random-bounce walk followed by the local planner) for answering the queries. This is a randomized algorithm and these occasional failures are to be expected. Our algorithm is to be judged on how often these failures happen, which is answered by Tables 3.6 and 3.7.

N+M	Initial N	Enhanc M	Avg size	Avg time (sec)	Success Rate (%) C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
900	600	300	362	12	90.0	22.5	27.5	35.0	17.5	87.5	25.0	20.0
1200	800	400	635	19	92.5	70.0	57.5	72.5	57.5	87.5	57.5	52.5
1500	1000	500	1010	25	100.0	92.5	80.0	80.0	77.5	95.0	80.0	72.5
1800	1200	600	1521	34	100.0	97.5	97.5	100.0	100.0	100.0	97.5	97.5
2100	1400	700	1829	40	100.0	100.0	100.0	100.0	97.5	100.0	100.0	97.5
2400	1600	800	2161	48	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
2700	1800	900	2469	56	100.0	100.0	100.0	100.0	100.0	97.5	100.0	100.0
3000	2000	1000	2779	65	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
3300	2200	1100	3081	74	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0

Table 3.6: Success rates with customized planner for Scene 2 (with enhancement)

N+M	Initial N	Enhanc M	Avg size	Avg time (sec)	Success Rate (%) C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
900	900	0	328	12	97.5	5.0	10.0	17.5	7.5	97.5	5.0	7.5
1200	1200	0	529	19	100.0	42.5	35.0	30.0	17.5	100.0	20.0	27.5
1500	1500	0	775	26	100.0	60.0	50.0	52.5	40.0	100.0	37.5	45.0
1800	1800	0	1001	34	100.0	60.0	60.0	60.0	50.0	100.0	50.0	47.5
2100	2100	0	1486	42	100.0	80.0	92.5	77.5	75.0	100.0	80.0	80.0
2400	2400	0	1997	50	100.0	100.0	97.5	100.0	97.5	100.0	97.5	97.5
2700	2700	0	2270	58	100.0	100.0	100.0	97.5	95.0	97.5	97.5	95.0
3000	3000	0	2615	68	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
3300	3300	0	2912	76	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0

Table 3.7: Success rates with customized planner for Scene 2 (without enhancement)

Initial N	Enhanc M	Comp (num)	Size of components	Prepr (sec)	Time to connect to largest component (sec) C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
600	300	123	365, 267, 14	12	0.00	1.10	F	F	F	2.92	F	F
800	400	140	476, 402, 16	19	0.02	3.00	2.55	6.20	4.63	4.78	6.43	F
1000	500	124	1231	25	0.02	0.02	0.02	0.02	0.02	0.13	0.00	0.00
1200	600	113	1554, 11	33	0.07	0.05	0.00	0.02	0.33	1.05	0.00	0.02
1400	700	105	1051, 795, 23	40	0.03	1.28	2.10	6.47	F	0.12	0.00	F
1600	800	110	2191	48	0.02	0.02	0.02	0.02	0.10	0.13	0.00	0.02
1800	900	87	2502	56	0.08	0.02	0.00	0.00	0.03	0.00	0.00	0.00
2000	1000	83	2829	65	0.02	0.02	0.02	0.02	0.02	0.07	0.00	0.02
2200	1100	94	3101	74	0.03	0.02	0.02	0.03	0.00	0.22	0.00	0.02

Table 3.8: Timings for connecting to the networks for Scene 2 (with enhancement)

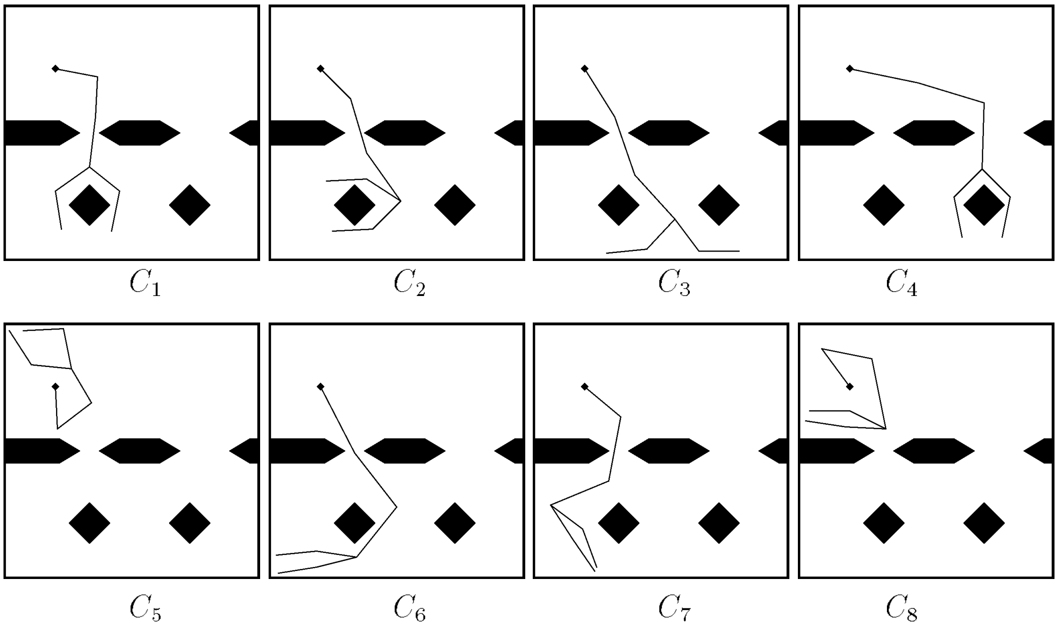


Figure 3.9: Scene 3, with 10-dof hand-like robot

Multiple-chain, Hand-like Robot in the Plane (10 dof)

In Figure 3.9 we show 8 manually chosen configurations of the hand-like robot described in Section 3.2. The first three links of this robot are extensible within limits.

Let us point out that the C-space in this case is disconnected for a trivial reason. Indeed, since we distinguish between the two hands of the robot, say hand 1 and hand 2, two configurations which have the relative positions of the two hands reversed (that is in one configuration hand 1 is to the left of hand 2 and in the other configuration it is vice versa) cannot be connected. It is easy to overcome this problem though by taking care not to ever produce random configurations which have hand 2 to the left of hand 1. This is very easy to accomplish; once the position of hand 1 has been randomly chosen, we restrict the range of the values for the dof of hand 2 in a way that it will fall always to the left, or always to the right, of hand 1.

In Tables 3.9 and 3.10 we give the success probabilities of our planner when tried on the configurations $C_1, \ldots, C_8$ for various values of N and M. Configuration C_6

N+M	Initial N	Enhanc M	Avg size	Avg time (sec)	Success Rate (%) C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
1500	1000	500	1420	87	97.5	85.0	80.0	100.0	100.0	17.5	47.5	100.0
2250	1500	750	2156	145	95.0	92.5	92.5	100.0	100.0	42.5	62.5	100.0
3000	2000	1000	2907	209	100.0	97.5	90.0	100.0	100.0	37.5	77.5	100.0
3750	2500	1250	3652	267	100.0	100.0	87.5	100.0	100.0	37.5	87.5	100.0
4500	3000	1500	4406	334	100.0	100.0	95.0	100.0	100.0	52.5	92.5	100.0
5250	3500	1750	5159	400	100.0	97.5	97.5	100.0	100.0	62.5	92.5	100.0
6000	4000	2000	5913	467	100.0	100.0	100.0	100.0	100.0	75.0	97.5	100.0
6750	4500	2250	6670	535	100.0	100.0	100.0	100.0	100.0	75.0	100.0	100.0
7500	5000	2500	7420	606	100.0	100.0	100.0	100.0	100.0	70.0	100.0	100.0
9000	6000	3000	8927	751	100.0	100.0	100.0	100.0	100.0	80.0	100.0	100.0

Table 3.9: Success rates with customized planner for Scene 3 (with enhancement)

N+M	Initial N	Enhanc M	Avg size	Avg time (sec)	Success Rate (%) C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
1500	1500	0	1389	101	45.0	7.5	2.5	100.0	100.0	0.0	0.0	100.0
2250	2250	0	2119	164	45.0	15.0	2.5	100.0	100.0	0.0	0.0	100.0
3000	3000	0	2852	230	55.0	15.0	7.5	100.0	100.0	0.0	0.0	100.0
3750	3750	0	3580	296	60.0	15.0	12.5	100.0	100.0	7.5	0.0	100.0
4500	4500	0	4328	362	80.0	40.0	37.5	100.0	100.0	5.0	10.0	100.0
5250	5250	0	5071	432	75.0	67.5	52.5	100.0	100.0	7.5	7.5	100.0
6000	6000	0	5805	502	85.0	57.5	55.0	100.0	100.0	22.5	32.5	100.0
6750	6750	0	6579	574	100.0	87.5	85.0	100.0	100.0	22.5	55.0	100.0
7500	7500	0	7313	646	95.0	77.5	67.5	100.0	100.0	25.0	52.5	100.0
9000	9000	0	8813	800	92.5	90.0	90.0	100.0	100.0	40.0	65.0	100.0

Table 3.10: Success rates with customized planner for Scene 3 (without enhancement)

proved to be very difficult to connect here. If we only look down column C_6 we see that the enhancement step has indeed helped a lot. With the exception of C_6, success rates higher than 90% are achieved after 334 seconds of preprocessing when enhancement is done. This is not the case without enhancement.

In Table 3.11 we give the time that is required for connecting each of the 8 considered configurations to the networks produced for different values of $N + M$, when enhancement is performed. Subsecond timings are the norm in all cases but C_6, where the large connection times of 5-6 seconds are due to the fact that a few random-bounce walks were executed before we were able to connect the configuration to the network. The F's persist down the column of C_6, and this was expected from the statistics in

Initial N	Enhanc M	Comp (num)	Size of components	Prepr (sec)	Time to connect to largest component (sec) C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
1000	500	1	1383	171	0.02	0.13	1.25	0.20	0.02	F	F	0.02
1500	750	1	2072	287	0.00	0.03	0.20	0.02	0.07	6.92	F	0.02
2000	1000	1	2742	316	0.00	0.12	0.42	0.00	0.02	13.77	0.02	0.02
2500	1250	1	3402	324	0.02	0.02	0.02	0.00	0.02	5.93	F	0.02
3000	1500	1	4163	377	0.02	1.22	0.02	0.02	0.02	F	0.03	0.02
3500	1750	2	4757, 286	723	0.02	0.03	0.03	0.02	0.02	5.90	0.03	0.02
4000	2000	1	5450	473	0.03	0.02	0.02	0.02	0.02	8.03	0.02	0.17
4500	2250	1	6145	557	0.02	0.03	0.02	0.02	0.02	6.63	0.00	0.02
5000	2500	1	7436	595	0.02	0.02	0.02	0.02	0.03	F	0.02	0.03
6000	3000	1	8960	745	0.05	0.70	0.02	0.03	0.05	0.28	0.02	0.03

Table 3.11: Timings for connecting to the networks for Scene 3 (with enhancement)

Table 3.9.

This robot and workspace is the only one that we studied where it was advantageous to use the third step of the preprocessing, that is the further reduction of connected components with the auxiliary planner (the Randomized Path Planner - RPP - in our case [Barraquand and Latombe, 91]). We noticed that we had to go to very large values of $N+M$ to get a single major component in the free C-space (Table 3.12). For small $N+M$ we had consistently more than one major components at the end of the network enhancement step. Thus, it paid off to dedicate time to a powerful planner. RPP was successful in merging connected components of significant size in reasonable time. In fact the time allocated to the third step of the preprocessing was more than enough for RPP to merge all significant components into one. We remind here that the time allocated to the third step of the preprocessing is equal to the total time of steps 1 and 2 of preprocessing. Because of the calls to the auxiliary planner, the preprocessing times in column 5 of Table 3.11 behave rather irregularly.

In Table 3.12 we show (for the run of our algorithm of Table 3.11) how the preprocessing time is divided among the three steps of preprocessing. In the last column we give the time required by the auxiliary planner to make the connections among the major components. This planner was not needed in the last two rows. Overall, our experience showed that the larger $N+M$ is, the shorter the time it takes for step 3 of preprocessing to finish. There are of course some exceptions in this table

Initial N	Enhanc M	Prepr time (sec)	Step 1 Construction (sec)	Step 2 Enhancement (sec)	Step 3 RPP connection (sec)
1000	500	171	60	28	72
1500	750	287	101	46	138
2000	1000	316	146	61	108
2500	1250	324	189	83	51
3000	1500	377	229	103	43
3500	1750	723	274	122	327
4000	2000	473	324	144	4
4500	2250	557	369	167	20
5000	2500	595	409	185	-
6000	3000	745	507	238	-

Table 3.12: Breakup of preprocessing time for the robot of Scene 3

to the stated fact, but we should not forget that this is a randomized algorithm and its running time may vary a lot from one run to another. It is especially true of the auxiliary planner that we used, RPP, that its running time may vary a lot and this is one of RPP's greatest disadvantages. Our algorithm is much more stable in this respect.

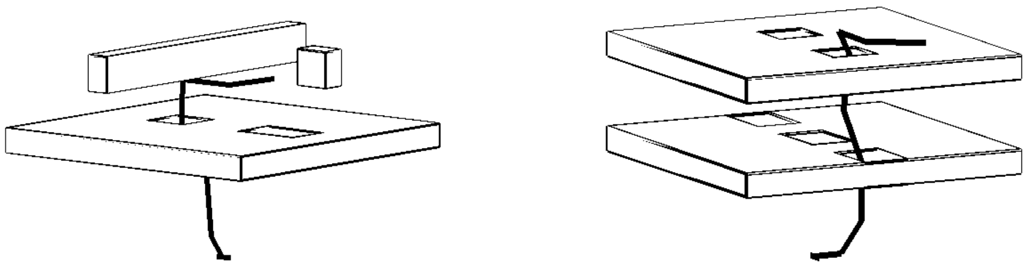
3.3 Articulated Robots in Space

We consider now serial articulated robots in space with spherical joints. The links of these robots are again line segments. To parametrize the C-space we need to describe the orientation of each link of the robots in space. This requires two angles θ and ϕ as described in page 8.

We consider two such robots:

1. The robot shown in Figure 3.10(a) has a fixed base and 6 spherical joints for a total of 12 dof. We draw a small rectangle at the base of this robot.

2. The articulated robot in Figure 3.10(b) has 8 links (16 dof) and moves in a very constrained environment. This is the example with the highest number of dof presented in this dissertation.



(a) (b)

Figure 3.10: (a) Fixed-base serial robot in space with 6 spherical joints (12 dof) and (b) fixed-base serial robot with 8 spherical joints (16 dof)

As in the planar case, we do not allow any two consecutive links to form an angle smaller than 10 degrees. This simulates mechanical stops. We describe now how the method of Chapter 2 can be customized to articulated robots in space.

3.3.1 The Local Planner for Articulated Robots in Space

Let a and b be two configurations of the robot that we wish to connect. A planner that is similar to the one used for planar robots (see Subsection 3.2.1) and gives better results than the straight-line generic planner works as follows: it tries to move the odd-numbered joints J_{2i+1} of the robot along the straight-line segment in workspace that connects their positions in configurations a and b, that is $J_{2i+1}(a)$ and $J_{2i+1}(b)$, and with constant speed. If q (the number of the last joint) is odd, the position of J_{q+1} is not determined by the above rule. We treat this problem as we did in the planar case by letting the last (θ, ϕ) pair move in a straight line in (θ, ϕ)-space to their final values.

Unlike the planar case the motion of the odd-numbered nodes does not uniquely determine the motion of the intermediate even-numbered nodes. Indeed, for any fixed position in space of J_1 and J_3 the joint J_2 can be anywhere on a circle with center the midpoint of J_1J_3. This circle lies on a plane perpendicular to J_1J_3, and has radius

$$r = \left(L^2 - \frac{1}{4}|J_1J_3|^2\right)^{1/2}, \tag{3.1}$$

where L is the common length of each segment.

So we proceed as follows: We start moving the odd-numbered nodes J_{2i+1} $i = 0, \ldots, \lfloor q/2 \rfloor$, from their initial positions $J_{2i+1}(a)$ to their final positions $J_{2i+1}(b)$, along the line segments $J_{2i+1}(a)J_{2i+1}(b)$ and by making a (safe) small step at a time. At each such step after having determined the new position of all J_{2i+1}'s the position of the even-numbered nodes J_{2i} $i = 1, \ldots, \lfloor q/2 \rfloor$, is determined by projecting the old position onto the new *locus* of the point J_{2i} (that is the circle determined by the new positions of the even-numbered nodes and (3.1)). It is not difficult to carry this projection. We first project the old position of J_{2i} onto the plane perpendicular to the (new) line segment $J_{2i-1}J_{2i+1}$ at the middle of it, and then we scale the resulting vector to length r. It is easy to show that the point obtained in this manner is the point of the circle which is closest to the previous position of J_{2i}. These calculations can be carried out very fast. However, the resulting local planner is not as fast as the generic straight-line planner.

3.3.2 Distance Computation

The distance in this case is given by exactly the same formula as in the planar case, that is

$$D(a,b) = \left(\sum_{i=1}^{q+1} |J_i(a) - J_i(b)|^2\right)^{1/2}.$$

3.3.3 Collision Checking

Collision checking in a three-dimensional workspace is expensive and the collision checking function should be written very carefully. We chose the following simple solution. Our workspace is modeled as a collection of two-dimensional rectangles in arbitrary position in space. We wrote a fast routine that checks for collision of a line segment and a 2D-rectangle in space and we apply this routine to all the links of the robots used. For fixed-base robots we check the links starting from the one further from the base to increase our chances of detecting a collision sooner rather than later. If the robot has a fixed base which is outside the obstacles, one of its links *must* intersect some of the rectangles that represent the obstacles, whenever the robot collides with the obstacles. This is not the case with a free-flying robot. Then the whole robot might be enclosed in a 3D-box obstacle, that we have modeled as a collection of 6 faces, without intersecting with any face. Of course this would be a collision with the solid obstacle. This problem can be avoided in many ways, say by testing a single joint of the robot for containment in all workspace obstacles along with the previous test of line segments intersecting the faces.

A large amount of work has been conducted for developing efficient collision checkers for 3D workspaces and the use of more sophisticated collision checking will be necessary when the workspace involves more complicated obstacles. In particular, we believe that collision checkers which generate successive approximations of the objects and the robot will be beneficial for this planner (for example see [Quinlan, 94]). The reason is that many collisions can already be detected with a coarse approximation of the robot and the obstacles, thus avoiding explicit checking of all the links of the robot against all the obstacles.

3.3.4 Experiments with Articulated Robots in Space

The tables that we present here have the same format as in the planar case. For each robot we manually define a test set and report how successful we are in connecting each of the configurations in the test set to the networks produced for different values of N and M. Success rates were measured over 40 independent runs of the method. Again for comparison purposes we present results obtained with the use of the network enhancement step and without it. The third step of the preprocessing has not been used in the 3D examples. The lengths of the links of the robot are again 0.16 and the workspace is $[0,1]^3$. The experiments were performed on the same machine as in the planar case, a DEC Alpha, and 10 seconds were spent in trying to connect a configuration to the largest component of the network produced by preprocessing before declaring failure. As far as the other numerical parameters of the method are concerned: *rand_bounce_length* and *max_neighbors* have the values 100 and 30 respectively and *eps* is set to 0.01. The value of *max_distance* is 0.42 for the 12-dof robot, while for the 16-dof robot it is 0.54.

Fixed-base Serial Articulated Robot in Space (12 dof)

Figure 3.11 shows the 8 configurations of the 12-dof robot that we try for connection to the networks produced for different values of N and M.

In Tables 3.13 and 3.14 we report the results of our experiments with enhancement and without enhancement respectively. From Table 3.13 we infer that 228 seconds of preprocessing time are necessary for connecting the configurations of the test with a 100% success rate with the exception of configuration C_7. Configuration C_7 turned out to be a difficult case and the enhanced version does rather better on this than the plain version of our algorithm. With enhancement we need 405 seconds to connect this configuration to the produced network 90% of the time (Table 3.13, row 6). Without enhancement the corresponding preprocessing time is 620 seconds (Table 3.14, row 8). Otherwise the two versions give rather comparable results and this is probably due to the fact that the example turned out to be an easy problem, despite the 12 dof. We have generally observed that the superiority of the enhanced version shows up mostly in difficult examples.

In Table 3.15 we show the time required to connect each of the 8 configurations examined to the networks produced. Subsecond connection times prevail after a preprocessing of 236 seconds. Notice also that one major component that comprises almost all the generated nodes is formed after a preprocessing time of 156 seconds. The size of that component is shown in column 4 of Table 3.15. The absence of smaller components (all other components had less than 10 nodes) proves that the example was an easy one for the method.

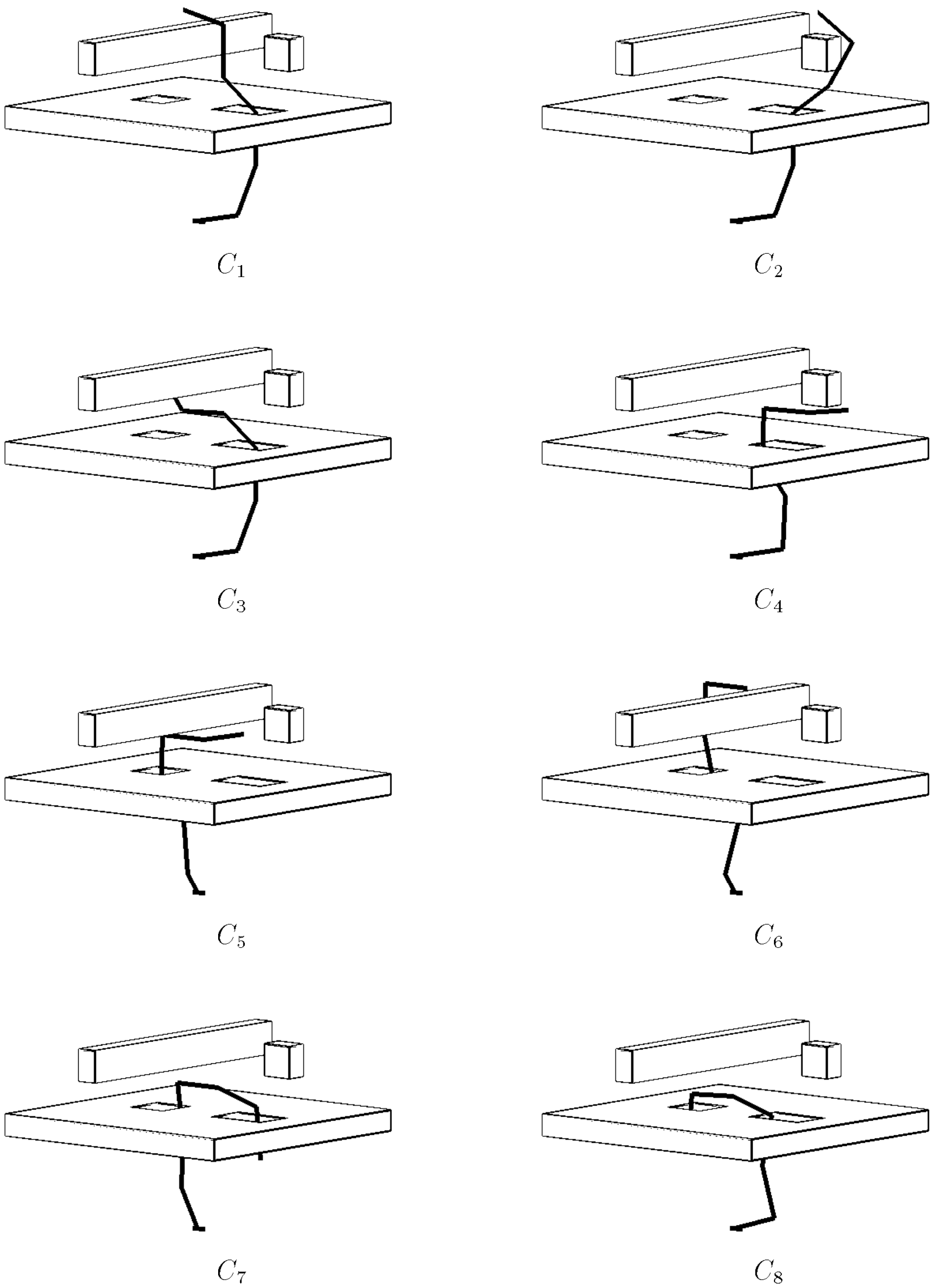


Figure 3.11: Scene 4, with a fixed-base robot which has 6 spherical joints (12 dof)

N+M	Initial N	Enhanc M	Avg size	Avg time (sec)	Success Rate (%) C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
1200	800	400	1104	49	95.0	87.5	87.5	87.5	60.0	75.0	7.5	75.0
1800	1200	600	1733	95	97.5	95.0	100.0	97.5	77.5	100.0	35.0	92.5
2400	1600	800	2342	156	100.0	100.0	100.0	100.0	90.0	95.0	55.0	100.0
3000	2000	1000	2962	228	100.0	100.0	100.0	100.0	100.0	100.0	70.0	100.0
3600	2400	1200	3568	312	100.0	100.0	100.0	100.0	100.0	100.0	82.5	100.0
4200	2800	1400	4174	405	100.0	100.0	100.0	100.0	100.0	100.0	90.0	100.0
4800	3200	1600	4776	504	100.0	100.0	100.0	100.0	100.0	100.0	95.0	100.0
5400	3600	1800	5376	622	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
6000	4000	2000	5988	734	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0

Table 3.13: Success rates with customized planner for Scene 4 (with enhancement)

N+M	Initial N	Enhanc M	Avg size	Avg time (sec)	Success Rate (%) C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
1200	1200	0	1123	39	92.5	87.5	92.5	85.0	50.0	75.0	5.0	77.5
1800	1800	0	1738	87	100.0	100.0	97.5	100.0	92.5	97.5	17.5	100.0
2400	2400	0	2349	153	100.0	100.0	100.0	100.0	97.5	100.0	40.0	97.5
3000	3000	0	2958	228	100.0	100.0	100.0	100.0	97.5	100.0	57.5	100.0
3600	3600	0	3563	316	100.0	100.0	100.0	100.0	100.0	100.0	77.5	100.0
4200	4200	0	4172	411	100.0	100.0	100.0	100.0	100.0	100.0	82.5	100.0
4800	4800	0	4771	512	100.0	100.0	100.0	100.0	100.0	100.0	77.5	100.0
5400	5400	0	5373	620	100.0	100.0	100.0	100.0	100.0	100.0	90.0	100.0
6000	6000	0	5977	728	100.0	100.0	100.0	100.0	100.0	100.0	97.5	100.0

Table 3.14: Success rates with customized planner for Scene 4 (without enhancement)

Initial N	Enhanc M	Comp (num)	Size of components	Prepr (sec)	Time to connect to largest component (sec) C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
800	400	31	1109, 22, 11	49	7.68	3.38	0.80	F	6.58	12.03	F	4.87
1200	600	15	1722, 15	98	0.02	0.38	0.00	0.02	0.28	1.98	F	0.02
1600	800	15	2336	156	0.00	0.13	1.95	0.03	0.03	3.55	F	1.38
2000	1000	10	2954	236	0.00	0.02	0.50	0.03	0.03	0.02	2.07	3.02
2400	1200	5	3577	323	0.02	0.08	0.17	0.07	0.00	0.02	0.02	2.88
2800	1400	4	4183	410	0.02	3.50	0.03	0.02	0.02	0.07	0.03	0.02
3200	1600	2	4795	510	0.02	3.07	0.13	0.07	0.02	0.02	0.03	3.10
3600	1800	2	5393	613	0.02	0.02	3.00	0.03	0.02	0.02	0.32	4.07
4000	2000	2	5989	726	0.02	0.05	0.03	0.07	0.03	0.02	7.18	0.15

Table 3.15: Timings for connecting to the networks for Scene 4 (with enhancement)

Fixed-base Serial Articulated Robot in Space (16 dof)

Figure 3.12 shows one of the most difficult examples we have run so far. The robot involved has 16 dof and it moves among narrow gates. The x and y dimensions of gates in the workspace of the robot are less than half the length of each link of this robot.

Tables 3.16 and 3.17 contrast the performance of the planning method with enhancement and without enhancement respectively. Notice from Table 3.16 that configurations C_1, C_2 and C_8 proved difficult to connect to the networks produced by preprocessing. However, after 478 seconds of preprocessing time the success rates for connecting all of $C_1, \ldots, C_8$ to the network produced are at least 90% (Table 3.16, row 5). This nice behavior is not observed when the enhancement step of the preprocessing is not used. In fact, the success rates for a preprocessing time of around 500 seconds are only 60%. Last row of Table 3.17 shows that even for large values of N, which amounts to a preprocessing time of nearly 1000 seconds, the version of the method without enhancement does not perform satisfactorily: for C_1 and C_2 the success rates for connecting them to the largest component of the network produced are 87.5% and 85.0% respectively, while for C_8 the corresponding success rate is only 72.5%.

In Table 3.18 we report the time needed to connect each of $C_1, \ldots, C_8$ to the network when $N + M$ increases and enhancement is performed. The size of the largest component is again reported in column 4 of the above table and an F denotes failure to connect to the network within 10 seconds. Subsecond times for connecting $C_1, \ldots, C_8$ are difficult to achieve in this example unless $N + M$ is large. Typically for this scene few random-bounce walks were performed before each of the $C_1, \ldots, C_8$ got connected to the network with the specific local planner. However, the timings reported for connection to the networks are very low: most of them range between 2 and 5 seconds.

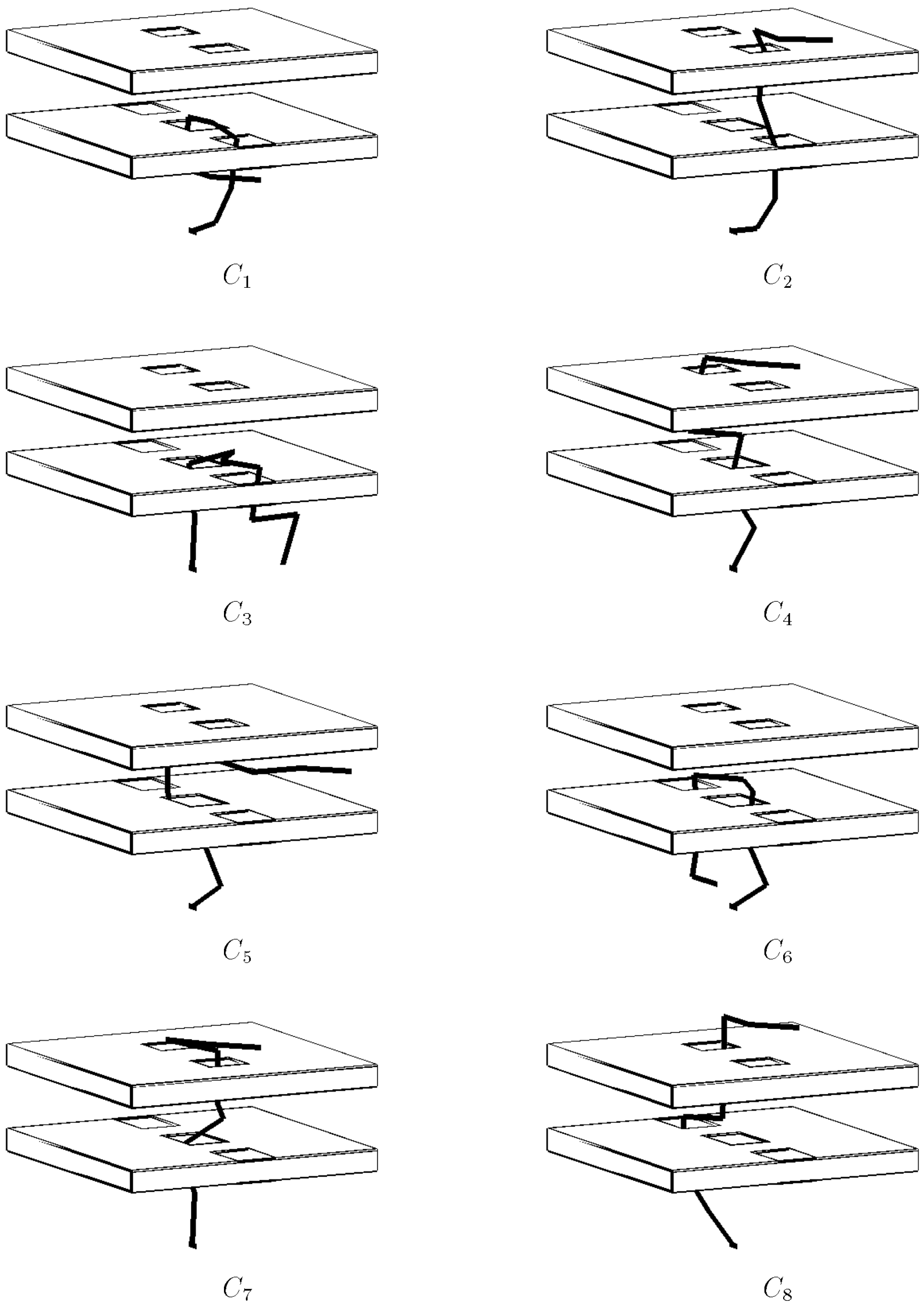


Figure 3.12: Scene 5, with a fixed-base robot which has 8 spherical joints (16 dof)

N+M	Initial N	Enhanc M	Avg size	Avg time (sec)	Success Rate (%) C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
1200	800	400	1088	72	77.5	62.5	62.5	70.0	97.5	97.5	57.5	67.5
1800	1200	600	1701	144	72.5	67.5	95.0	97.5	100.0	100.0	85.0	67.5
2400	1600	800	2316	241	85.0	77.5	100.0	100.0	100.0	100.0	87.5	70.0
3000	2000	1000	2917	350	85.0	80.0	100.0	100.0	100.0	100.0	92.5	80.0
3600	2400	1200	3522	478	97.5	90.0	100.0	100.0	100.0	100.0	100.0	90.0
4200	2800	1400	4123	620	95.0	92.5	100.0	100.0	100.0	100.0	100.0	87.5
4800	3200	1600	4725	769	97.5	90.0	100.0	100.0	100.0	100.0	97.5	92.5
5400	3600	1800	5331	931	97.5	97.5	100.0	100.0	100.0	100.0	100.0	90.0

Table 3.16: Success rates with customized planner for Scene 5 (with enhancement)

N+M	Initial N	Enhanc M	Avg size	Avg time (sec)	Success Rate (%) C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
1200	1200	0	1043	66	67.5	52.5	42.5	52.5	97.5	95.0	35.0	45.0
1800	1800	0	1642	145	75.0	42.5	67.5	70.0	100.0	90.0	37.5	65.0
2400	2400	0	2235	251	77.5	55.0	87.5	85.0	100.0	97.5	60.0	62.5
3000	3000	0	2844	375	60.0	60.0	97.5	90.0	100.0	97.5	87.5	67.5
3600	3600	0	3447	513	80.0	65.0	97.5	97.5	100.0	100.0	82.5	60.0
4200	4200	0	4054	663	87.5	65.0	97.5	100.0	100.0	100.0	95.0	67.5
4800	4800	0	4648	824	77.5	70.0	100.0	100.0	100.0	100.0	95.0	70.0
5400	5400	0	5252	990	87.5	85.0	100.0	100.0	100.0	100.0	95.0	72.5

Table 3.17: Success rates with customized planner for Scene 5 (without enhancement)

Initial N	Enhanc M	Comp (num)	Size of components	Prepr (sec)	Time to connect to largest component (sec) C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
800	400	37	1086, 27, 18	73	5.88	F	F	F	6.27	F	F	F
1200	600	51	1653, 18, 16	147	2.65	5.52	3.70	2.45	5.50	2.68	F	4.02
1600	800	26	2333	250	5.20	F	3.05	6.30	4.13	1.72	0.03	F
2000	1000	21	2907	360	5.48	F	0.42	9.95	0.02	1.40	2.40	2.90
2400	1200	22	3537	476	6.08	4.5	0.03	2.52	0.05	2.62	3.20	5.60
2800	1400	15	4128	628	4.43	2.65	0.07	2.05	0.13	0.10	4.42	4.07
3200	1600	16	4736	780	4.72	2.87	2.78	0.03	0.07	0.50	2.22	5.80
3600	1800	12	5354	961	2.77	3.25	1.03	0.05	0.10	0.03	0.02	4.30

Table 3.18: Timings for connecting to the networks for Scene 5 (with enhancement)

3.4 Some Remarks

The experimental results shown in this chapter are very promising. For the robots considered our method can efficiently solve problems which are beyond the capabilities of other existing techniques. For example, for planar articulated robots with many dof, the customized implementation of Section 3.2 is much more consistent than the Randomized Path Planner of [Barraquand and Latombe, 91]. The latter planner although it can be very fast on some difficult problems, it may also take prohibitive time on some others. We have not observed such disparity with our randomized network method. Notice however, that the problems considered in this chapter involve many narrow gates in the workspace of the robot and are typically the problems that cannot be handled satisfactorily by potential-field techniques like the Randomized Path Planner.

Overall, we have found the method quite reliable and easy to use. When preprocessing time increases, a better network is always obtained. The incremental nature of the technique permits the gradual construction of a network that captures sufficiently well the connectivity of the free C-space even for difficult examples. Also, it helps to select easily some of the numerical parameters of the method ($N, M, query\_time$) for a given class of problems.

We would like to emphasize that the problems described in the previous sections of this chapter are quite difficult. We change below some of the workspace features of these examples (i.e., make the gates in the workspace larger) to reveal the difficulty of these problems. This short experiment also shows how well our method performs in easier examples. All the percentages reported are obtained over 40 independent runs of the method, as in Sections 3.2 and 3.3. Figure 3.13 shows the 10-dof hand-like robot described in Section 3.2.4. The robot now moves in an environment where the left gate of its workspace wall is as wide as the right gate. We used the customized version of our randomized method with enhancement, and obtained the following result: only 80 seconds of preprocessing time are required for success rates of at least 97.5% for the configurations $C_1, \ldots, C_8$ of Figure 3.9. This preprocessing time is very

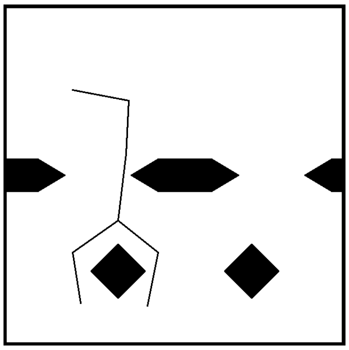


Figure 3.13: 10-dof hand-like robot moving in a less constrained environment

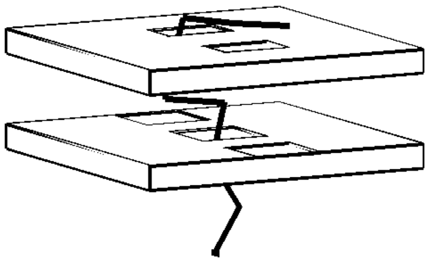


Figure 3.14: 16-dof fixed-base robot moving in an environment with wide gates

low compared to the 300 to 400 seconds reported in Table 3.9.

As another example consider the 16-dof robot of Section 3.3.4. Suppose we make the gates in its workspace one and a half times wider than before, as shown in Figure 3.14. Then the preprocessing time needed to arrive to success rates that are larger than 97.5% is only 72 seconds. The test set consists of the configurations $C_1, \ldots, C_8$ shown in Figure 3.12. Compare the above result with the preprocessing times for the workspace with the narrow gates in Table 3.16, where 478 seconds were needed for comparable success rates. It is indeed true that the method performs remarkably well in easier scenes, despite the large number of dof of the robots involved.

An important question is how our method scales up when we consider scenes with more complicated geometry, since the cost of collision checking can be expected to increase. First, let us note that in 2D workspaces the effect is likely to be limited if the bitmap collision-checking technique of Chapter 5 is used. Indeed, once bitmaps have been precomputed, collision checking is a constant-time operation; and the cost of computing bitmaps using the FFT-based technique of Chapter 5 only depends on the resolution (i.e., the size) of these bitmaps. However, more complicated geometry may require increasing the bitmap resolution in order to represent geometric details with the desired accuracy. With 3D workspaces the situation is completely different, since we can no longer use the bitmap technique. Our experiments of Section 3.3 show that the higher cost of collision checking mainly increases the duration of the preprocessing phase. It also increases the time needed in the query phase, since collision checking is needed to connect the start and goal configurations to the network.

For more complicated shapes of robots and obstacles than the ones shown here, the use of an iterative collision checker, like the one in [Quinlan, 94], will be advantageous. This collision checker considers successive approximations of the objects and its running time, on the average, does not depend much on the geometric complexity of the scenes. It is possible to simplify the geometry of the space during a large part of the preprocessing. In the network construction step, we could use a local planner that is simple not only in the paths it generates but also in the geometry it considers (as long as this geometry is conservative). Such a planner will create a number of connected components at the end of the network construction step. Then a planner that considers the exact geometry of the space can be used to obtain connections among these components. RPP is another planner that heavily relies on collision checking. For long RPP was run on geometrically simple problems; but, recently, it was used to automatically animate graphic 3D scenes of complex geometry [Koga et al, 94] using the above iterative collision checker. No dramatic slowdown was observed in the time taken by the planner. Finally, we should emphasize that the potentially large cost of collision checking can be offset by a massive parallelization of the method which is definitely possible, as explained in the end of Chapter 2.

Chapter 4

Theoretical Analysis of the Performance of the Method

4.1 Introduction

In this chapter we attempt to analyze formally the performance of our algorithm. This is a very difficult task. Like any method employed in practice our method uses heuristics, which may differ from case to case, and takes shortcuts to achieve better performance. It is thus necessary to work on a simplified and concrete case and rid the method from any elements that are either not essential or are too difficult to take into account.

We arrive at the following simplified model of our method. In this model the C-space is assumed to be two-dimensional. Later in this chapter, we show that our analysis can be carried over to higher C-space dimensions without any complications.

The parameters of our model are given below:

- The Free Space
  An arbitrary open subset $\mathcal{F}$ of the unit square $\mathcal{W} = [0,1]^2$.

- The Robot

A point which is free to move in $\mathcal{F}$.

- The Local Planner

  The local planner takes the robot from point x to point y along a straight line. It succeeds if the straight line segment xy is contained in $\mathcal{F}$.

- The Collection of Random Configurations

  A collection of N independent points uniformly distributed over $\mathcal{F}$.

In other words we throw N independent random points in $\mathcal{F}$ and connect any two of them that can be connected by a free straight line. A graph G with possibly more than one connected components results in this fashion. To solve any planning problem, that is to go from any point a to any point b, we try to connect both a and b to two nodes in the same connected component of G using straight lines. Our algorithm succeeds if and only if this is possible.

Having made the situation as clear as it can be, our purpose becomes to analyze the probability of success of our algorithm as a function of all the relevant parameters. For this we take any two, but fixed, points $a, b \in \mathcal{F}$, for which we assume that they can be connected via a continuous path

$$p \;:\; [0,1] \longmapsto \mathcal{F}, \text{ where } p(0) = a \text{ and } p(1) = b.$$

Let also $\mathcal{O}$ be the complement of $\mathcal{F}$ in $\mathcal{W}$ (the C-obstacle) and for any $x \in \mathcal{W}$ write $r(x)$ for the Euclidean distance of x to $\mathcal{O}$, that is

$$r(x) = \min_{y \in \mathcal{O}} |x - y|,$$

where $|x - y|$ is the Euclidean distance of the points x and y of the plane.

We shall give some upper bound for the probability of failure of our algorithm, which involves the number N of random points, the function $r(p(t))$ for $t \in [0,1]$, as well as the length L of p. Our bound will hold for any path p that joins a and b. The dependence on $r(p(t))$ and L is to be expected. If we have two points for which any connecting path has small $r(p(t))$, this intuitively means that the problem is difficult since we have to go close to the C-obstacle. Similarly, if any connecting path is long, it gets more difficult to find it, since a larger number of relevant intermediate configurations must be present in our random sample of $\mathcal{F}$.

Unfortunately the bounds computed here are hard to use *a priori* since they depend on the properties of the postulated connecting path $p(t)$ from a to b, which are difficult to measure. But they give us some idea of the dependence of the performance on N, and allow some restricted questions to be answered.

We first introduce some notation. For any vector $x = (x_1, x_2)$ we write $|x| = \sqrt{x_1^2 + x_2^2}$ for its Euclidean length. We also write

$$B_R(x) = \{y \in \mathcal{W} : |x - y| \leq R\}$$

for the Euclidean ball with center x and radius R. If two points x and y belong to a curve $C : [0,1] \longmapsto \mathcal{W}$ we denote by $d(x,y)$ the arc-length from x to y along the curve C. We also denote by $|\mathcal{F}|$ the area of the free space.

4.2 The Failure Probability for Paths Uniformly Away from the Obstacles

In this section we give some simple upper bounds on the failure probability when connecting pairs of points a and b. It is assumed that a and b can be connected by *some* path $p : [0,1] \longmapsto \mathcal{F}$ which keeps uniformly away from the obstacles, that is all its points are at least a certain distance away from the C-obstacles. The key idea is that of covering the path with few balls which overlap to a certain degree. The parameters mentioned in the theorem below are drawn in Figure 4.1.

Theorem 4.1 *Let $p : [0,1] \longmapsto \mathcal{F}$ be a path of length L, $p(0) = a$, $p(1) = b$, and let $R = \min_{0 \leq t \leq 1} r(p(t))$ be the distance of the path to the obstacles. Then the probability that our algorithm will fail to connect the points a and b is at most*

$$\frac{2L}{R}\left(1 - \frac{\pi R^2}{4|\mathcal{F}|}\right)^N . \tag{4.1}$$

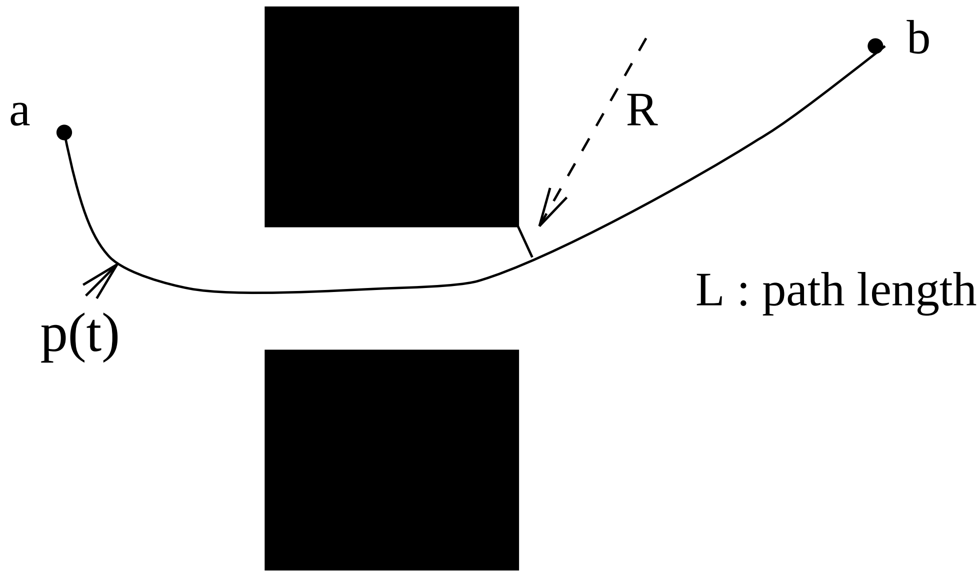


Figure 4.1: The parameters involved in the theorem

Proof: Let $n = 2L/R$. Then we can find points $x_0 = a, x_1, \ldots, x_n = b$ on the curve p, for which $d(x_j, x_{j+1}) \leq R/2$, for all j. Notice that for each j

$$B_{R/2}(x_j) \subseteq B_R(x_{j+1}). \tag{4.2}$$

This is a direct consequence of the triangle inequality and the inequality $|x - y| \leq d(x, y)$.

Assume now that $c \in B_{R/2}(x_j)$ and $d \in B_{R/2}(x_{j+1})$. Observe then that also $c \in B_R(x_{j+1})$ because of (4.2), which implies that the straight line segment $\overline{cd}$ is free, since both c and d are contained in the same free ball $B_R(x_{j+1})$. An illustration of this argument is given in Figure 4.2.

Let now $q_1, \ldots, q_N$ be the random points that our algorithm produced. According to the preceding observation, it is enough that we have at least one of the q_k's, $k = 1, \ldots, N$ in each ball $B_{R/2}(x_j)$, for our algorithm to succeed to connect the points a and b. Since the q_k's are independent and uniformly distributed over $\mathcal{F}$, we conclude that the probability that the ball $B_{R/2}(x_j)$ contains none of the q_k's is equal to $(1 - |B_{R/2}|/|\mathcal{F}|)^N$, where $|B_{R/2}|$ is the area of the ball of radius $R/2$. Here we use the fact that we have thrown N *independent* points in $\mathcal{F}$.

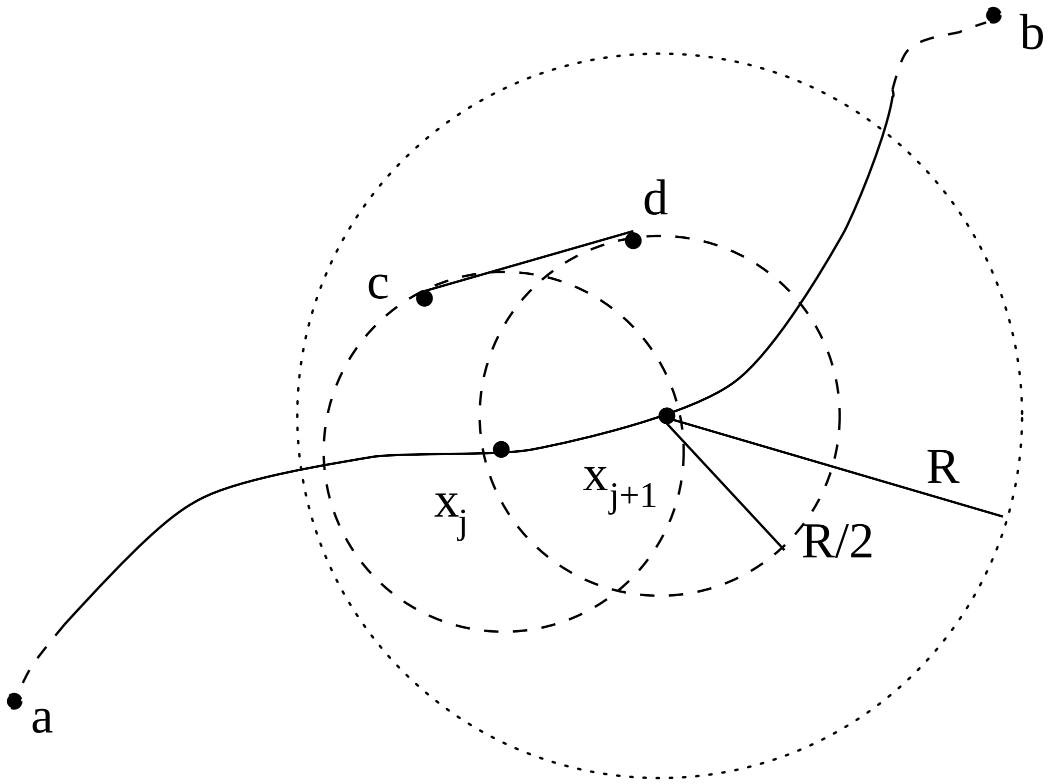


Figure 4.2: The proof of the theorem

Thus,

$$\begin{aligned}\Pr[\text{FAILURE}] &\leq \Pr[\text{Some ball is empty}] \\ &\leq \sum_{j=1}^{n} \Pr[\text{The } j\text{-th ball is empty}] \\ &= n\left(1 - \frac{|B_{R/2}|}{|\mathcal{F}|}\right)^{N}. \end{aligned} \tag{4.3}$$

But since in two dimensions the area of the ball with radius $R/2$ is $\pi R^2/4$, the above relation becomes

$$\Pr[\text{FAILURE}] \leq \frac{2L}{R}\left(1 - \frac{\pi R^2}{4|\mathcal{F}|}\right)^{N},$$

which concludes the proof of the theorem. □

Let us note here that the same analysis applies in C-spaces of higher dimension. If d is the dimension of the C-space, the balls covering any path p are also d-dimensional. From (4.3) the estimate for the probability of failure becomes

$$\Pr[\text{FAILURE}] \leq \frac{2L}{R}\left(1 - \omega_d \frac{R^d}{2^d|\mathcal{F}|}\right)^N,$$

where ω_d is the volume of the unit sphere in d dimensions.

Although the quantities L and R are difficult to know a priori, the preceding theorem at least sheds some light on the nature of the dependence of the algorithm on them. The fact that the dependence on N is exponential is a good feature. The base of the exponential $1 - \omega_d R^d/(2^d|\mathcal{F}|)$ can of course be very close to 1 and this means that we need to go to higher values of N. The dependence of this base on R is also derived. Another nice feature revealed is that the dependence on L is only linear. Any estimate on L, R and $|\mathcal{F}|$ would allow us, using the above analysis, to decide how large N should be so that we have at least a certain probability of success.

Chapter 5

Computation of C-space Using the Fast Fourier Transform

5.1 Introduction

In Chapter 3 of this dissertation we constructed C-space bitmaps to aid collision checking in two-dimensional workspaces. Precomputed C-space bitmaps explicitly represent the free part of the C-space (with 0's) and the part that gives rise to collision with an obstacle (with 1's) and reduce collision checking to a constant time operation [Latombe, 91b].

In fact, there is a variety of other ways a precomputed C-space bitmap can be exploited by a planner. For example, in [Lengyel et al, 90] it is used by a wavefront propagation algorithm to numerically compute a local-minima-free potential field with a single minimum at the goal. Another possible use of the bitmap is to generate a more concise representation of the free space by grouping adjacent free configurations into hyperparallelepipeds of various sizes ("approximate cell decomposition" approach to path planning [Brooks and Lozano-Pérez, 85, Zhu and Latombe, 91]).

We describe below a new method for computing the C-space bitmap. In its most basic form, it applies to the case where the robot is a d-dimensional ($d = 2$ or $d = 3$)

rigid object translating in an d-dimensional workspace among obstacles. The C-space, which in this case is also d-dimensional, can be regarded as a convolution of the workspace and the robot [Guibas et al, 83]. We compute this convolution via a Fast Fourier Transform (FFT) algorithm. The running time of our algorithm depends only on the resolution of the discretization used. For a fixed resolution it is independent of the complexity and the shape of the robot and the obstacles. Moreover, the method can benefit directly from specific hardware developed for the FFT algorithm.

The same method can be extended to a robot that can both translate and rotate in the plane by discretizing the range of orientations of the robot and building a three-dimensional C-space bitmap slice by slice. In theory, this extension also applies to a robot that translates and rotates in three dimensions, but then the dimension (6) of the C-space makes the construction of an explicit bitmap unrealistic. However, if the rotation is restricted to occur about a single axis, a relatively coarse four-dimensional bitmap can be constructed as a set of three-dimensional slices, each computed at a fixed orientation of the robot. If the robot is a planar set of K rigid bodies connected by prismatic and revolute joints, the method can be used to build K bitmaps, each representing the C-space of one of the bodies as if it were free to translate and rotate. In the context of the planning approach described in [Barraquand and Latombe, 91] and the method described in this dissertation, these K bitmaps can be used to compute whether a configuration of the robot is collision-free or not in constant time.

Although there has already been considerable research and results in C-space computation, fast computation of C-space obstacles remains an important issue. This is in particular the case when the environment changes dynamically. We believe that C-space computation is basic enough in robotics (and other domains) to make it benefit from parallel or specific hardware implementations. In that respect, bitmap representations seem to be very suitable.

This chapter is organized as follows. Section 5.2 gives a survey of previous related work. In Section 5.3 we show that the C-space bitmap is a convolution of the workspace and the robot. In Sections 5.4 and 5.5 we discuss how to use the FFT algorithm for the efficient calculation of the above convolution. In Section 5.6 we present an experimental evaluation of the proposed algorithm and in Section 5.7 we discuss the possibility of hardware and parallel implementations using recent work on the FFT.

5.2 Survey of Existing Algorithms

Let us first introduce some notation. The robot is denoted by $\mathcal{A}$. $\mathcal{A}$ is a rigid body and moves in the workspace $\mathcal{W} \subset \mathbf{R}^d$, with $d = 2$ or $d = 3$. Let $\mathcal{B}$ denote one of the obstacles in $\mathcal{W}$, and $\mathcal{C}$ the C-space of $\mathcal{A}$. The subset of $\mathcal{W}$ occupied by $\mathcal{A}$ at configuration q is denoted by $\mathcal{A}(q)$. That is $\mathcal{A}(q) = \mathcal{A} + q = \{x + q : x \in \mathcal{A}\}$. The obstacle $\mathcal{B}$ in $\mathcal{W}$ maps into $\mathcal{C}$ to the region $\mathcal{CB} = \{q \in \mathcal{C} \mid \mathcal{A}(q) \cap \mathcal{B} \neq \emptyset\}$, which is called a C-obstacle.

Many proposed algorithms for computing the C-obstacles for $\mathcal{A}$ deal with the case where $\mathcal{A}$ can only translate and restrict the shape of the robot and the obstacles. We begin our survey with these algorithms.

The case where $\mathcal{A}$ and $\mathcal{B}$ are convex polygons has been extensively studied [Lozano-Pérez and Wesley, 79, Lozano-Pérez, 83] and an optimal algorithm has been proposed by Lozano-Pérez [Lozano-Pérez, 83] and Guibas [Guibas et al, 83]. They obtain the vertices of the also convex $\mathcal{CB}$ in $O(n_\mathcal{A} + n_\mathcal{B})$ time, where $n_\mathcal{A}$ and $n_\mathcal{B}$ denote the number of vertices of $\mathcal{A}$ and $\mathcal{B}$ respectively. In the case where $\mathcal{A}$ and $\mathcal{B}$ are non-convex polygons, Sharir [Sharir, 87] gave an algorithm which computes the boundary of $\mathcal{CB}$ in $O(n_\mathcal{A}^2 n_\mathcal{B}^2 \log(n_\mathcal{A} n_\mathcal{B}))$ time. In the case where $\mathcal{A}$ and $\mathcal{B}$ are generalized convex polygons, i.e., regions bounded by straight-line segments or circular arcs, Laumond [Laumond, 87] showed that $\mathcal{CB}$ can be computed in $O(n_\mathcal{A} + n_\mathcal{B})$ time. When $\mathcal{A}$ and $\mathcal{B}$ are locally non-convex generalized polygons a decomposition of $\mathcal{A}$ and $\mathcal{B}$ into convex generalized polygons yields an $O(n_\mathcal{A}^2 n_\mathcal{B}^2 \log(n_\mathcal{A} n_\mathcal{B}))$ algorithm [Laumond, 87]. For $\mathcal{A}$ or $\mathcal{B}$ not locally non-convex, the algorithm has $O((n_\mathcal{A} n_\mathcal{B} + c)\log(n_\mathcal{A} n_\mathcal{B}))$ time complexity, where c is $O(n_\mathcal{A}^2 n_\mathcal{B}^2)$ in the worst case [Latombe, 91a]. The case where $\mathcal{A}$ and $\mathcal{B}$ are convex polyhedra has been studied in [Lozano-Pérez, 81, Lozano-Pérez, 83]. The best known algorithm has been given by Guibas and Seidel [Guibas and Seidel, 86]. It constructs the boundary of $\mathcal{CB}$ in $O(n_{\mathcal{A}} + n_{\mathcal{B}} + c)$ time, where c is in the worst case $O(n_{\mathcal{A}} n_{\mathcal{B}})$. Finally, the method of Avnaim and Boissonnat in [Avnaim and Boissonnat, 88] can be adapted to compute the boundary of $\mathcal{CB}$ in $O(n_{\mathcal{A}}^3 n_{\mathcal{B}}^3 \log(n_{\mathcal{A}} n_{\mathcal{B}}))$ time, when $\mathcal{A}$ and $\mathcal{B}$ are any polyhedra.

The above algorithms *analytically* compute the boundary of the C-obstacle for each connected workspace obstacle and then take the union of all constructed boundaries to find the boundary of all C-obstacles. Path planners operating in discretized C-spaces feed the results produced by these methods to routines that build the C-space bitmap. For example, Latombe [Latombe, 91b] and Lengyel et al. [Lengyel et al, 90] used the $O(n_{\mathcal{A}} + n_{\mathcal{B}})$ algorithm of Lozano-Pérez and Guibas to compute the vertices of the C-obstacles and then filled the C-obstacles with a raster graphics polygon-filling routine. Paths were computed in the produced bitmap.

Other methods for computing C-space maps that are closer to the spirit of our method have been proposed. Newman and Branicky [Newman and Branicky, 91] identify "elemental building blocks" (shapes) that are easily transformed from the workspace to the C-space. They store the C-space transforms of these shapes, frequently as bitmaps, and combine them to obtain the C-space bitmaps for complex shapes. They report experiments where the running time of their algorithm is very good. Lozano-Pérez and O'Donnell [Lozano-Pérez and O'Donnell, 91] implemented an algorithm on the Connection Machine that computes a family of C-space bitmaps which are then combined to construct bitmaps for more complex workspace objects and a 6-dof robot. An algorithm given by Dehne, Hassenklover and Sack [Dehne et al, 89] computes C-space bitmaps for arbitrary obstacles but "rectilinearly" convex robots on a $N \times N$ mesh of processors in $O(N)$ time. The above bound is generalized in here to non-convex robots. This improvement is also reported in similar to ours work done independently by Schwarzkopf [Schwarzkopf, 93].

There are some inherent differences between the algorithms that analytically compute the C-obstacles and the algorithms that construct bitmaps for the C-space. The time complexity of the first is measured in terms of the number of vertices of the robot and the obstacles. On the other hand, the running time of the bitmap algorithms is a function of the size of the constructed bitmap. Typically, robots and obstacles with many vertices demand a high resolution of discretization for describing their shapes with sufficient accuracy. Hence, there is a relationship between the two complexity measures, but this relationship is not well-defined.

In the next sections we describe a method for computing C-space maps whose only inputs are a bitmap of the workspace and an algorithm to draw the robot on that bitmap. The output is a C-space bitmap representing both the boundary and the interior of the C-obstacles (1's) and the free space (0's). Thus, when using C-space maps, we restrict the C-space of the robot to a discrete space, where each dof can only assume a finite number of values. The running time of our method is independent of the number and the shape of the obstacles in the workspace $\mathcal{W}$, and depends only on the resolution of discretization.

5.3 Configuration Space as a Convolution

In this section, we explain in two dimensions how the C-space can be obtained as a convolution of the workspace and the translating robot. The same discussion also applies in three dimensions.

Consider a workspace $\mathcal{W} = [a,b] \times [c,d] \subset \mathbf{R}^2$. We discretize $\mathcal{W}$ into a $N \times N$ array W[1], where N is large enough to represent the obstacles in $\mathcal{W}$ with the desired accuracy. We define

$$cell_{i,j} = [a + i\frac{(b-a)}{N}, a + (i+1)\frac{(b-a)}{N}] \times [c + j\frac{(d-c)}{N}, c + (j+1)\frac{(d-c)}{N}],$$

where $i,j \in S = \{0,\ldots,N-1\}$. If there is an obstacle anywhere in the cell $cell_{i,j}$ we let $W(i,j) = 1$, else $W(i,j) = 0$. The bitmap array W can easily be constructed if the obstacles are input as a collection of algebraic shapes, e.g., polygons represented by their vertices. When the data about the obstacles is obtained through sensing,

[1]In general, $\mathcal{W}$ is discretized into a $N \times M$ array. The fact that we use a square array is only for simplifying our presentation. It has no particular importance in our method.

the bitmap representation may be more straightforward to obtain than an algebraic representation.

As described in Section 2.1 $\mathcal{C}$ is the set of all triples (x, y, θ), where (x, y) are the coordinates of a fixed reference point on the robot, $p_{\mathcal{A}}$, and θ is the orientation of the robot. In general the parameters x, y, θ can assume any real value in $[a, b] \times [c, d] \times [0, 2\pi]$. We discretize the workspace and the orientations of the robot and define

$$x = a + i\frac{(b-a)}{N}, \;\; y = c + j\frac{(d-c)}{N} \text{ and } \theta = k\frac{2\pi}{N},$$

where $i, j, k \in S$. The C-space can then be stored as a $N \times N \times N$ binary array.[2]

The robot $\mathcal{A}$ can be approximated by a set of points (i, j), $i, j \in S$, which are drawn by a simple procedure given the orientation θ of the robot and the coordinates (x, y) of $p_{\mathcal{A}}$. For each fixed value of (x, y, θ) we consider the $N \times N$ binary array $A_{(x,y,\theta)}$ where only the points that belong to the robot are marked with 1's.

With the conventions of the previous paragraphs, a point (x, y, θ) in the (discrete) C-space is legal (free) if and only if

$$C(x, y, \theta) \equiv \sum_{i,j=0}^{N-1} W(i, j) A_{(x,y,\theta)}(i, j) = 0.$$

We observe that whenever θ is fixed and x, y are varying, the various bitmaps $A_{(x,y,\theta)}$ are all translations of each other, and in particular of $A_{(0,0,\theta)}$. Then

$$C(x, y, \theta) = \sum_{i,j} W(i, j) A_{(0,0,\theta)}(i - x, j - y).$$

For the moment we will ignore any complication that might arise concerning the range of the indices i, j. We can assume that W and $A_{(0,0,\theta)}$ are infinite in all directions and padded with zeros (which indicate free space and do not affect the above sum). What we have shown is that the array $C(\cdot, \cdot, \theta)$ is the convolution of the arrays W and A'_θ, where

$$A'_\theta(i, j) = A_{(0,0,\theta)}(-i, -j).$$

Indeed, the convolution of two arrays Q and T is defined as $(Q \star T)(x, y) = \sum_{ij} Q(i, j) T(x - i, y - j)$. Hence, $C(\cdot, \cdot, \theta) = W \star A'_\theta$.

[2]Footnote 1 applies for θ as well.

5.4 Computation of Convolution Via FFT

The convolution of two functions f and g can be computed by taking the Fourier Transform (FT) of the two functions, multiplying the two transforms pointwise, and then taking the inverse FT of the product. This is asserted by the Convolution Theorem given below:

Convolution Theorem: If functions f and g defined on $\mathbf{R}$ are integrable then, $\widehat{f \star g}(x) = \hat{f}(x)\hat{g}(x)$, where $\hat{h}$ denotes the FT of function h.

However, calculating the convolution with the use of the Convolution Theorem is of computational interest only when the FFT algorithm can be applied. The FFT algorithm can be used when the functions f and g are periodic with the same period, or what amounts to the same thing, when they are defined on a "cyclic" space. The convolution and the FT are then defined as follows for the two-dimensional case [Ramirez, 85]:

Definition 1: The convolution of two functions f and g defined on the set S^2, where $S = \{0, 1, \ldots, N-1\}$, is the function $(f \star g)(x, y) \equiv \sum_{i,j \in S} f(i, j)g(x - i, y - j)$ on S^2, where the arithmetic on the indices is done modulo N.

Definition 2: The FT of f on S^2 is the function $\hat{f}(x, y) = \sum_{j,k \in S} f(j, k)\zeta^{-jx-ky}$ on S^2, where $\zeta = \exp(2\pi i/N)$.

The Convolution Theorem now holds for f and g defined as above. When this theorem is used, the convolution computed is the one given in Definition 1.

The classic FFT algorithm computes the one-dimensional FT of functions on S, as well as its inverse (which is again defined on S), in time $O(N \log N)$. The two-dimensional FT can be computed by first taking the one-dimensional FT of all rows of the array, which is a function on S^2, storing the results in the rows themselves, and then taking the FT of all columns and putting the result back in the columns. The time needed to compute a two-dimensional FT is $O(N^2 \log N)$. Analogous definitions hold for the d-dimensional FT whose computation takes $O(N^d \log N)$, when d is fixed.

5.5 Computation of Configuration Space

5.5.1 Basic Algorithm

Let us now consider the problem of computing the convolution of the workspace bitmap W and the robot map A'_θ. We could proceed by using the definition of the convolution which directly yields an $O(N^4)$ time procedure. Rather, we wish to use the FFT results and compute this convolution in $O(N^2 \log N)$ time. However, the functions W and A'_θ involved in the convolution are *not* "cyclic" as is required by Definition 1. We eliminate this difficulty by simply setting $W(i,j)$ to 1 on the boundary of S^2. By doing so, we make sure that the robot cannot "wrap around" the workspace without a collision.

The actual running time of the proposed method critically depends on how fast we can compute the discrete FFT of N points. The FFT algorithm is a very popular algorithm. Optimized implementations of this algorithm are available on virtually every computer. Many FFT implementations have been tailored to exploit the pipelines of RISC processors and achieve close to peak performance. As discussed in Section 5.7, FFT hardware can reduce the running time of our method.

5.5.2 Algorithm for a translating and rotating planar robot

In Figure 5.1 we give the algorithm that computes the C-space bitmap for a robot $\mathcal{A}$ that can translate and rotate in the plane. As before, x and y are the coordinates of a fixed reference point ($p_{\mathcal{A}}$) of the robot and θ is the orientation of the robot; N is the resolution of the discretization on the workspace bitmap W. If N is also the number of the desired orientations of the robot, then a $N \times N \times N$ C-space bitmap CSPACE is constructed. Even when the number of desired orientations is chosen different from N, it is expected to be $O(N)$ and the above assumption does not affect our analysis in any way.

Put $W(i,j) = 1$ when either i or j is 0 or $N-1$,
  or an obstacle is present.
Compute $\widehat{W}$, the 2D FT of W.
For all desired values of θ:
begin
  Construct A'_θ.
  Compute $\widehat{A'_\theta}$, the 2D FT of A'_θ.
  Let $X = \widehat{W} \cdot \widehat{A'_\theta}$ (pointwise multiplication).
  Compute Y as the inverse FT of X.
  Let CSPACE$(x, y, \theta) = 1$ iff $|Y(x,y)| = 1$.
end

Figure 5.1: Computation of C-space map

If we have a good way to draw the robot in the bitmap for every new orientation, the running time of the algorithm is not significantly affected by the shape of the robot. Its time complexity is essentially $O(N^3 \log N)$. Robot drawing can often be simplified by drawing only the boundary of the robot. Then the region filled by 1's in the CSPACE bitmap is only a subset of the true C-obstacles, but it contains the boundary of the latter. For planners like the one described in [Barraquand and Latombe, 91], this is sufficient to prevent collision because the robot is never allowed to cross a C-obstacle boundary.

5.5.3 Algorithm for a translating robot in three dimensions

Although we focused our previous discussion on a robot moving in two dimensions, the same method applies in three dimensions. If $\mathcal{A}$ can only translate, a three-dimensional C-space can be constructed in $O(N^3 \log N)$ time, by taking the three-dimensional convolution of the workspace W and the robot $A'_{\phi\theta\psi}$. If rotation is allowed to occur about a single axis, say θ, a four dimensional bitmap can be constructed by an algorithm similar to the one given in Figure 5.1. The algorithm then has a time complexity of $O(N^4 \log N)$, if N is the number of desired values of θ. The same principle applies if rotation is allowed along the other axes. However, if a reasonable resolution of discretization is required, the size of the C-space bitmap becomes very large and it is not realistic to store it in the memory of current workstations.

5.6 Experimental Evaluation

We have implemented the FFT-based algorithm as this is described in Figure 5.1 except for a minor change to compensate for possible arithmetic errors. We set CSPACE$(x, y, \theta) = 1$ when $|Y(x, y)| >$ THRESHOLD, instead of when $|Y(x, y)| = 1$ (line 9). In our experiments THRESHOLD was set to 0.9. We have conducted a series of experiments aimed at comparing the performance of the algorithm with (i) an implementation of the $O(n_{\mathcal{A}} + n_{\mathcal{B}})$-time algorithm of Lozano-Pérez [Lozano-Pérez, 83] and Guibas [Guibas et al, 83] and (ii) an optimized implementation of the $O(N^4)$-time algorithm that computes the convolution of the workspace and the robot in a straightforward way. All algorithms in this section are implemented in C and run on a DEC 5000 workstation. This machine is rated at 18.5 SPECmarks89. (This machine is considerably slower than the one used in the experiments of Chapter 3.)

5.6.1 Comparison with the linear $O(n_{\mathcal{A}} + n_{\mathcal{B}})$ algorithm

As discussed in Section 5.2, the input and output data are different for algorithms that compute C-obstacles (in this case the $O(n_{\mathcal{A}} + n_{\mathcal{B}})$ algorithm) and algorithms that compute C-space bitmaps (in this case the FFT-based algorithm). An analytic comparison between these algorithms does not seem possible. We tried to establish a reasonable framework for an experimental comparison. Our aim is to provide orders of magnitude for the running times of the FFT-based algorithm and the $O(n_{\mathcal{A}} + n_{\mathcal{B}})$ algorithm, which has been used in many motion planners [Latombe, 91b, Lengyel et al, 90].

The input of the implemented FFT-based algorithm is an $N \times N$ bitmap W of the workspace. The robot is defined as a polygon. Its bitmap representation is

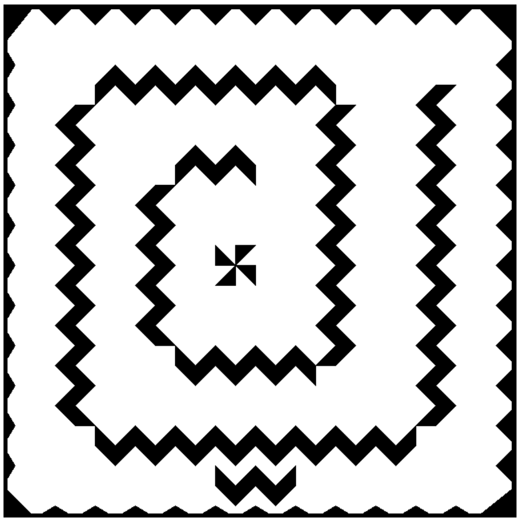


Figure 5.2: Workspace with 150 convex polygon obstacles

not precomputed; the algorithm computes the bitmap representation of the robot's boundary for each of the N orientations of the robot that we considered. The output is an $N \times N \times N$ bitmap CSPACE representing the C-space.

The input of the implemented linear algorithm is a collection of convex polygons representing the robot and another collection of polygons representing the workspace obstacles. Hence, the decomposition of non-convex polygons into convex ones is not part of the running times given below. The output of the linear algorithm is entered into a routine mapping this boundary in a C-space bitmap of the same resolution as the one built by the FFT-based algorithm. This bitmap is the output of the algorithm. When we have an analytic description of the C-obstacles, we can construct a CSPACE bitmap of high resolution with accuracy. In the FFT-based algorithm, we first project the obstacles and the robot in the workspace bitmap and then compute a CSPACE bitmap of the same resolution. Thus, in order to obtain an accurate C-space bitmap, we have to work with a resolution high enough to represent the shape of the obstacles. In our experiments we made sure that the CSPACE bitmaps produced by the algorithms we compare were almost the same.

Figure 5.2 shows the workspace (150 convex polygonal obstacles) and Figure 5.3

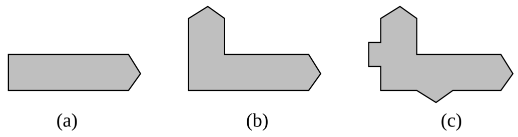


Figure 5.3: Polygonal robots

shows the three robots we used in one series of experiments. The larger dimension of the robots is approximately 1/10 of the dimension of the workspace. The workspace bitmap W was represented as a 128×128 array and we considered 128 orientations for the robot. Hence, the algorithms built a $128 \times 128 \times 128$ bitmap CSPACE.

- For the convex robot of Figure 5.3(a), the FFT-based algorithm took 90 seconds to compute CSPACE and the linear algorithm 11 seconds.
- For the robot of Figure 5.3(b), which is non-convex, the FFT-based algorithm took 91 seconds and the linear algorithm 22 seconds.
- For the robot shown in Figure 5.3(c), which has four convex parts, the FFT-based algorithm took 91 seconds and the linear algorithm 32 seconds.

Experiments show clearly that the linear algorithm is preferable for simple polygonal robots and workspaces. However, as the complexity of the environment increases, our FFT-based algorithm becomes more and more comparable. One could imagine more complex environments where the gap in the running times of the FFT-based and the linear algorithm will be reduced or even inverted.

We emphasize that we ran our experiments with algorithms implemented in software on a conventional single-processor architecture. A hardware or parallel implementation of the FFT algorithm would certainly lower the "break-even" complexity where the FFT-based algorithm becomes preferable to the linear algorithm.

5.6.2 Comparison with the direct convolution algorithm

Although the FFT-based algorithm has an asymptotically better running time than the $O(N^4)$ direct algorithm, the latter requires only a small bitmap for the robot and

Size of robot bitmap	Bitwise operations (sec)
5×5	20
10×10	39
20×20	77
30×30	115
40×40	199

Table 5.1: Direct convolution with a 128×128 workspace

can be implemented using (fast) bitwise operations, instead of (slow) multiplications.

The input and the output of the implemented direct convolution algorithm are exactly the same as those of the FFT-based algorithm. The direct algorithm augments the workspace bitmap and pads it with zeros to avoid problems at boundary configurations when the convolution is computed. Table 5.1 summarizes experimental results for a 128×128 workspace. The FFT-based algorithm computes the bitmap CSPACE in approximately 90 seconds. Column 2 of Table 5.1 shows the time required to compute CSPACE for different sizes of the bitmap of the robot when bitwise operations are used. It is clear that the direct approach is preferable when the size of the robot is small. Bitwise operations are implemented in hardware in the machine we used and take at most a couple of machine cycles.

5.7 Hardware and Parallel Implementations

A hardware implementation of the basic FFT-based method for computing C-space maps is possible because of the uniformity of the calculations involved in the algorithm. Such an implementation may substantially reduce the running time of the algorithm. Even if the whole method is not implemented in hardware, specialized FFT hardware [Bilardi et al, 91] can be used. The latter is widely available and has been used for years in signal processing and other applications.

Hardware implementations are also possible for the direct convolution algorithm and cannot be regarded as an advantage of the FFT-based algorithm only. This issue has been studied in the context of the morphological operations used in image processing [Haralick and Shapiro, 92]. The computation of these operations requires a small bitmap, referred to as a *kernel*, to be moved over the original image. Many specialized architectures have been proposed [Reeves, 84, Pratt, 85] to efficiently perform these operations. However, these architectures achieve high performance for operations with large images and small kernels. Typical kernel sizes for a 512×512 image are 3×3 or 5×5. In our problem, we might not want to restrict ourselves to such small robots.[3]

As far as our basic algorithm is concerned, its parallel implementations can be based on existing parallel implementations of the FFT. Let us mention some existing results. Leighton [Leighton, 92] showed that the N-point discrete FFT can be implemented in $\log N$ steps on an $N(\log N + 1)$-node butterfly or an N-node hypercube. It can also be implemented in $O(\log N)$ steps on an N-node butterfly [Leighton, 92]. Issues related to the implementation of the FFT on hypercubes are discussed in [Swarztrauber, 87]. On an $N \times N$ mesh of processors, it is easy to implement the (two-dimensional) FT of a function defined on N^2 points in $O(N)$ time [Leighton, 92]. Then our basic algorithm will itself have a complexity of $O(N)$. This is asymptotically optimal for the problem, and is an improvement over the algorithm of Dehne, Hassenklover and Sack [Dehne et al, 89] which can only deal with "rectilinear" robots. For the Connection Machine there exist many implementations of the FFT [Johnsson and Krawitz, 92, Tong and Swarztrauber, 91]. On a shared memory bus-based multiprocessor with few processors (8) an implementation of the multi-dimensional FFT exhibits a speedup close to linear with the number of processors used.[4]

[3]Even in AVG applications, path planning frequently has to be done locally, since more global planning often makes use of higher-level symbolic maps defining intermediate goals.

[4]The almost linear speedup is justified by the way the d-dimensional FFT is computed. For example, for d=2, we first take the FFT of the rows and then the FFT of the columns. We can assign different rows/columns to different processors. The partial computations are independent.

5.8 Some Remarks

We presented an FFT-based algorithm to compute the C-space map used by several path planners including the one described in this dissertation. The algorithm is based on the observation that the C-space map is the convolution of the workspace and the robot and applies to obstacles and robots of any shape input as bitmaps. The FFT-based method produces a bitmap representation of the C-space that is directly exploitable by potential field based path planners such as those described in [Barraquand and Latombe, 91, Lengyel et al, 90], and the planning approach described in Chapters 2 and 3 of this dissertation. In particular, this bitmap allows collision checking to be done in constant time for a rigid robot, or a robot with a small number rigid parts. For a fixed discretization, the running time of our algorithm is independent of the number and the shape of the obstacles in the workspace.

The FFT-based method was presented in the case of a planar translating robot. Rotation is not directly handled by the method and requires slice by slice construction of the C-space bitmap. The method also applies to a three-dimensional robot that can only translate. Furthermore, it can be used when the robot is a planar articulated linkage, by building a three-dimensional C-space for each link.

Our method depends highly on our ability to perform the FFT. When implemented in software on a single-processor computer, its running time can be a disadvantage, especially when the obstacles and the robot are simple. However, when the combined complexity of the obstacles and the robot is high, the FFT-based method becomes advantageous. It can also benefit from existing special-purpose hardware. Implementations are possible on a wide variety of parallel architectures, by exploiting recent work done on parallel implementations of the FFT. The use of specific hardware or parallel architectures can drastically reduce the running time of our algorithm, making it compare even more favorably to previous algorithms for computing C-space maps.

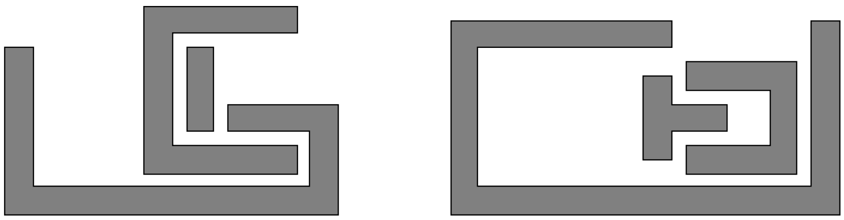
Chapter 6

The Complexity of Assembly Partitioning

6.1 Introduction

In the final part of this dissertation, we study the *assembly partitioning* problem in the plane: given a collection of non-overlapping polygons, decide if there is a proper subcollection of them that can be removed as a rigid body without colliding with or disturbing the other parts of the assembly.

The partitioning problem arises in assembly planning, where a sequence of (possibly simultaneous) assembly motions are sought to bring separated parts into their relative goal positions in an assembly. Automation of this process would be invaluable for the evaluation of the design process in manufacturing, as well as for the repair and maintenance of mechanical parts. Natarajan [Natarajan, 88] showed that in its most general form, assembly planning is PSPACE-hard. In practice, however, most real assemblies can be constructed with *monotone two-handed* assembly plans [Homem de Mello and Lee, 91]. Such plans consist of a sequence of operations, where each operation merges two rigid subassemblies to make a larger one that stays rigid for the rest of the plan. For instance, the assembly in Figure 6.1(a) can be constructed



(a) (b)

Figure 6.1: Assembly (a) admits a monotone two-handed plan, while (b) does not

with a monotone two-handed plan, while the assembly in Figure 6.1(b) cannot.

Since assembly is the reverse of disassembly, a monotone two-handed assembly plan can be found by *partitioning* the assembly—removing a rigid subassembly—and then partitioning each of the two subassemblies, etc., and finally reversing the motions. Thus partitioning is the key to assembly planning in this case.

Recent work has presented polynomial-time partitioning algorithms in special cases. Arkin, Connelly and Mitchell [Arkin et al, 89] give an algorithm to partition assemblies of polygons if the separating motions are limited to single infinite translations. Wilson [Wilson, 92] presents algorithms to partition assemblies of polyhedra, where the separating motions are either infinite translations or infinitesimal rigid motions (the latter identifying a superset of the removable subassemblies for general separating motions). Interestingly, Snoeyink and Stolfi [Snoeyink and Stolfi, 93] present an assembly of convex polyhedra that cannot be partitioned. Other related geometric separation problems are studied in [Guibas and Yao, 83, Nussbaum and Sack, 93, Pollack et al, 88, Toussaint, 85].

Here we show that the partitioning problem for polygons in the plane is NP-complete. Our proof extends to some interesting variants of the problem, including the case where all motions are restricted to translations.

6.2 Complexity of Planar Partitioning

Let a *rigid motion* of a subassembly S be a set of simultaneous motions (translations and rotations) of the parts of S that preserve the relative positions of these parts throughout the motion. The subassembly S is then called a *rigid* subassembly. The problem considered is stated below:

Planar Partitioning (PP) *Given a set A of non-overlapping polygons in the plane, decide if there is proper subset S of A that can be separated from $A \setminus S$ by a collision-free rigid motion of S.*

We will show that PP is NP-complete. It is clearly in NP, since a nondeterministic algorithm can guess S and then invoke a path planner to find the path of S out of the assembly. Schwartz and Sharir [Schwartz and Sharir, 83a] have shown that path planning can be done in polynomial time for the case considered here.

We show that PP is NP-hard by a reduction from 3-Satisfiability (3-SAT), a well known NP-complete problem [Garey and Johnson, 79]. An instance of 3-SAT is a set of clauses $C = \{c_1, c_2, \ldots, c_m\}$ on a set of boolean variables $U = \{u_1, u_2, \ldots, u_n\}$, where each clause is a disjunction of 3 terms. A term is either a variable u_i or the negation of a variable $\overline{u_i}$. The problem is to determine if there exists a truth assignment of the variables that satisfies the conjunction of the clauses.

For any instance of 3-SAT, we construct in polynomial time an assembly of non-overlapping polygons that can be partitioned iff a satisfying truth assignment exists. Figure 6.2 shows the *outside box* and the *key* of the constructed assembly. Other parts are contained in the *assignment mechanism* and the *AND/OR mechanism*, detailed in Figures 6.3 and 6.4 respectively. Our construction is summarized in the following:

- The part labeled as key in Figure 6.2 must be removed before any other part is removed from the assembly. The reason is that the key blocks the only exit gate of the assembly.

- The key can be removed only through the assignment construct. For this to

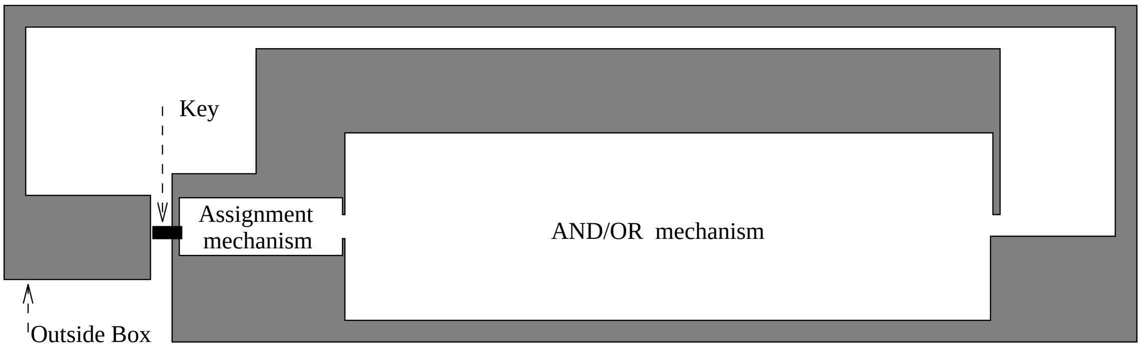


Figure 6.2: A sketch of the final assembly

happen some other parts of the assignment construct must move rigidly with it. These parts represent a truth assignment for the variables of the 3-SAT instance.

- The subassembly can be removed only through the AND/OR mechanism. This mechanism enforces the clauses of the 3-SAT instance.

The assignment mechanism (Figure 6.3) consists of (i) the walls of the assignment that are drawn in grey, (ii) the key which is a 3×1 rectangle drawn in black, and (iii) two 3×1 white rectangles for each of the variables of the 3-SAT instance. One of the two rectangles that correspond to the variable u_i is labeled with U_i, and the other with $\overline{U_i}$. Notice that U_i is placed always on top of $\overline{U_i}$ in the assignment construct. If U_i, ($\overline{U_i}$ resp.) is a member of S, we consider that the truth assignment *true* (*false* resp.), has been chosen for the variable u_i. We indicate with x_0 the initial position of the key and with x_i, the initial position of the assignment rectangles for the variable u_i, $i = 1, \ldots, n$. The choice of the x_i's is crucial and it is described below.

From Figure 6.2 it is clear that the key initially can move only to the right. We observe that the key will not be able to pass through the assignment mechanism if no other parts of the mechanism are moved. In addition, the parts removed must translate rigidly. Hence, the only subassembly S that stands a chance to move out of the assignment mechanism, and eventually out of the total assembly, is a subassembly that consists of the key and at least one of U_i or $\overline{U_i}$, for each i. Since the exit gate of

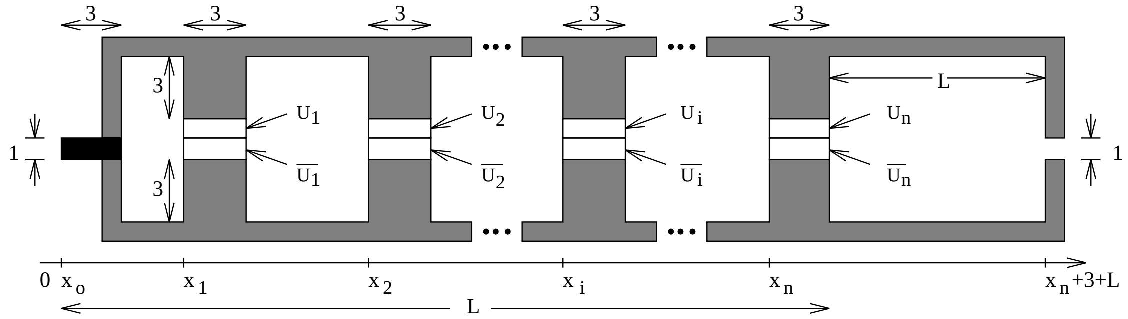


Figure 6.3: The assignment construct (not drawn to scale)

the assignment mechanism has a height of only 1, exactly one of U_i or $\overline{U_i}$, for each i, must be selected in S. Let L be the length of the moving subassembly S.

The collision-free motion of S out of the assignment construct is possible only if the x_i's are selected carefully. We present now our choice of x_i's and justify it. For the following assume that $S = \{R_0, R_1, \ldots, R_n\}$, where R_0 denotes the key and R_i denotes either U_i or $\overline{U_i}$, $1 \leq i \leq n$. Also, $pos(R_i)$ denotes the x-coordinate of the bottom-left vertex of the rectangle R_i.

Observation *Let $x_i = 10 \cdot a_i$, $i = 0, 1, \ldots, n$, where $a_0, a_1, \ldots, a_n$ is an integer sequence such that all pairwise differences $a_i - a_j$ are distinct when $i \neq j$. When $|x_p - pos(R_i)| < 4$, $p \neq i$, then for all R_j, $j \neq i$, we have that $|x_q - pos(R_j)| > 4$.*

Proof: At some time during the motion of S, R_i is 4-close to the position x_p, that is $|x_p - pos(R_i)| < 4$. Consider now any R_j, $j \neq i$, and any position x_q. Let us bound the quantity $|x_q - pos(R_j)|$ from below

$$|x_q - pos(R_j)| \geq |(x_p - pos(R_i)) - (x_q - pos(R_j))| - |x_p - pos(R_i)|,$$

or equivalently,

$$|x_q - pos(R_j)| \geq |(x_p - x_q) + (pos(R_j) - pos(R_i))| - |x_p - pos(R_i)|.$$

But since $|x_p - pos(R_i)| < 4$ and $pos(R_j) - pos(R_i) = x_j - x_i = 10 \cdot a_j - 10 \cdot a_i$ the above inequality becomes

$$|x_q - pos(R_j)| > 10 \cdot |a_p - a_q - (a_j - a_i)| - 4.$$

From the hypothesis about a_n the quantity $|a_p - a_q + (a_j - a_i)| = |(a_j - a_q) - (a_i - a_p)|$ must be at least one, since $j \neq i$ and $i \neq p$. Thus we get

$$|pos(R_j) - x_q| > 10 \cdot 1 - 4 > 4,$$

which completes the proof. $\Box$

In other words, for any k when R_k is close to position x_p, $p \neq k$, and needs to go through the hole that has been created at this position, all the other parts of S are in the wide free sections of the assignment mechanism and can follow the constrained motion of the part that is close to x_p. Hence, the assignment mechanism ensures independent selection of the variable assignments of the 3-SAT instance and allows the resulting subassembly to translate out of the mechanism.

The key element in the above proof is the property of the sequence a_i, namely that the pairwise differences of its terms are all distinct. The 10 is just a scaling factor that gives extra space for the width of the parts. It is not hard to choose the a_i's; a straightforward example is given by $a_i = 2^i$. Using a result of Erdős [Halberstam and Roth, 83], it is possible to select the a_i's in such a way that $max\{a_0, \ldots, a_n\} = n^2$. The complexity of the assignment mechanism is measured in the number of vertices, and it is polynomial in n, no matter which of the above sequences is used. Using the result of Erdős the physical length of the assignment mechanism is $O(n^2)$.

Let us also point out that if we just want a sequence a_n of polynomial growth we can easily construct one with $a_n \leq n^3$ as follows: Let $a_1 = 1$ and having already chosen $a_1, \ldots, a_n$ we select as the value of a_{n+1} the smallest positive integer x such that

$$x \neq a_i + a_j - a_k, \quad \text{for all } i, j, k = 1, \ldots, n.$$

Indeed this selection guarantees that all sums of two elements of the sequence a_n are distinct and it is very easy to show that $a_n \leq n^3$ by induction (see [Kavraki and Latombe, 93a]).

Once S translates out of the exit gate of the assignment construct it must pass through the AND/OR mechanism. This mechanism is a sequence of OR gates, one

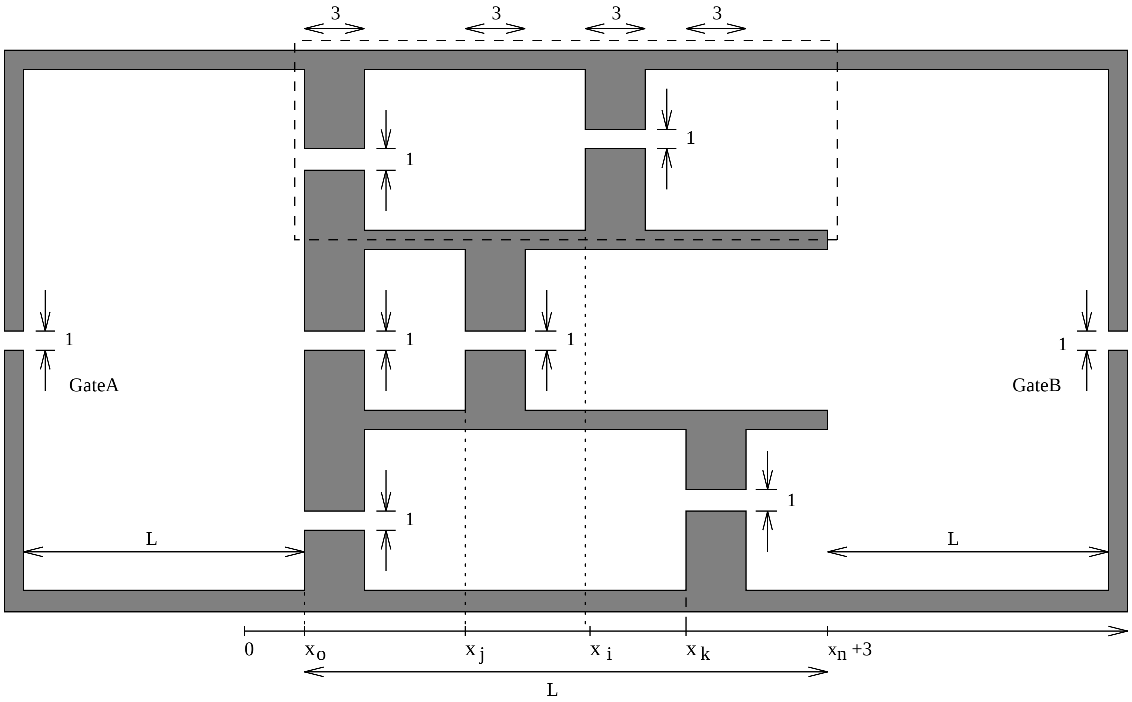


Figure 6.4: The OR gate for $c_l = u_i \vee \overline{u_j} \vee u_k$ (not drawn to scale)

for each clause of the 3-SAT instance. The OR gate for the clause $u_i \vee \overline{u_j} \vee u_k$ is shown in Figure 6.4. We observe from the figure that there are three possible ways for S to go from GateA to GateB. Each of these enforces a truth assignment for one of the terms of the clause. Suppose for example that S goes through the section enclosed in the dashed box in Figure 6.4. This section enforces the truth assignment *true* for the variable u_i. Here is why: when the key is at x_0, the rectangle chosen for the truth assignment of u_i is at position x_i. Unless U_i is selected in S, it is impossible for S to go through the dashed box of Figure 6.4. Notice however that the rest of S can be threaded through the gates at x_0 and x_i without problems: because of the property of the x_i's mentioned above, when a part of S needs to go through the above gates, none of the other parts of S is close to a narrow passage. Hence, S can follow the motion of its constrained part without being obstructed by the walls of the OR gate.

Suppose there exists a satisfying truth assignment for the 3-SAT instance. Let S be the subassembly that consists of the key and encodes this truth assignment. S can go through the AND/OR mechanism since it can translate through each of its OR gates. Then S can translate to the upper left corner of the assembly, rotate by 90 degrees and exit through the 2-unit wide gate that was initially blocked by the key.

Conversely, assume that the assembly in Figure 6.2 can be partitioned and let S be the subassembly that is removed from it. S clearly contains the key. As we argue above, the key can be removed only in a subassembly that contains a truth assignment for the variables of the 3-SAT instance. Since S can pass through the AND/OR mechanism, it represents a satisfying truth assignment for the 3-SAT instance. Finally, it can be shown easily that the reduction presented above is polynomial in the size of the 3-SAT problem. It follows that:

Theorem 6.1 *PP is NP-complete.*

6.3 Variants of Planar Partitioning

In this section we present some variants of the partitioning problem that are also NP-complete. A detailed discussion of these variants can be found in [Kavraki and Latombe, 93a, Kavraki et al, 93]. A different proof of variant 2.3 can be found in [Wilson et al, 92].

6.3.1 Partitioning with Translations

In the construction of the previous section, the moving subassembly S rotates in the upper-left corner of the assembly before exiting. We can modify that construction to remove the rotation needed, and show the following theorem:

Theorem 6.2 *Planar partitioning with translation only is NP-complete.*

A more complicated exit gate allows S to pass through in translation, requiring small changes in the rest of the construction.

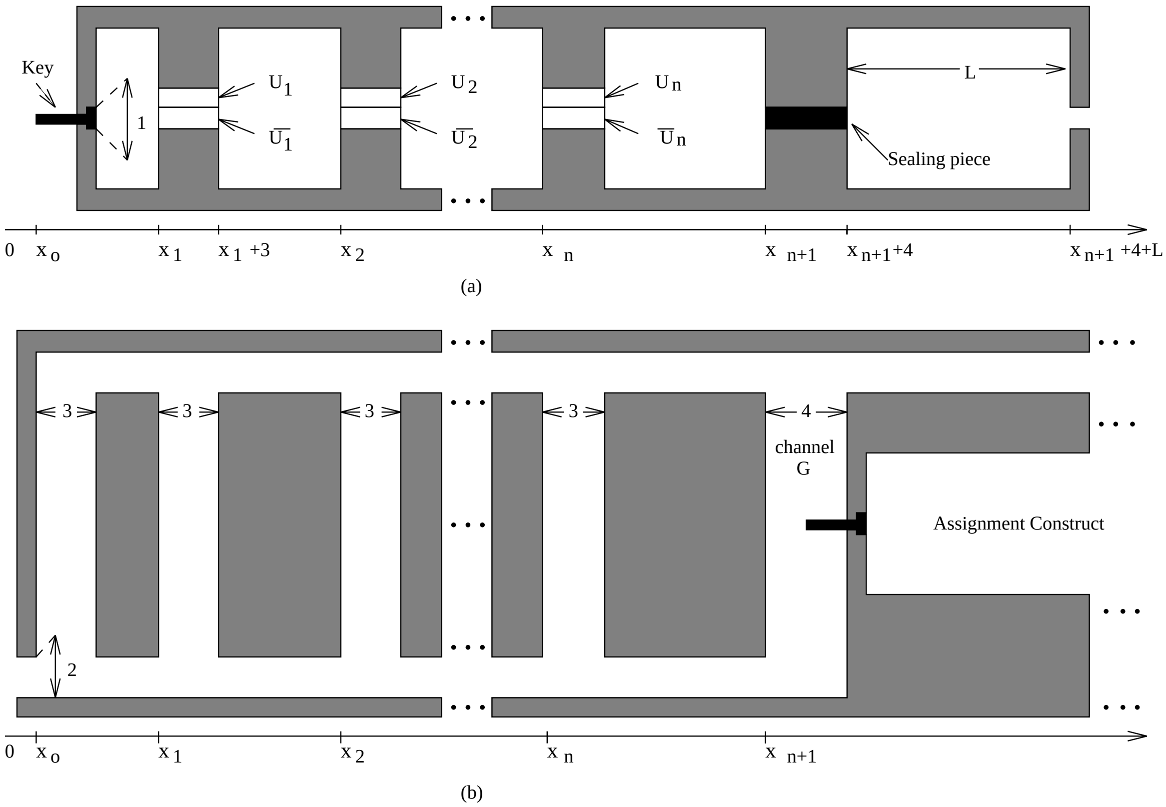


Figure 6.5: Partitioning with translation only: (a) the assignment mechanism (b) the exit mechanism

The new exit gate is depicted in Figure 6.5(b). A series of vertical channels allow the key and assignment rectangles to move down through the exit gate, then exit to the left. However, with no other changes, rectangle U_n or $\overline{U_n}$ could be removed alone. To prevent this, a *sealing piece* is added to the assignment construct, to the right of all the assignment rectangles, at position x_{n+1} (see Figure 6.5(a)); a corresponding channel (labeled G) is added to the exit gate. The sealing piece and channel G are both 1 unit wider than the other channels, and no rotations are allowed, so the sealing piece can only exit through gate G. Finally, since the key no longer touches the left side of G, its shape is changed to only allow movement to the right.

The sealing piece blocks the motion of the assignment rectangles, so it must be in S. The key blocks gate G, so it and exactly one of U_i or $\overline{U_i}$, for all i, must also be in S. Since the sealing piece is at position x_{n+1}, it does not affect the ability of S to move out of the assignment construct or through the AND/OR mechanism.

6.3.2 Partitioning on a Grid

Consider a polygonal assembly whose parts must (i) have their vertices on an $N \times N$ grid, (ii) have only right angles, and (iii) translate only in the vertical and horizontal directions by grid increments. N is considered the size of the problem. The planar partitioning problem for this assembly is called *partitioning on a grid* and is a very constrained partitioning problem.

Partitioning on a grid is clearly in NP: we guess a subassembly and a path of N^2 unit steps on the grid, then check whether the path is collision-free and remove the subassembly.

The reduction from 3-SAT is possible only because we can construct an assignment mechanism whose length is polynomial in the size of the 3-SAT instance (see Section 6.2). This leads to an assembly of length $O(mn^2)$, where n is the number of variables, and m is the number of clauses of the 3-SAT instance. The construction of Section 6.3.1 satisfies the above restrictions on the assembly, and the grid rules out non interesting motions. Thus, we have proved the following:

Theorem 6.3 *Partitioning on a grid is NP-complete.*

6.3.3 Assemblies With Parts of Constant Complexity

The NP-completeness result holds for partitioning with translation and rotation and also for partitioning with translation only, even if we require that the assembly to be partitioned is composed of parts of constant complexity. Parts with up to a constant number of vertices are considered as parts with constant complexity.

The only parts of non-constant complexity in our assembly construction are the walls of the assignment mechanism. We can build the same walls from parts of constant complexity as shown in Figure 6.6. Note that none of the D_{i1}, D_{i2} rectangles

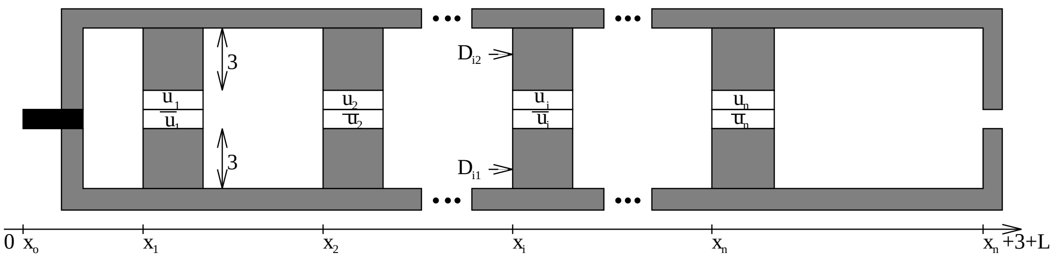


Figure 6.6: The assignment mechanism made from parts of constant complexity

can move out of the assignment mechanism and their presence does not affect the reduction from 3-SAT. It follows that:

Theorem 6.4 *Partitioning an assembly with parts of constant complexity is NP-complete.*

6.3.4 Completely Separable Assemblies

Let us now consider the case of an assembly that can be completely separated to its constituent parts by repeated monotone partitioning. This case is interesting because it concerns assemblies that can also be built from their constituent parts by a monotone assembly sequence.

The assembly constructed Figure 6.5 for PP without translation cannot be fully decomposed. The basic parts are the only parts that can move out of the assembly. Other parts, e.g., the walls of the assignment mechanism and the parts in the OR gates, are too large to pass through the narrow exit gate. Below, we modify the assembly of Figure 6.5 to allow its complete decomposition.

We first modify the exit mechanism. The new mechanism is shown in Figure 6.7(b). It is aligned below the assignment mechanism for PP without rotations, repeated and magnified here from Figure 6.5, for clarity purposes. The exit mechanism is composed of polygons $P_1, \ldots, P_{n+1}$. These parts are interlocked in their initial positions. They cannot translate out of the assembly, unless the key is removed. However, the relative positions of the gates in the exit mechanism are exactly the

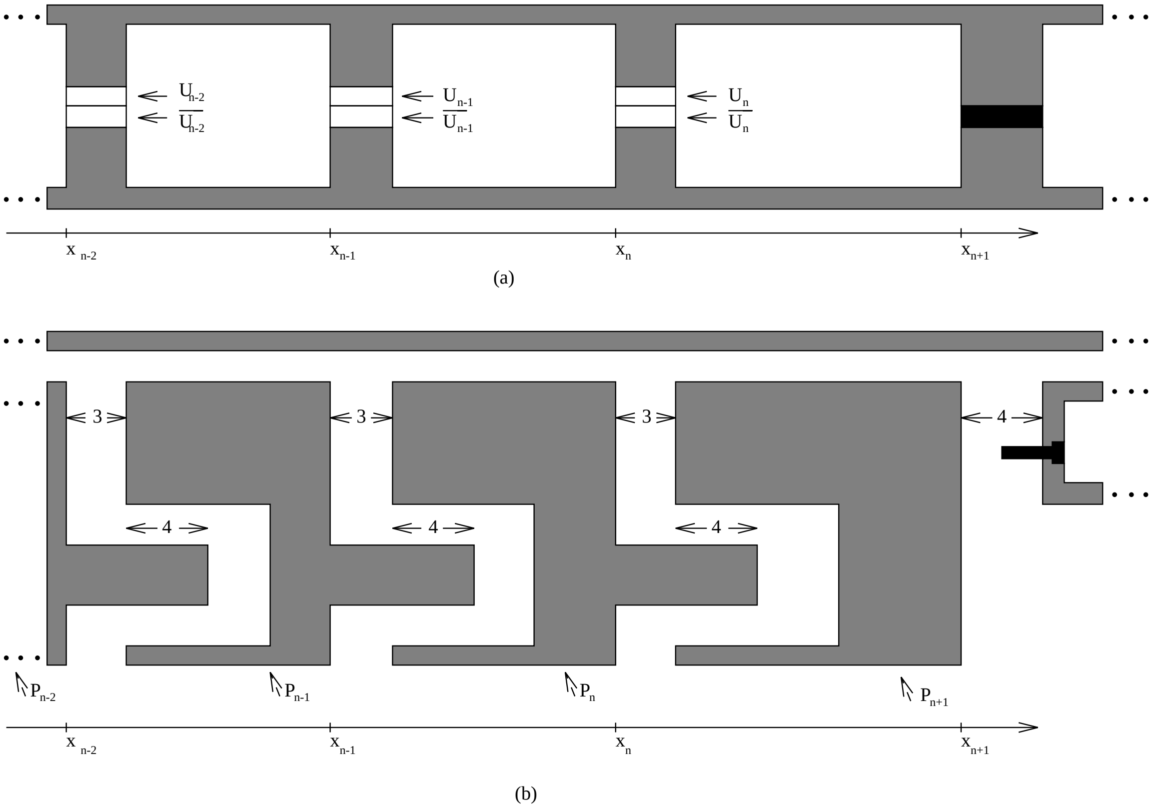


Figure 6.7: Completely separable assemblies: (a) the assignment mechanism, and (b) the exit mechanism

same as the relative positions of the parts in S. Hence, once S reaches the upper-left part of the outside box, it can translate out by moving through the spaces between $P_1, P_2, \ldots, P_{n+1}$.

After the key is removed with S, P_{n+1} can move horizontally to the left until it touches the walls of the assignment mechanism and then vertically down to translate out of the assembly. Once P_{n+1} is gone, all other P_i's can move out of the assembly. To permit the translation of the other parts out of the assembly, we break them into smaller parts. Namely, we break the assignment walls, as well as the walls and inside parts of the AND/OR mechanism, into parts of length L. These will be able translate out of the assembly one after the other through the opening, which is now L long. The total number of the parts inside the assembly is polynomial, since the physical size of the assembly can be made polynomial in the size of the 3-SAT instance. From the above we infer that:

Theorem 6.5 *Planar partitioning restricted to completely separable assemblies is NP-complete.*

6.3.5 Partitioning of Assemblies of Polyhedra

We now describe a polynomial transformation that reduces the problem of deciding if there is a monotone partition of an assembly of polygons, to the problem of deciding if there is a monotone partition of an assembly of polyhedra.

Let A be an arbitrary assembly of polygons. We construct an assembly of polyhedra, A', that has three layers. The lower and the upper layer are plates. The middle layer consists of the polyhedra produced by giving all the polygons of A a thickness of 1. Let P be one of these polyhedra. We connect P to the lower and upper layer to form a single part. We also make the size of the lower and upper plates many times the size of A. This is to ensure that in case P is chosen in the moving subassembly, all the parts of the middle layer will remain in between the upper and lower plates, until the moving subassembly is completely separated.

There exists a monotone partition for the polygons of A when these move in the plane, iff there exists a monotone partition for the polyhedra of A' when these move in space. Thus, monotone partitioning of a polyhedral assembly is at least as hard as monotone partitioning of a planar assembly. In addition, partitioning of an assembly of polyhedra is in NP from the results of [Schwartz and Sharir, 83a]. We have thus proved the following:

Theorem 6.6 *Deciding if there is a monotone partition of an assembly of polyhedra is NP-complete.*

All the polyhedral counterparts of the planar partitioning decision problems presented in this chapter can be shown NP-complete using an argument similar to the above.

6.4 Some Remarks

We have shown that monotone partitioning of an assembly of polygons in the plane is NP-complete when the polygons are allowed to both translate and rotate, and when they are allowed only to translate. We also presented some variants of the above partitioning problems that are NP-complete. Finally, we extended our results to assemblies of polyhedra. For results in other special cases see [Wilson, 92, Wilson et al 93].

In experimental assembly sequencing, an additional constraint is often added, requiring both S and $A \setminus S$ to be connected. A subassembly is considered connected if the union of its parts is a connected set. This constraint is useful in practice, since connected assemblies are easier to grasp and manipulate. It is not hard to see that partitioning of assemblies of polyhedra into connected subassemblies is NP-complete: in the construction of Section 6.3.5 a plate can be placed over the augmented parts, rigidly attached to the key and removed with S. In [Kavraki and Kolountzakis, 94] it is shown that partitioning of assemblies of polygons under the connectedness constraint is an NP-complete problem.

Chapter 7

Conclusion

7.1 Summary

This work addressed three different problems in the area of robotics.

The main part of the dissertation (Chapters 2, 3, and 4) described a two-phase method to solve robot motion-planning problems in static workspaces. In the preprocessing phase, the method constructs a probabilistic network as a collection of configurations properly selected across the free C-space. In the query phase, it uses this network to quickly process path-planning queries, each specified by a pair of configurations. The preprocessing phase includes a heuristic evaluator to identify difficult regions in the free C-space and increase the density of the network in those regions. This feature is key to solving difficult queries.

The method is general and can be applied to virtually any type of holonomic robot. With little modification it can also be useful for nonholonomic car-like robots (see [Švestka, 93, Švestka and Overmars, 94]). It is also an inherently parallel method permitting the class of solvable problems to go even beyond what we report in this work. The distributed nature of processing the C-space, simultaneously as a whole, makes the method conceptually simple and natural, not least in the sense that it follows the computational paradigm that the human brain does – processing the information not piecewise but globally.

We also showed that the planning method can be easily customized to run more efficiently on a given family of problems. Customization consists of replacing general components of the method, such as the local planner, by more specific ones fitting better the characteristics of the considered scenes.

One of the goals of this dissertation was to demonstrate how effective the technique is for difficult path-planning problems involving many-dof robots. It is precisely these problems for which there exist few practical path planners. To that effect, we reported extensive experiments with articulated robots moving in the plane or in space. To our knowledge, the problems we considered are very hard to solve, if they can be solved at all, with other existing planning techniques.

Sections 2.10 and 3.4 discussed several other nice features of the randomized method, as well as its drawbacks. It is clear from our experiments that the method is quite stable and that there is no disparity in the time taken to answer different planning queries, once a good network is constructed in the free C-space. In addition, failure to find a path between two configurations is quickly reported. In the case when many queries fail, it is possible to take advantage of the incremental nature of the technique and augment the current network. This can be done by reapplication of the preprocessing steps. It is possible to further "train" the network to a particular workspace/robot pair by recording in it the initial and final configurations of a series of queries, as well as the paths computed for these queries.

The numerical parameters of the method have to be chosen manually at this stage and this is a disadvantage of the approach. It would be nice to have some way to predict these parameters (see Section 7.2). But we have observed that good values for these parameters are easy to obtain through some preliminary experiments with the considered scene. Another potential drawback of the method is its heavy reliance on an efficient collision checker. This checker is called often and its speed is crucial. Of course, the latter observation holds for many other path-planning methods (the Randomized Path Planner, for example).

We finally attempted a theoretical analysis of the performance of the method. The analysis helps us to understand the dependence of the method on some of its parameters (e.g., the number N of nodes of the random network) and certain features of the space in which the robot moves (e.g., distance of possible paths between two nodes from the C-obstacles). Unfortunately, these parameters appear to be very difficult to estimate in advance of applying our method. Their estimation would greatly facilitate our selection of the numerical parameters for the scheme so that the failure probability of path planning queries remains below a value that we are willing to tolerate. Nevertheless, our analysis makes the nature of the dependence of the running time on the parameters quite clear, and, we believe, can pave the way for a more usable rigorous result.

In the second part of this dissertation (Chapter 5) we presented a method for computing the C-space map of obstacles used in many motion-planning algorithms including the one presented in Chapters 2 and 3. The C-space map reduces collision checking to a constant time lookup operation as discussed in Section 5.1. We compute this map as a convolution of the robot and the obstacles with the use of the Fast Fourier Transform (FFT). The method is practical for two-dimensional workspaces. It is particularly interesting that the method builds upon existing experience on the FFT and can benefit from hardware available for that transform and from other optimized software.

In the third part (Chapter 6) a problem from assembly planning was discussed. It was shown that partitioning an assembly of polygons with rigid motions in the plane is an NP-complete problem. This result was obtained by a reduction from 3-Satisfiability, a known NP-complete problem. In a broader perspective, it is interesting to find out if our complexity proof can suggest some reasonable restrictions that can be imposed on the parts of an assembly, on an assembly as whole, or on the allowed motions of its parts, so that, under these restrictions, the partitioning problem can be solved in polynomial time.

7.2 Future Work on the Planning Method

Theoretical analysis and estimation of numerical parameters

The theoretical analysis of the performance of the method deserves more attention. First of all, we would very much like to obtain an analysis that can explain one of our main experimental observations: namely, that the networks that capture sufficiently well the connectivity of high-dimensional C-spaces are small compared to the size these spaces.

Secondly, an analysis that will help estimate good values for the numerical parameters of the method, will be especially helpful. It will reduce the time needed to tailor the method to a particular set of problems. We note however that there may be an alternative to making progress in the theoretical understanding of the performance of our method in order to estimate its numerical parameters. Statistical or learning methods may be able to adaptively adjust the values of the parameters of the approach. It seems that such methods will be easier to come up with, compared to achieving a powerful theorem that will enable us to choose the parameters beforehand.

Parallel implementation of the method

A feature of the randomized planning method that should not be underestimated is its large potential for parallelism. The preprocessing phase can be efficiently implemented on a massively parallel machine (e.g., on the Connection Machine). This will permit the efficient creation of networks with tens or hundreds of thousands of nodes. Massive parallelization of the preprocessing phase will also keep network construction times low, even in cases when collision checking is expensive.

Dynamic environments

A challenging research goal would be to extend the method to dynamic scenes. One first question is: how should a network computed for a given workspace be updated if a few obstacles are removed or added? Answering this question would be useful in order to apply our method to scenes subject to small incremental changes. Such changes occur in many manufacturing (e.g., assembly) cells; while most of the geometry of such a cell is permanent and stationary, a few objects (e.g., fixtures) are added or removed between any two consecutive manufacturing operations. Similar incremental changes also occur in automatic graphic animation.

Bibliography

[Ahuactzin et al, 92] J. M. Ahuactzin, E. G. Talbi, P. Bessière, E. Mazer, Using genetic algorithms for robot motion planning, *10th European Conference on Artificial Intelligence*, John Wiley and Sons, Ltd., London, England, 1992, 671-675.

[Arkin et al, 89] E. M. Arkin, R. Connelly, J. S. B. Mitchell, On monotone paths among obstacles with applications to planning assemblies, *Proceedings of the 5th ACM Symposium on Computational Geometry*, 1989, 334-343.

[Avnaim and Boissonnat, 88] F. Avnaim, J. D. Boissonnat, *Polygon placement under translation and rotation*, Technical Report No. 890, INRIA, 1988.

[Barraquand and Ferbach, 94] J. Barraquand, P. Ferbach. Path planning through variational dynamic programming, *Proceedings of the IEEE International Conference on Robotics and Automation*, San Diego, CA, 1994, 1839-1846.

[Barraquand et al, 90] J. Barraquand, B. Langlois, J.-C. Latombe, Robot motion planning with many degrees of freedom and dynamic constraints, *Robotics Research 5*, H. Miura and S. Arimoto (Eds.), MIT Press, Cambridge, MA, 1990, 435-444.

[Barraquand et al, 92] J. Barraquand, B. Langlois, J.-C. Latombe. Numerical potential field techniques for robot path planning, *IEEE Transactions on Systems, Man, and Cybernetics*, 22(2), 1992, 224-241.

[Barraquand and Latombe, 91] J. Barraquand, J.-C. Latombe, Robot motion planning: a distributed representation approach, *The International Journal of Robotics Research*, The MIT Press, Cambridge, MA, 10(6), 1991, 628-649.

[Bilardi et al, 91] G. Bilardi, S. Hornick, M. Sarrafradeh, Optimal VLSI architectures for multidimensional DFT, *ACM Computer Architecture News*, 19(1), 1991, 45-52.

[Brooks and Lozano-Pérez, 85] R. A. Brooks, T. Lozano-Pérez, A subdivision algorithm in configuration space for findpath with rotation, *IEEE Transactions on Systems, Man, and Cybernetics*, 15(2), 1985, 224-233.

[Canny, 88] J. F. Canny, *The complexity of Robot Motion Planning*, MIT Press, Cambridge, MA, 1988.

[Canny and Reif, 87] J. F. Canny, J. Reif, New lower bound techniques for robot motion planning problems, *Proceedings of the 28th Annual Symposium on the Foundations of Computer Science*, 1987, 49-60.

[Canny and Lin, 90] J. F. Canny, M. C. Lin, An opportunistic global path planner, *Proceedings of the IEEE International Conference on Robotics and Automation*, Cincinnati, OH, 1990, 1554-1559.

[Chalou and Gini, 93] D. Chalou, M. Gini, Parallel robot motion planning, *Proceedings of the IEEE International Conference on Robotics and Automation*, Atlanta, GA, 1993, 24-25.

[Chen and Hwang, 92] P. C. Chen, Y. K. Hwang, SANDROS: A motion planner with performance proportional to task difficulty, *Proceedings of the IEEE International Conference on Robotics and Automation*, Nice, France, 1992, 2346-2353.

[Dehne et al, 89] F. Dehne, A. L. Hassenklover, J. R. Sack, Computing the configuration space for a robot on a mesh-of-processors, *Parallel Computing*, 12(2), 1989, 221-231.

[Eldracher, 94] M. Eldracher. Neural subgoal generation with subgoal graph: an approach, *Proceedings of the World Conference on Neural Networks*, San Diego, CA, 1994, 142-146.

[Faverjon and Tournassoud, 87] B. Faverjon, P. Tournassoud, A local approach for path planning of manipulators with a high number of degrees of freedom, *Proceedings of the IEEE International Conference on Robotics and Automation*, Raleigh, NC, 1987, 1152–1159.

[Faverjon and Tournassoud, 90] B. Faverjon, P. Tournassoud, A practical approach to motion planning for manipulators with many degrees of freedom, *Robotics Research 5*, H. Miura and S. Arimoto (Eds.), MIT Press, Cambridge, MA, 1990, 65-73.

[Garey and Johnson, 79] M. R. Garey, D. S. Johnson, *Computers and Intractability - A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1979.

[Giezeman, 93] G.-J. Giezeman, *PlaGeo—A library for Planar Geometry*, Technical Report, Department of Computer Science, Utrecht University, Utrecht, The Netherlands, 1993.

[Gilbert et al, 88] E. G. Gilbert, D. W. Johnson, S. S. Keerthi, A fast procedure for computing the distance between complex objects in three-dimensional space, *IEEE Transactions on Robotics and Automation*, 4(2), 1988, 193-203.

[Glavina, 89] B. Glavina, Solving findpath by combination of goal-directed and randomized search, *Proceedings of the IEEE International Conference on Robotics and Automation*, Cincinnati, OH, 1990, 1718-1723.

[Graux et al, 92] L. Graux, P. Millies, P. L. Kociemba, B. Langlois, Integration of a path generation algorithm into off-line programming of airbus panels, *Aerospace Automated Fastening Conference and Exposition*, SAE Tech. Paper 922404, 1992.

[Guibas et al, 83] L. Guibas, L. Ramshaw, J. Stolfi, A kinetic framework for computational geometry, *Proceedings of the Annual Symposium on the Foundations of Computer Science*, 1983, 100-111.

[Guibas and Seidel, 86] L. Guibas, R. Seidel, Computing convolution by reciprocal search, *Proceedings of the ACM Symposium on Computational Geometry* , Yorktown Heights, NY, 1986, 90-99.

[Guibas and Yao, 83] L. Guibas, F. Yao, On translating a set of rectangles, *Computational Geometry*, F. P. Preparata (Ed.), Advances in Computing Research, Vol. 1, JAI Press, London, 1983, 61-67.

[Gupta and Gou, 92] K. Gupta, Z. Gou, Sequential search with backtracking, *Proceedings of the IEEE International Conference on Robotics and Automation*, Nice, France, 1992, 2328-2333.

[Gupta and Zhu, 94] K. Gupta, X. Zhu, Practical motion planning for many degrees of freedom: a novel approach within sequential framework, *Proceedings of the IEEE International Conference on Robotics and Automation*, San Diego, CA, 1994, 2038-2043.

[Halberstam and Roth, 83] H. Halberstam, K. F. Roth, *Sequences*, Springer-Verlag NY, 1983.

[Halperin and Sharir, 93] D. Halperin, M. Sharir, Near-quadratic bounds for the motion planning problem for a polygon in a polygonal environment, *Proceedings of the 34th Annual Symposium on the Foundations of Computer Science*, Palo Alto, CA, 1993, 382-391.

[Haralick and Shapiro, 92] R. Haralick, L. Shapiro, *Computer and Robot Vision*, Addison Wesley, 1992.

[Hershberger and Suri, 93] J. Hershberger, S. Suri, Efficient computation of euclidean shortest paths in the plane, *Proceedings of the 34th Annual Symposium on the Foundations of Computer Science*, Palo Alto, CA, 1993, 508-517.

[Homem de Mello and Lee, 91] L. S. Homem de Mello, S. Lee editors, *Computer-aided Mechanical Assembly Planning*, Kluwer Academic Publishers, Boston, 1991.

[Hopcroft et al, 84] J. E. Hopcroft, J. E. Schwartz, 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), 1984, 76-88.

[Hopcroft and Wilfong, 86] J. E. Hopcroft, G. Wilfong, Motion of objects in contact, *International Journal Robotics Research*, 4(4), 1986, 32-46.

[Horsch et al, 94] Th. Horsch, F. Schwarz, H. Tolle, Motion planning for many degrees of freedom - random reflections at C-space obstacles, *Proceedings of the IEEE International Conference on Robotics and Automation*, San Diego, CA, 1994, 2138-2145.

[Johnsson and Krawitz, 92] L. Johnsson, R. Krawitz, Cooley-Tukey FFT on the Connection Machine, *Parallel Computing*, 18, 1992, 1201-1221.

[Joseph and Plantiga, 85] D. H. Joseph, W. H. Plantiga, On the complexity of reachability and motion planning questions, *Proceedings of the ACM Symposium on Computational Geometry*, 1985, 62-66.

[Kavraki, 93] L. Kavraki, Computation of configuration-space obstacles using the Fast Fourier Transform, *Proceedings of the IEEE International Conference on Robotics and Automation*, Atlanta, GA, 1993, 255-261. To appear in *IEEE Transactions on Robotics and Automation.*

[Kavraki and Kolountzakis, 94] L. Kavraki, M. Kolountzakis, Partitioning a planar assembly into two connected parts is NP-hard, submitted to Information Processing Letters, 1994.

[Kavraki and Latombe, 93a] L. Kavraki, J.-C. Latombe, *Complexity of partitioning a planar assembly*, Technical Report STAN-CS-93-1467, Department of Computer Science, Stanford University, CA, 1993.

[Kavraki and Latombe, 93b] L. Kavraki, J.-C. Latombe, *Randomized preprocessing of configuration space for fast path planning*, Technical Report STAN-CS-93-1490, Department of Computer Science, Stanford University, CA, September 1993.

[Kavraki and Latombe, 94a] L. Kavraki, J.-C. Latombe, Randomized preprocessing of configuration space for fast path planning, *Proceedings of the IEEE International Conference on Robotics and Automation*, San Diego, CA, 1994, 2138-2145.

[Kavraki and Latombe, 94b] L. Kavraki, J.-C. Latombe, Randomized preprocessing of configuration space for path planning: Articulated robots, *Proceedings of the IEEE/RSJ/GI International Conference on Intelligent Robots and Systems*, München, Germany, 1994.

[Kavraki et al, 93] L. Kavraki, J.-C. Latombe, R. H. Wilson, On the complexity of assembly partitioning, *Information Processing Letters*, 48, 1993, 229-235.

[Kavraki et al, 94] L. Kavraki, P. Švestka, J.-C. Latombe, M. Overmars, *Probabilistic roadmaps for path planning in high dimensional configuration spaces*, Technical Report STAN-CS-94-1519, Department of Computer Science, Stanford University, Stanford, CA, 1994.

[Koga et al, 94] Y. Koga, K. Kondo, J. Kuffner, J.-C. Latombe, Planning motions with intentions, *Proceedings of the SIGGRAPH'94*, FL, 1994, 395-408.

[Kondo, 91] K. Kondo, Motion planning with six degrees of freedom by multistrategic bidirectional heuristic free-space enumeration, *IEEE Transactions on Robotics and Automation*, 7(3), 1991, 267-277.

[Latombe, 91a] J.-C. Latombe, *Robot Motion Planning*, Kluwer Academic Publishers, Boston, MA, 1991.

[Latombe, 91b] J.-C. Latombe, A fast path planner for a car-like indoor mobile robot, *Proceedings of the 9th National Conference on Artificial Intelligence*, 1991, 659-665.

[Laumond, 87] J.-P. Laumond, Obstacle growing in a non-polygonal world, *Information Processing Letters*, 25(1), 1987, 41-50.

[Leighton, 92] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: arrays, trees, hypercubes*, Morgan Kaufmann Publishers Inc., Los Altos, CA, 1992.

[Lengyel et al, 90] J. Lengyel, M. Reichert, B. R. Donald, D. P. Greenberg, Real-time robot motion planning using rasterizing computer graphics hardware, *Proceedings of the SIGGRAPH'90*, Dallas, TX, 1990, 327-335.

[Lin and Canny, 91] M. C. Lin, J. F. Canny, A fast algorithm for incremental distance calculation, *Proceedings of the IEEE International Conference on Robotics and Automation*, Sacramento, CA, 1991, 1008-1014.

[Leven and Sharir, 87a] D. Leven, M. Sharir, An efficient and simple motion planning algorithm for a ladder moving in two-dimensional space amidst polygonal barriers, *Journal of Algorithms*, 8, 1987, 192-215.

[Leven and Sharir, 87b] D. Leven, M. Sharir, Planning a purely translational motion of a convex object in two-dimensional space using generalized Voronoi diagrams, *Discrete and Computational Geometry*, 2, 1987, 9-31.

[Lozano-Pérez, 81] T. Lozano-Pérez, Automatic planning of manipulator transfer movements, *IEEE Transactions on Systems, Man, and Cybernetics*, 11(10), 1981, 681-698.

[Lozano-Pérez, 83] T. Lozano-Pérez, Spatial Planning: A configuration space approach, *IEEE Transactions on Computers*, 32, 1983, 108-120.

[Lozano-Pérez and O'Donnell, 91] T. Lozano-Pérez, P. O'Donnell, Parallel robot motion planning, *Proceedings of the IEEE International Conference on Robotics and Automation*, Sacramento, CA, 1991, 1000-1007.

[Lozano-Pérez and Wesley, 79] T. Lozano-Pérez, M. Wesley, An algorithm for planning collision-free paths among polyhedral obstacles, *Communications of the ACM*, 22(10), 1979, 560-570.

[Natarajan, 88] B. K. Natarajan, On planning assemblies, *Proceedings of the 4th ACM Symposium on Computational Geometry*, 1988, 299-308.

[Newman and Branicky, 91] W. Newman, M. Branicky, Real-time configuration space transforms for obstacle avoidance, *International Journal of Robotics Research*, 10(6), 1991, 650-667.

[Nussbaum and Sack, 93] D. Nussbaum, J. R. Sack, Disassembling two-dimensional composite parts via translations, *International Journal of Computational Geometry*, 3, 1993, 71-84.

[Ó'Dúnlaing and Yap, 82] C. Ó'Dúnlaing, C. K. Yap, A retraction method for planning the motion of a disc, *Journal of Algorithms*, 6, 1982, 104-111.

[O'Rourke, 85] J. O'Rourke, *A lower bound for moving a ladder*, Technical Report JHU/EECS-85/20, The Johns Hopkins University, Baltimore, 1985.

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

[Overmars and Švestka, 94] M. Overmars, P. Švestka, A probabilistic learning approach to motion planning, *Workshop on the Algorithmic Foundations of Robotics*, San Francisco, CA, 1994 (to appear).

[Pollack et al, 88] R. Pollack, M. Sharir, S. Sifrony, Separating two polygons by a sequence of translations, *Discrete and Computational Geometry*, 3, 1988, 123-136.

[Pratt, 85] W. Pratt, A pipeline architecture for image processing and analysis, *IEEE Computer Architecture for Pattern Analysis and Image Database Management*, Miami, 1985, 516-520.

[Quinlan, 94] S. Quinlan, Efficient distance computation between non-convex objects, *Proceedings of the IEEE International Conference on Robotics and Automation*, San Diego, CA, 1994, 3324-3330.

[Ramirez, 85] R. Ramirez, *The FFT fundamentals and concepts*, Prentice Hall, Englewood Cliffs, NJ, 1985.

[Reif, 79] J. Reif, Complexity of the mover's problem and generalizations, *Proceedings of the 20th Annual Symposium on the Foundations of Computer Science*, 1979, 421-427.

[Reif and Sharir, 85] J. Reif, M. Sharir, Motion planning in the presence of moving obstacles, *Proceedings of the 26th Annual Symposium on the Foundations of Computer Science*, 1985, 144-154.

[Reeves, 84] A. P. Reeves, Parallel computer architectures for image processing, *Computer Vision, Graphics and Image Processing*, 25(1), 1988, 68-88.

[Schwartz and Sharir, 83a] J. T. Schwartz, M. Sharir, On the piano movers' problem I: The case of a two-dimensional rigid polygonal body moving amidst polygonal barriers, *Communications Pure and Applied Mathematics*, 36, 1983, 345-398.

[Schwartz and Sharir, 83b] J. T. Schwartz, M. Sharir, On the piano movers' problem II: General techniques for computing topological properties of real algebraic manifolds, *Advances in Applied Mathematics*, Academic Press, 4, 1983, 298-351.

[Sharir, 87] M. Sharir, Efficient algorithms for planning purely translational collision-free motion in two and three dimensions, *Proceedings of the IEEE International Conference on Robotics and Automation*, 1987, 1326-1331.

[Schwarzkopf, 93] O. Schwarzkopf, Computing convolutions on mesh-like structures, *Proceedings of the 7th International Parallel Processing Symposium*, 1993, 1083-1087.

[Snoeyink and Stolfi, 93] J. Snoeyink, J. Stolfi, Objects that cannot be taken apart with two hands, *Proceedings of the 9th ACM Symposium on Computational Geometry*, 1993, 247-256.

[Švestka, 93] P. Švestka, *A probabilistic approach to motion planning for car-like robots*, Technical Report RUU-CS-93-18, Department of Computer Science, Utrecht University, Utrecht, The Netherlands, 1993.

[Švestka and Overmars, 94] P. Švestka, M. Overmars, *Motion planning for car-like robots using a probabilistic learning approach*, Technical Report RUU-CS-94-33, Department of Computer Science, Utrecht University, Utrecht, The Netherlands, 1994.

[Swarztrauber, 87] P. Swarztrauber, Multiprocessor FFTs, *Parallel Computing*, 5(1), 1987, 197-210.

[Tong and Swarztrauber, 91] Ch. Tong, P. Swarztrauber, Ordered FFTs on a massively parallel hypercube multiprocessor, *Parallel Computing*, 1991, 50-59.

[Toussaint, 85] G. T. Toussaint, Movable separability of sets, *Computational Geometry*, Elsevier Science Publishers, North Holland, 1985.

[Wilson, 92] R. H. Wilson, *On geometric assembly planning*, PhD Thesis, Technical Report STAN-CS-92-1416, Department of Computer Science, Stanford University, CA, 1992.

[Wilson et al 93] R. H. Wilson, L. Kavraki, T. Lozano-Perez, J.-C. Latombe, Two-handed assembly sequencing, to appear in *International Journal Robotics Research*; also available as Technical Report STAN-CS-93-1478, Department of Computer Science, Stanford University, CA, 1993.

[Wilson et al, 92] R. H. Wilson, J.-C. Latombe, T. Lozano-Pérez, *On the complexity of partitioning an assembly*, Technical Report STAN-CS-92-1458, Department of Computer Science, Stanford University, CA, 1992.

[Zhu and Gupta, 93] X. Zhu, K. Gupta, *On local minima and random search in robot motion planning*, Technical Report, Simon Fraser University, Burbany, BC, Canada, 1993.

[Zhu and Latombe, 91] D. Zhu, J.-C. Latombe, New heuristic algorithms for efficient hierarchical path planning, *IEEE Transactions on Robotics and Automation*, 7(1), 1991, 9-20.