AN OBSTACLE-BASED PROBABILISTIC ROADMAP METHOD FOR PATH PLANNING

A Thesis

by

YAN WU

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

August 1996

Major Subject: Computer Science

AN OBSTACLE-BASED PROBABILISTIC ROADMAP METHOD

FOR PATH PLANNING

A Thesis

by

YAN WU

Submitted to Texas A&M University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Approved as to style and content by:

Nancy M. Amato (Chair of Committee)	Jeffrey C. Trinkle (Member)
Peter Stiller (Member)	Richard A. Volz (Head of Department)

August 1996

Major Subject: Computer Science

ABSTRACT

An Obstacle-Based Probabilistic Roadmap Method
for Path Planning. (August 1996)

Yan Wu, B.E., Xian Jiaotong University;
M.E., Xian Jiaotong University

Chair of Advisory Committee: Dr. Nancy M. Amato

This thesis presents a new obstacle-based probabilistic roadmap method for motion planning for many degree of freedom robots that can be used to obtain high quality roadmaps even when the robot's configuration space is crowded. The main novelty in this approach is that roadmap candidate points are chosen on the constraint surfaces corresponding to obstacles in the workspace. As a consequence, the roadmap is likely to contain difficult paths, such as those traversing long, narrow passages in the robot's configuration space. The approach can be used for both collision-free path planning and for planning contact tasks. A path planner based on this approach is implemented for planar articulated robots in a two-dimensional workspace with polygonal obstacles. Experimental results with various types of robots and a range of environments show that well connected roadmaps are generated for most environments, even when the number of nodes in the roadmap is small. After the roadmap is built during preprocessing, many difficult path planning operations are carried out in less than a second.

To My Wife

ACKNOWLEDGMENTS

I had my first contact with my advisor, Dr. Nancy Amato, in the spring of 1995, when I took her algorithms class. My first impression proves itself as time goes by: she is an optimistic, energetic professor who is always prepared to answer questions and provide advice and original thoughts, and who is always willing to listen to her students. During the semester, she became my advisor and together, we launched the mission which gradually shaped itself into the path planning method presented in this thesis.

A few words cannot fully describe my deep appreciation to her. In fact, she is the person who first came up with the new idea and then went through almost every detail of it. As my advisor, she was always there for me no matter what questions I brought to her, either difficulties with the project or just a spelling query. Her vision and ambition in research enabled me to believe in what I was doing, even in the very early stages of the project, and her optimistic personality gave me many enjoyable memories during the last one and a half years. I believe those memories will stay with me for many years. My committee members, Dr. Trinkle and Dr. Stiller, whose expertise in robotics helped me finish this project, also provided many valuable comments and suggestions. I would like to thank them for the time and effort they spent on my thesis.

I would also like to take this chance to thank the other members of the robotics research group in the Department of Computer Science of Texas A&M University: David Stewart (Post-Doc), Soonlin Yeap, Wookho Son, and Li Han. Their work showed me different aspects of the area and I enjoyed the time we spent together. My thanks are also due to Robert Wilkinson, who made the expedition with me on the parallel implementation of the path planning method in the spring of 1996.

During the last one and a half years when I was concentrating on my work towards this thesis and for many years before that, my family has always been my best source of support and a place where I could turn to for encouragement. My parents taught me how to face different challenges in my life and my little brother shared with me many fights as well as joys. My last words, but not the least, are for my wife, Wenye Yang, who always urges me, as well as herself, to pursue higher goals. Without her love, her understanding, and her effort, this thesis may not have been possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

A. Automatic Path Planning

Automatic path planning is one of the most important yet unsolved areas in robotics research. It has applications in many areas, such as industrial assembly design, virtual reality systems and computer-aided design systems. The goal of robot path planning is to find a sequence of configurations (positions and orientations) to move a robot from a start configuration to a goal configuration without colliding with the obstacles or objects in the workplace of the robot (called the workspace). The difficulty of the problem lies in the computational complexity which has been proved to be at least PSPACE-hard in most cases. In other words, the problem is generally intractable. Many different path planning methods have been proposed throughout the last several decades, but most of them are impractical since they are computationally infeasible except for some restricted cases, e.g., when the robot has a small number of degrees of freedom (dof); the dof of a robot is the minimum number of parameters needed to specify a configuration of the robot [20, 27].

On the other hand, the rapid development of computer technology provides a promising future for achieving better solutions to this problem. In the past ten years, both the processor speed and main memory size have been improved by a factor of nearly 100. However, even with such dramatic growth the computational ability of modern computers is still inadequate to solve many path planning problems. In particular, since even simple path planning problems are known to be PSPACE-hard,

---

The journal model is *IEEE Transactions on Robotics and Automation.*

it has become apparent that with existing technology, the complexity of the problem can only be overcome through the use of *probabilistic* or *heuristic* methods (as opposed to *complete* methods which are guaranteed to find a solution or determine that none exists).

The following is a simple example illustrating how the computational complexity grows rapidly with the number of degrees of freedom of the robot. For a 6-joint manipulator (6 dof), a straight forward bitmap representation of its configuration space (C-space) with a resolution of 9° (360° for each joint) needs 512MB of memory. For a 7-joint manipulator (7 dof), it becomes 20GB which is not practical even on a state-of-the-art supercomputer. Some of the manipulators being proposed for space use have more than 17 degrees of freedom.

Several promising heuristics for path planning have been proposed. Among them, the Randomized Path Planner (RPP) of Barraquand and Latombe [5] is a potential field method that uses random walks to attempt to escape local minima. Although this method has been shown to work well for many dof robots, there exist simple situations in which it performs poorly (i.e., does not find a solution even when one is known to exist). Researchers have proposed various potential functions (e.g., [4]) and other techniques, such as learning [16], for escaping local minima. In general, potential field methods can be quite effective when the configuration space (C-space) is relatively uncluttered, that is, when most of the C-space is collision free. However, they have not been as successful for planning when C-space is crowded.

Recently, Kavraki and Latombe [22] proposed a new path planning method which uses randomization during preprocessing to construct a graph (often called a *roadmap* [27]) whose nodes correspond to collision-free configurations of the robot, and in which two nodes are connected by an edge if a path between the two corresponding configurations can be found by a simple, 'local' planner. Briefly, roadmap candidate nodes

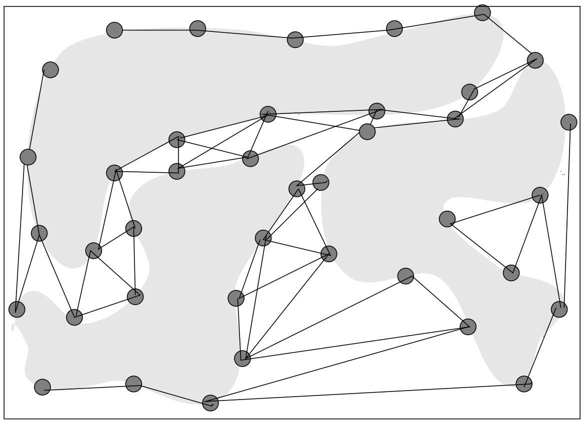


Fig. 1. A roadmap in C-space that might be obtained when candidate nodes are uniformly distributed on constraint surfaces.

are selected according to a uniform distribution over C-space and those found to be collision-free are retained. Next, connections between candidate pairs are attempted with a simple planner, and then 'difficult' regions of C-space are identified and the roadmap is enhanced in these regions with additional roadmap nodes. Planning involves connecting the initial and goal configurations to the roadmap, and then finding a path in the roadmap between these two connection points. Planning can be done very fast in many situations. However, long, narrow passages between obstacles in C-space (called C-obstacles) can be difficult to find. An approach for low dof robots that bears strong similarities to [22] was proposed by Overmars [33] and was further extended to many dof through the use of learning techniques [34].

A more detailed discussion of different path planning methods is presented in Chapter II.

B. Contributions of This Thesis

In this thesis, we develop and experimentally evaluate a new path planning algorithm, namely, the *obstacle-based probabilistic roadmap method* (OBPRM), for robots with many degrees of freedom.

The general approach follows traditional roadmap methods: during preprocessing a graph, or roadmap, is built in C-space. Planning consists of connecting the initial and goal configurations to the roadmap, and then finding a path in the roadmap between these two connection points. The main novelty in our approach is a new method for generating roadmap candidate points. In particular, we attempt to generate candidate points randomly distributed on the surface of each C-obstacle. (See Figure 1.) Using this approach, high quality roadmaps can be obtained even when C-space is crowded.

Experimental results with planar articulated robots with 6–20 dof, a planar free-flying robot with 8 dof and an 8-dof planar articulated robot with fingers show that, after preprocessing, many difficult path planning operations can be carried out in a fraction of a second.

Our approach can be applied to some important situations that have so far not been satisfactorily dealt with by heuristic methods:

- Paths through long, narrow passages in a crowded C-space can be found since they are likely to appear in our roadmap.

- Our method can be used for planning contact tasks since the roadmap can be built on the constraint surfaces.

The previous work most closely related to ours is the path planning method of Kavraki and Latombe [22]. The main difference in our approach is that we generate points on C-obstacle surfaces. We find these roadmap candidate points using binary search, which is more costly than in [22] where candidate points are randomly selected according to a uniform distribution over C-space. However, our experimental results show that this additional computation in the node generation phase is negligible in comparison to the total preprocessing time, which in both methods is dominated by the roadmap connection phase. Finally, in some sense, one can view our work as extending the strategy of [22] to contact tasks.

C. Outline

In Chapter II, we first give a formal definition of robot path planning and other basic terms used in this thesis. We also discuss the complexity of the problem and provide a survey of previous path planning methods. Chapter III describes the obstacle-based probabilistic roadmap (OBPRM) method in detail. Implementation issues are discussed in Chapter IV and experimental scenarios, results and analysis are presented in Chapter V. We conclude in Chapter VI with a discussion of the strong points and weak points of this method and directions for further research.

CHAPTER II

OVERVIEW OF PATH PLANNING METHODS

A. Problem Definition

A *robot* is a man-made machine equipped with actuators and sensors and controlled by a computing unit for performing certain tasks in an environment called the *workspace*. The *workspace* consists of physical objects called obstacles and free space in which the robot can move. A *path* is a sequence of robot configurations connecting two specified configurations of the robot. Each configuration on the *path* should be collision-free, which means the robot does not collide with any of the obstacles in the workspace.

The basic *path planning* (or *motion planning*) problem can be defined as follows. Let $\mathcal{A}$ be a *robot* which is either a single rigid object or a collection of rigid objects connected by rotary or prismatic joints, each of which can move independently. $\mathcal{A}$ can move in a two or three-dimensional workspace $\mathcal{W}$. Let $\mathcal{B}$ be a collection of fixed rigid objects (called *obstacles*) distributed in $\mathcal{W}$ which are known to the robot. Given an initial position and orientation $\mathcal{S}$ and a goal position and orientation $\mathcal{G}$ of $\mathcal{A}$ in $\mathcal{W}$, the path planning problem is to determine whether there exists a continuous sequence of positions and orientations of $\mathcal{A}$ (a path), starting at $\mathcal{S}$ and ending at $\mathcal{G}$ that avoids collision with $\mathcal{B}$, and if so, to find such a sequence.

The *degrees of freedom* (*dof*) of a robot is the minimum number of parameters needed to specify a configuration of the robot. For example, if the robot is a rigid rectangle in a plane, we need three parameters (two for position and one for orientation) to specify a configuration. Thus, this robot has three dof.

The set of all possible configurations of the robot is called the *configuration space*

(C-space) of the robot [28, 29]. Let r_i be the range of the ith parameter used to specify a configuration of robot $\mathcal{A}$; for a rotary joint, r_i is $[0, 2\pi)$, and for a prismatic joint, r_i is $[l_1, l_2]$ where $l_1, l_2 \in \mathbf{R}$. Assuming the dof of the robot $\mathcal{A}$ is d, the C-space of $\mathcal{A}$ is defined as follows:

$$\mathcal{C}(\mathcal{A}) = r_1 \times r_2 \times \ldots \times r_d.$$

The number d is called the dimension of $\mathcal{C}(\mathcal{A})$.

Notice that if the ith parameter of robot $\mathcal{A}$ is a rotary joint, then r_i is not a Euclidean space because, for example, the distance between $\pi/4 \in r_i$ and $2\pi - \pi/4 \in r_i$ is $\pi/2$, instead of $3\pi/2$ which is the Euclidean distance. Hence $\mathcal{C}(\mathcal{A})$ is not $\mathbf{R}^d$ (Euclidean space) if robot $\mathcal{A}$ has rotary joints. A local transformation can avoid this inconvenience and make the Euclidean distance metric available for $\mathcal{C}(\mathcal{A})$. Details are discussed in Chapter IV.

After we define the C-space of $\mathcal{A}$, each configuration of $\mathcal{A}$ is mapped to a point in $\mathcal{C}(\mathcal{A})$ which can be represented by a d-tuple $\langle x_1, x_2, \ldots, x_d \rangle$, where $x_i \in r_i$. All the points in C-space can be classified into two categories: the forbidden set in which each point represents a configuration for which there is a collision between $\mathcal{A}$ and some obstacle(s) in workspace or a configuration that violates $\mathcal{A}$'s mechanical constraints (a self-collision), and the free set in which each point represents a configuration of $\mathcal{A}$ that is feasible and free of collision with obstacles. Also, each obstacle in the workspace can be mapped to a subset of the forbidden set which we call a *C-obstacle*. The forbidden set is the union of all C-obstacles and the pseudo C-obstacle created by the mechanical constraints of the robot.

The path planning problem can thus be defined as finding a path in C-space $P = \{p_1, p_2, \ldots, p_n\}$, in which each p_i is a point in the free set of C-space, p_1 is the start configuration, p_n is the goal configuration, and p_{i-1} can be connected to p_i by a 'local' path planner (where locality is defined in terms of some chosen distance metric).

We assume that the environment including the information about the obstacles in the workspace and the description and position of the robot is known. We also assume that the environment is static and will not change during the planning operation. Later we remove these assumptions and discuss how the path planning methods can be extended for these cases.

B. General Classes of Path Planning Methods

A large number of methods have been developed for path planning; most of them work for specific cases and have certain limitations. In general, most path planning methods can be categorized into a few general classes (see Hwang and Ahuja [20] and Latombe [27]).

1. Roadmap methods

Roadmap (or skeleton) methods represent the free C-space by a *roadmap* (or *network*). In particular, the idea is to extract the connectivity information of the free C-space into a graph of one-dimensional curves. Then, after connecting the start and goal points to the roadmap, planning becomes a graph search problem. The roadmap must contain all topologically feasible paths in C-space to make the planning algorithm complete. Although for high-dimensional C-spaces the complexity of finding all feasible paths makes the computation of a complete roadmap infeasible, probabilistic methods can generate acceptable results in many cases. Some well-known skeletons are the *visibility graph* [2], the *Voronoi diagram* [3], Canny's *silhouette* [6], and the *subgoal network* [10].

Path planning for a planar d-link manipulator in an obstacle-free circular region is studied by Hopcroft *et al.* [19]. The algorithm first moves the arms to a *normal form* and then changes the orientation of each link using only *simple motions*. A *simple motion* is defined such that up to four joint angles can be changed at one time. In a *normal form*, all the joints up to i lie on a radius of the bounding circle and the rest of the joints touch the circle. The algorithm is a roadmap method, with the normal form and simple motion being the roadmap node and local planner, respectively. This algorithm has a overall complexity of $O(d^3)$ and is a polynomial-time method.

A subgoal network method was first proposed by Faverjon and Tournasoud [15]. The algorithm divides the C-space into cells and assigns each cell a probability that a local algorithm would succeed in that cell. Initially, the probabilities of the cells are equal and then they are updated using a local algorithm. An A^* search is used to find a sequence of regions with the highest probabilities. A potential field local method is then applied to the sequence of cells. If the local method finds a path within a cell, the probability of that cell is increased, otherwise it is decreased. The search is done repeatly until a path is found or the planning is terminated. The off-line algorithm shows a very high complexity when the manipulator has 8 dof.

An efficient and resolution-complete motion planner is presented by Chen and Hwang [10]. Techniques such as hierarchical and multi-resolution subgoal networks, bidirectional search, and the ICORS paradigm are used to achieve efficiency and completeness. This algorithm can solve easy problems fast and harder problems with a gradual increase in computation time. It has an average computation time of 10 minutes on a 17-MIPS computer for a 6-dof manipulator in a polyhedral environment.

2. Cell decomposition methods

The cell decomposition approach is to partition the free C-space into a set of cells and to compute the adjacency relationships between those cells. Planning involves locating the cells that contain the start and goal configurations and then finding a sequence of connected cells between them. The computation of the cells can be either obstacle-dependent or obstacle-independent. In obstacle-dependent decomposition, the obstacle boundaries are used to form the cell boundaries so that the shape and size of the cells are irregular. If obstacle-independent decomposition is used, then configuration space is partitioned, for example, by a grid or a tree and each cell is tested for collision. If a cell is not collision-free, then decomposition on this cell continues or a finer global partitioning is used.

The first resolution-complete planner for general manipulators was proposed by Lozano-Pérez [30]. In the planner the C-obstacles are computed using the needle method and the free C-space is decomposed into regions. A graph is constructed to specify the connectivity of the regions and A* search is used to find the shortest path. The C-obstacle computation begins with the first joint angle and continues to the last one. For polyhedral obstacles and manipulator links, it takes $O(r^{d-1}(mn)^2)$ time to build the C-obstacles, where each joint angle θ_i is quantized into r values, d is the number of degrees of freedom, m and n are the numbers of faces and edges of the manipulator links and obstacles, respectively. The main difficulty of this method is that it has exponential time and space requirements due to its exhaustive nature.

The methods developed by Herman [18] and Faverjon [14] use swept volumes to compute octree representations of C-space and A* search to find a path. These methods only find solutions in relatively uncluttered environments, i.e., the free C-space is not very narrow. Paden *et al.* [35] use a Jacobian-based method to compute the C-obstacles and brush fire search on a 2^d-tree to find a collision-free path. Kondo [26] uses multiple-heuristic search strategies on a rectangular grid representing C-space; this method has a resolution limitation due to the huge memory requirement.

3. Potential field methods

Potential field methods are based on the construction of a function of configuration vectors, called the *potential function*, such that the value of the function is monotonicly increasing toward obstacles and monotonicly decreasing toward the goal configuration where it achieves a minimum value. Planning begins at the start configuration and follows the decreasing gradient of the potential function until the goal configuration is reached. The main difficulty with this method is defining a potential function for which the global minimum is also the only local minimum. In many cases potential field methods are more suitable for constructing local planners (used to connect nearby points in C-space) than global planners.

Many potential functions have been proposed. Among them, the artificial potential repulsion approach proposed by Khatib [24], an energy interpretation of the potential function by Newman and Hogan [32], and the superquadric potential function by Khosla and Volpe [25] are local, short-term planners which can be used in real-time control robots.

The Randomized Path Planner (RPP) of Barraquand and Latombe [5] is a potential field method that uses random walks to attempt to escape local minima. Although this method has been shown to work well for many dof robots, it has not been as successful for planning in cluttered C-space.

4. Mathematical programming

These methods represent the obstacle boundaries by systems of inequalities, and hence transform the motion planning problem into a nonlinear optimization problem to find a curve between the start and goal configurations minimizing a certain scalar quantity. Normally a numerical method is required to generate an optimal solution.

Maciejewski and Klein [31] developed a local-collision avoidance algorithm for redundant (dof > 6) manipulators. The goal of the planner is to make the end effector follow a specified trajectory which is a collision-free path for a point object. The values of joint variables are found using a generalized inverse from the Jacobian equation. Then the joint variables are modified to move the point of the manipulator closest to obstacles away from the obstacles while the end effector is following the trajectory. Chen and Vidyasagar [11] developed an optimal trajectory planner for planar n-link manipulators. A grid of points in C-space is used to detect collisions with the obstacles. The collision points are grouped and approximated by ellipses. The equations of these ellipses are then used as constraints in the optimal-control formulation, which is solved numerically.

C. The Complexity of Path Planning Problems

A common approach to analyze the complexity of a problem is to decide if the problem belongs to a class of problems by showing its equivalence to a problem already in that class. P, NP and PSPACE are among the most frequently used classes for this purpose.

- The class P corresponds to the problems that can be solved by a polynomial time algorithm on a deterministic Turing machine.

- The class NP corresponds to the problems that can be solved by a polynomial time algorithm on a nondeterministic Turing machine, i.e., a machine that has an infinite number of processors to verify in parallel all possible branches of the search tree of the problem.

- The class PSPACE corresponds to the problems that can be solved by an algorithm using polynomial space on a nondeterministic Turing machine.

It has been proved that P $\subseteq$ NP $\subseteq$ PSPACE. Most computer scientists believe that a strict inclusion relation is true, but this is still an open problem.

Let S be a class of problems (e.g., NP or PSPACE). A problem $\mathcal{P}$ is *hard* for S, or S-hard, if every problem in S is reducible to $\mathcal{P}$ by a polynomial time transformation. This means $\mathcal{P}$ is at least as difficult to solve as any problem in S. If $\mathcal{P}$ is both in S and S-hard then $\mathcal{P}$ is said to be *complete* for S, or S-complete. A more detailed discussion about hardness can be found in Garey and Johnson [17].

Therefore, if a problem is known to be in NP-hard or PSPACE-hard, then it is reasonable to believe that the time required to solve this problem is inherently exponential in the size of the problem. In other words, this problem is considered intractable.

Lower bounds have been established on the complexity of several typical motion planning problems. In particular, it is known that most path planning problems are PSPACE-hard, indicating that heuristic methods should be used.

Polyhedral world. The planning problem for a robot with polyhedral parts connected by joints located at vertices in a 3D workplace of polyhedral obstacles is PSPACE-hard [36].

Planar arm among polygonal obstacles. The planning problem for a planar arm robot with arbitrarily many non-extensible links serially connected by rotary joints in a plane with polygonal obstacles is PSPACE-hard [21].

Shortest paths in space among polyhedra. The problem of finding the shortest path between two given points in space that avoids a collection of polyhedral obstacles is NP-hard. In the plane this problem has a polynomial time solution [7].

CHAPTER III

OBSTACLE-BASED PROBABILISTIC ROADMAP METHOD

A. The Two-Step Method for Path Planning

The obstacle-based probabilistic roadmap method (OBPRM) we advocate is basically a two-step subgoal network approach. This strategy is attractive because it can provide a relatively small response time after the initial preprocessing of the environment. It has the advantage of avoiding local minima which are hard to overcome in potential field methods. Using a 'learning' loop after a short preprocessing phase, this method becomes adaptive and can be used in problems involving dynamic environments.

Step 1: Preprocessing. This step includes roadmap candidate node generation and roadmap connection. First a set of roadmap candidate nodes is generated on the C-obstacle boundaries (more details are given below). Next, the candidate nodes are interconnected to form a network (roadmap) using a local planner (called the *connection planner*). This stage also includes the processing on the roadmap such as the transformation of the roadmap from a directed to an undirected graph and analysis of its connectivity. After preprocessing, a roadmap is constructed and is ready for planning.

Step 2: Planning queries. This step involves finding a feasible path between the given start and goal configurations. First, a local planner (called the *query planner*) is used to connect the start and goal configurations to the same connected component of the roadmap, if possible. If both configurations are successfully connected to one component of the roadmap, a graph search is performed on that component to find a path between the connection points. Then a path is generated by combining the

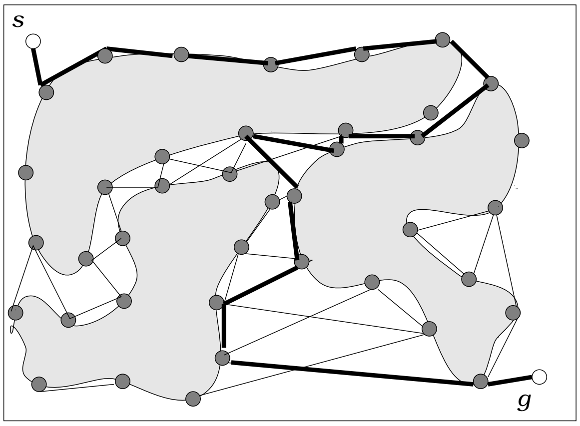


Fig. 2. Roadmap planning method in C-space.

subpath from the start configuration to its connection point in the roadmap, the subpaths between each pair of nodes on the path in the roadmap (reconstructed with the connection planner) and the subpath from the goal connection point to the goal configuration.

Figure 2 shows an example of path planning in a 2D C-space using the obstacle-based probabilistic roadmap method. s and g are start and goal configurations (white nodes). They are connected to the roadmap and a path is found by searching the roadmap (the bold edges in graph).

B. Preprocessing

The tasks required to build the roadmap are generating the roadmap candidate nodes and connecting the candidates to form the roadmap. The description of these tasks below is for a general many dof robot.

1. Generating candidate nodes

Let d be the number of degrees of freedom of the robot, n_{obs} be the number of obstacles and let $S = \{s_i | 1 \leq i \leq n_{obs}\}$ be the obstacles in the workspace. Without loss of generality, assume that C-space has dimension d, and let $S' = \{s'_i | 1 \leq i \leq n_{obs}\}$ denote the set of C-obstacles corresponding to the obstacles in the workspace.

In this phase, we generate a set V of candidate roadmap nodes, each of which corresponds to a point in C-space. The general strategy of the node generation process is to construct a set V_i of candidate nodes for each obstacle s_i such that each $p \in V_i$ lies on the surface of s'_i and is not contained in the interior of any $s'_i \in S'$. The set of roadmap candidate nodes is the union of the candidate sets computed for each obstacle, i.e., $V = \cup_i V_i$.

We now consider how to compute the candidate set for an obstacle. To obtain a high quality roadmap, we would like the nodes to be uniformly distributed (according to some metric in C-space) on the constraint surfaces that delineate the robot's free space. To simplify the exposition, in the following we assume the metric is the Euclidean distance in C-space; this and some other possible metrics are mentioned in Chapter IV. The easiest way to generate a candidate set for an obstacle $s_i \in S$ is to first compute a set of points on the surface of s'_i, and then discard those points found to be internal to any other C-obstacle. The process is sketched below for $s_i \in S$.

1. Generate m_i points $p_1, p_2, \ldots, p_{m_i}$ on the surface of s'_i, and place these points in V_i.

2. Remove all points from V_i that lie in the interior of some $s'_j \in S', j \neq i$.

Briefly, the points in Step 1 are identified by binary searches which perform collision detection checks at each step. Ideally, the number m_i of points generated in

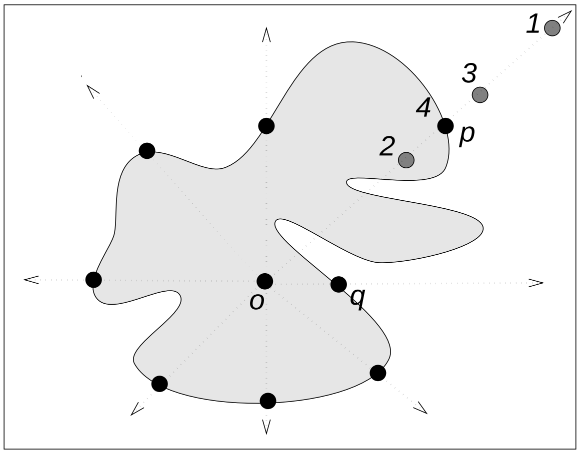


Fig. 3. Generating points uniformly on a C-obstacle in two-dimensional C-space with $m = 8$.

Step 1 will depend upon the size and shape of s'_i, and thus may vary from C-obstacle to C-obstacle. For simplicity, we choose the same m_i for each C-obstacle s'_i in our implementation. In Step 2, collision detection is used to determine which points are to be removed from V_i. We next describe the point generation process in more detail. The collision detection algorithm is discussed with other implementation issues in Chapter IV.

a. Finding points on the surfaces of C-obstacles

We now consider how to generate m points on the surface of a C-obstacle $s' \in S'$. For now, assume that we know the number of points we wish to obtain; the issue of determining a reasonable value for the parameter m is discussed in Chapter IV. Ideally, as mentioned above, these m points should be uniformly distributed on the surface of s'. However, since we wish to avoid the costly computation of the constraint surfaces, we propose using the randomized method outlined below to generate the points.

1. Determine a point o (the origin) inside s'.

2. Select m rays with origin o and directions randomly distributed in C-space.

3. For each ray identified in Step 2, use binary search to determine a point on the boundary of s' that lies on that ray.

An example of the method for a two-dimensional C-space and $m = 8$ is shown in Figure 3.

In Step 1, an internal point o can be determined by selecting any configuration in which the robot is known to collide with the obstacle s. This is trivially done for a free-flying robot, and requires only slightly more effort otherwise. Of course, to obtain as uniform a distribution as possible, one would like to select the origin somewhere near the center of the C-obstacle. A simple heuristic is to select a configuration in which a central point of the obstacle (e.g., an average of the coordinates of the obstacle's vertices) coincides with a central portion of the robot.

The rays in Step 2 can be selected using a simple random number generator. Consider a ray r identified in Step 2. The binary search in Step 3 proceeds by continually narrowing the region on r known to contain a surface point of s'. Initially, we know that the origin o is internal to s', and assume that we also know a point external to s', e.g., the point labeled 1 in Figure 3. Next, the segment determined by these internal and external points is divided at its midpoint, e.g., the point labeled 2 in the figure. The search then proceeds recursively in the sub-segment which has one internal and one external endpoint. The process terminates when a point on the boundary of s' is found and/or the minimum step size is reached, e.g., the point labeled 4 in the figure. Before beginning the binary search we must find a point on r known to be external to s'. This can be done, for example, by choosing some point known to be at the edge of the relevant portion of C-space, or by iteratively testing points of increasing distance from the origin. Note that it is possible no external point is found (and thus also no surface point for this ray), either because r may not pierce s', or because our search for such a point is not successful.

b. Potential problems and optimizations

There are several situations in which the point generation scheme described above may not yield a (close to) uniform distribution on the surface of the C-obstacle. Clearly, the shape of the C-obstacle and the selected origin both have a great influence on the quality of the resulting distribution. For example, if the C-obstacle is long and narrow, then no choice of origin will yield a uniform distribution. In particular, the surface points that are farthest from the origin will have a much larger distance from their neighbors than those that are close to the origin. An optimization that can be applied in this case is to generate more points on the surface between points that are considered too far apart. An example is shown in Figure 4. Another heuristic is to compare the surface normal of a point with those of its neighbors: if they are very different, then the constraint surface must curve between them and we should generate more points. Another option is to select multiple 'origin' points inside a C-obstacle. This will reduce the risk of locating the origin point at a 'bad' area of the C-obstacle.

Another difficulty arises if the C-obstacle boundary has 'folds', i.e., a ray from the origin may intersect the boundary multiple times. In this case, one cannot determine if a good distribution has been obtained by inspecting the Euclidean distances between neighboring nodes. For example, in Figure 3, although the Euclidean distances between neighboring nodes are fairly uniform, the boundary distance between p and q is much larger than between other neighboring nodes. Unfortunately, this situation is difficult to identify without computing the C-obstacle. However, our method was developed specifically to avoid this expensive computation. The heuristic mentioned

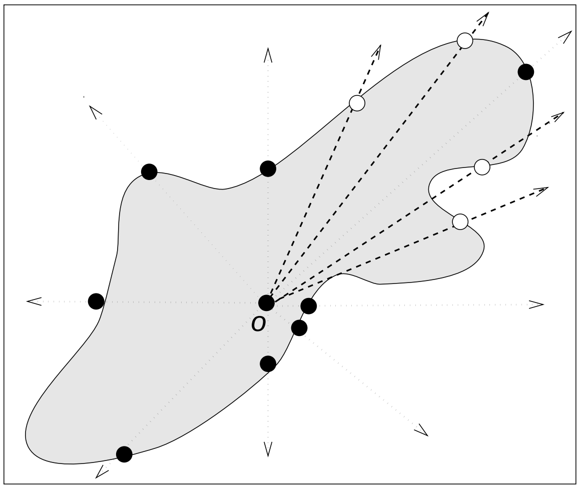


Fig. 4. To get a better distribution on 'oblong' C-obstacles, additional points (white) are generated near points with large nearest neighbor distance.

above of comparing the surface normals of neighboring points can be used to identify some of these situations, although admittedly it is not guaranteed to find them all. Finally, even if one knows *a priori* that the boundary has 'folds', it is not clear how this information can be used to obtain a better distribution without performing a significant amount of computation.

c. Randomized node generation algorithm and complexity

Figure 5 shows the pseudo-code of the algorithm for node generation using randomization and binary search. Assume the complexity of the collision detection algorithm is $C(n_{dof}, n_{obs})$, where n_{dof} is the dof of the robot and n_{obs} is the number of obstacles in the workspace. The complexity of the node generation algorithm is:

$$T_{gen-node} = O\Big(n_{obs} * m * C(n_{dof}, n_{obs})\Big)$$

where m is the number of directions generated for each obstacle.

The experimental results show that the computation time needed to generate

```
Algorithm  NodeGeneration (RobotSystem, m, NodeList)
Input:  RobotSystem -- robot and obstacle definition
        m -- number of directions per obstacle
Output: NodeList -- list of candidate nodes generated

Begin
  for each obstacle O(i) in RobotSystem do
    GenNode (O(i), List(i), m)
  end /* for */
  add each List(i) onto NodeList
End.


Procedure  GenNode (Obstacle, m, List)
Input:  Obstacle -- obstacle definition
        m -- number of directions per obstacle
Output: List -- list of candidate nodes generated

Begin
  determine a point o inside Obstacle
  for i = 1 to m do
    find a direction using the random number generator
    from o find a point c along the direction which is on
      the boundary of Obstacle using binary search
    check c for collision with other obstacles
    if c is a collision-free node, add it into List
  end /* for */
End.
```

Fig. 5. The randomized node generation algorithm.

candidate nodes is orders of magnitude less than the roadmap connection time. Hence the use of more complicated techniques to generate an 'elite' set of candidates is not only affordable, but also very desirable because the quality of the candidate nodes will directly affect both the computation time of the roadmap connection phase and the quality of the roadmap constructed.

2. Roadmap connection

We now consider how to connect the candidate nodes $V = \cup_i V_i$ to create the roadmap. The basic idea is to use simple, fast, local planners to connect pairs of roadmap candidate nodes. To save space, the local planner should be deterministic so the paths found in this stage don't have to be recorded (since they can be re-generated quickly). After the connections are made, the connected components in the roadmap are identified, e.g., by breadth-first search.

a. Connection strategy

Ideally, the roadmap will include paths through all corridors in C-space. The degree to which this goal can be met depends upon a number of factors: the number and distribution of the candidate nodes, the effectiveness of the simple planner, and the number of connections attempted for each candidate node. Thus, a trade-off exists between the quality of the resulting roadmap and the resources (computation and space) one is willing to invest in building it. Also, the amount of preprocessing required to achieve a given degree of connectivity will depend greatly on the complexity of the C-obstacles and the robot's free-space.

Clearly, the method used to determine adjacencies in the roadmap will depend upon the intended use of the roadmap. In particular, different planners will be needed for path and manipulation planning. However, in all cases the planners should be

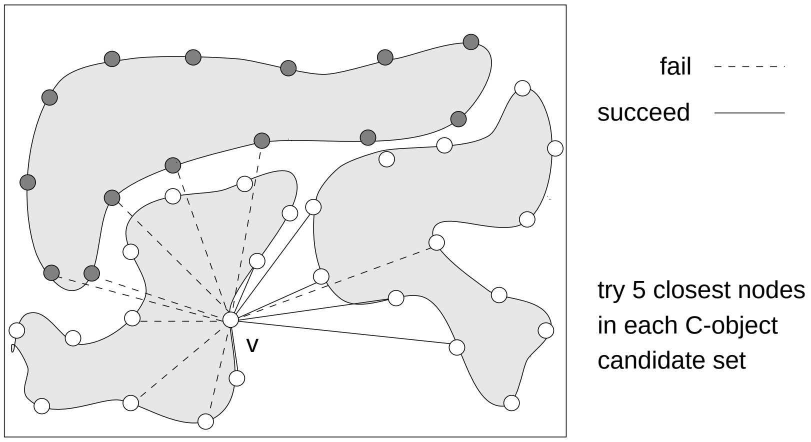


Fig. 6. Connecting of candidate nodes.

simple and deterministic so the paths are easy to reconstruct. In addition, as outlined below, the connection strategy (i.e., determining which connections to attempt) may vary from application to application. In the following, k is a parameter, and distances are measured according to the chosen metric in C-space.

Many different connection strategies could be used in path planning applications. For example, the method used in [22] is to try to connect each node $v \in V$ to its k nearest neighbors in V. A strategy that improves the chance of finding connections across wide C-space corridors is as follows. For each node $v \in V_i$, attempt to connect it to its k nearest neighbors in $V_j \in V$, for $1 \leq j \leq n$ (See Figure 6). Note that both of these strategies require a non-trivial amount of computation to determine the k closest nodes. Thus, it might be desirable to use some heuristic method to identify the 'close' nodes.

The connection strategies for manipulation planning would be more constrained since planning is restricted to the constraint surfaces. In this case a simple approach is to attempt to connect each node $v \in V_i$ to its k_1 nearest neighbors in V_i and also to its k_2 nearest neighbors in $V - V_i$; usually one would select $k_1 > k_2$.

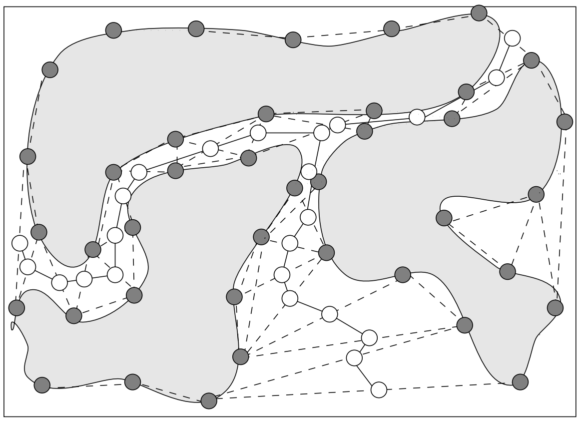


Fig. 7. Improving the roadmap quality for path planning by selecting new roadmap nodes (white) in C-space corridors.

b. Possible optimizations

Since the roadmap nodes are located on constraint surfaces (i.e., points of contact between the robot and an obstacle), the paths encoded in the roadmap will 'skip' along the surfaces of C-obstacles and will not be very desirable for path planning applications. In the planning phase, smoothing techniques can be used to improve the paths. We can also try to improve the quality of the paths in the roadmap by generating additional roadmap nodes near the center of C-space corridors. This could be done, for example, by adding an additional node near the midpoint of the path found by the simple planner when connecting two roadmap nodes (lying on different C-obstacles). Next, connections would be attempted between pairs of these new corridor nodes (see Figure 7).

Another potential optimization is to adaptively determine the number of roadmap candidate nodes to generate and the number of connections to attempt. For example, one could begin by attempting a small number of connections for each node. Then, if the resulting roadmap does not form one connected component, additional connections could be attempted. This process could be iterated until the roadmap

```
Algorithm  NodeConnection (RobotSystem, NodeList, Roadmap, l, k)
Input:  RobotSystem -- robot and obstacle definition
        NodeList -- list of candidate nodes generated

l -- number of connection trails for each node
k -- out degree of graph
Output: Roadmap -- a graph representation of roadmap

Begin
  for each obstacle O(i) in RobotSystem do
    for each obstacle O(j) in RobotSystem do
      for each node x in NodeList(i) do
        find l candidate nodes from NodeList(j)
        for each node y of l candidates do
          if x can be connected to y using a local planner then
            add (x,y) into the roadmap
          if out degree of x equal to k then break
        end /* for */
      end /* for */
    end /* for */
  end /* for */
End.
```

Fig. 8. Roadmap connection algorithm.

is deemed to be sufficiently connected or until no further improvement is obtained. This concept is similar to the learning approach proposed by Overmars [34].

c. Connection algorithm and complexity

Figure 8 shows the pseudo-code of the roadmap connection algorithm. Assume the complexity of the local planner is $F(n_{dof}, n_{obs}, n_{step})$, where n_{dof} is the dof of the robot, n_{obs} is the number of obstacles in the workspace, and n_{step} is the resolution of the local planner. The complexity of the node connection algorithm is:

$$T_{connect-node} = O\Big(n^2 * l * F(n_{dof}, n_{obs}, n_{step})\Big)$$

where n is the total number of nodes in roadmap and l is number of connection trials for each node.

Because the connection phase is the most computationally intensive part of the preprocessing stage, we have devised several techniques to reduce the processing time. Among them, the selection of the value of the parameters and the use of a partitioning heuristic are the most important and are described in detail in Chapter IV.

3. Roadmap connectivity analysis

The roadmap connectivity analysis is the last phase of the preprocessing. The input to this stage is a directed graph generated during the network connection stage.

The method we use is to first transform the directed graph into an undirected graph by duplicating all edges, then we apply a connected component algorithm to the undirected graph. The first step is necessary because the local planner we used in the network connection stage implies an undirected edge in graph. The connected component algorithm we used is a breadth-first search algorithm. The complexity of this phase is:

$$T_{create-roadmap} = O(nk)$$

where n is the total number of nodes in roadmap, and k is the out degree of each node.

After the connected components are computed, the connected component information is added to the roadmap and path planning queries may be performed. Notice that if we have a poorly connected roadmap (there are several big connected subsets and they are isolated from each other), then we may want to either generate more candidate nodes and try the connection phase again, or use a learning scheme to 'learn' critical paths that improve the connectivity of roadmap. However, note that

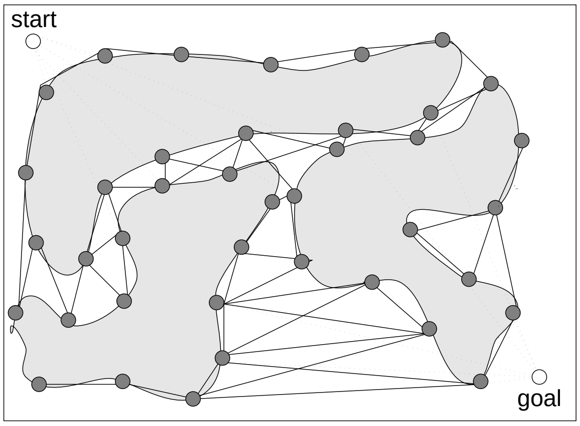


Fig. 9. Query planning.

in some cases it may not be possible to connect all roadmap nodes into one connected component.

C. Planning Queries

After the roadmap is built during the preprocessing stage, planning is carried out as in any roadmap method: we attempt to connect the start and goal configurations to the roadmap, and then find a path in the roadmap between the two connection points. Figure 9 shows the planning scheme. The following approach, proposed in [22], is well suited to our roadmap.

First, the simple planner is used to try to connect the start and goal nodes to the roadmap; connections are attempted between a configuration x (the start or goal configuration) and the k closest roadmap nodes (in the same connected component). If no connection is made for x, then we execute a random walk and try to connect the end node of the random walk to the roadmap. This can be repeated a few times if necessary. If we still cannot connect both nodes to the same connected component of the roadmap, then we declare failure. After both connections are made, we find a path in the roadmap between the two connection points using breadth-first search. Recall that we must regenerate the motion plans between adjacent roadmap nodes since they are not stored with the roadmap. Finally, smoothing techniques can be applied to improve the resulting path.

In many situations, some paths may be preferred to other paths. This information could be encoded in the roadmap by giving edges on desired (undesired) paths lower (higher) weights, and then using a shortest path computation to find the path between the start and goal connection points. Other methods for favoring certain paths have been proposed that could potentially be integrated into the roadmap (e.g., [9]).

The pseudo-code for query processing and path generation is shown in Figure 10. In our experiments the roadmap generated during preprocessing usually contained only one main connected component. For each connected component, we tried MAX_TRY candidate nodes based on a Euclidean distance metric.

Assume the query planner has complexity $C(n_{dof}, n_{obs}, n_{step})$, where n_{dof} is the dof of the robot, n_{obs} is the number of obstacles in the workspace and n_{step} is the resolution of the local planner. The total time of the query processing algorithm is:

$$\begin{aligned} T_{query} &= O\Big(\text{MAX\_TRY} * C(n_{dof}, n_{obs}, n_{step})\Big) \\ &= O\Big(C(n_{dof}, n_{obs}, n_{step})\Big). \end{aligned}$$

The breadth-first search is the same one used in the connected component algorithm and thus the path generation algorithm has complexity:

$$T_{gen-path} = O(nk)$$

where n is the total number of nodes in the roadmap, and k is the out degree of each node.

```
Algorithm Query (RoadMap, sc, gc)
Input:  RoadMap -- roadmap
        sc, gc -- start and goal configurations
Output: boolean return value -- 1 successful, 0 failed

Begin
  for each connected component C of RoadMap do
    try to connect sc and gc to C using a local planner
    if both successful return 1
    else continue
  end /* for */
End.

Algorithm  PathGenerating (C, sc, gc, path)
Input:  C -- a connected component of roadmap
        sc, gc -- start and goal configurations
Output: path  -- a smoothed path between sc and gc

Begin
  let sc' and gc' be the connection nodes of sc and gc
  find a shortest path in C from sc' to gc' using breadth-first search
  connect and smooth the path from sc to gc
End.
```

Fig. 10. Query processing and path generation algorithm.

CHAPTER IV

IMPLEMENTATION ISSUES

We implemented a path planner in a two-dimensional workspace with polygonal obstacles for a fixed-base articulated manipulator, a free-flying segmented robot, and a fixed-base articulated manipulator with fingers. In this chapter we discuss some general issues common to each case. Issues related to individual cases will be mentioned in the next chapter. The descriptions below are for a planar robot with J dof.

A. C-Space Distance Metric

In our planner, a simple Euclidean distance metric in C-space is used (also adopted in [22]). Let $j_i(x)$ denote the position of the ith joint when the robot is in configuration x, $1 \leq i \leq J$. The distance $d(x, y)$ between configurations x and y is

$$d(x,y) = \left(\sum_{i=1}^{J} (d_i(x,y))^2 \right)^{1/2}$$

where $d_i(x, y)$ is the distance between $j_i(x)$ and $j_i(y)$ defined below.

As mentioned before, C-space is not strictly a Euclidean space since there exists a 'wrap-back' problem at the boundary of $[0, 2\pi)$. Some modifications to the distance computation were used to address this problem. In our case, we define $d_i(x, y)$ as:

$$d_i(x,y) = \begin{cases} |j_i(x) - j_i(y)| & \text{if } 0 \leq |j_i(x) - j_i(y)| < \pi \\ 2\pi - |j_i(x) - j_i(y)| & \text{if } \pi \leq |j_i(x) - j_i(y)| < 2\pi \end{cases}.$$

This metric is used in both the roadmap connection and the planning phases to select 'nearby' candidate nodes. Although the metric is simple and intuitive, in some cases a more sophisticated metric may be desirable. For example, one might want to assign greater (less) weight to joints controlling longer (shorter) links of the robot. Another possibility is to assign larger weights to the joints closer to the base of the robot which normally are considered to be more important than the end links.

B. Number of Candidate Nodes

Recall that we attempt to generate a set V_i of m_i roadmap candidate nodes for each obstacle s_i in the workspace. Intuitively, the more nodes in the roadmap, the better it will represent the connectivity of the robot's free-space. However, even though the generation of candidate nodes is very fast in our implementation, we do not want to generate too many since this increases the time spent connecting the roadmap, which is by far the most costly part of the preprocessing. As previously mentioned, the number of candidate nodes that should be generated depends on both the amount of preprocessing one is willing to do and on the geometry of the C-obstacle.

On the other hand, since the node generation is randomized, the actual number of candidate nodes n_i generated for each obstacle s_i is a random function:

$$n_i = F(m, s_i)$$

where m is the number of ray directions for each C-obstacle. For simplicity, we used the same m for each C-obstacle.

After node generation, n_i will reflect the difficulty of generating candidate nodes for each C-obstacle, e.g., the bigger (smaller) the number n_i, the easier (harder) to generate nodes on s_i. This information could be utilized to enhance the quality of the roadmap, e.g., to identify which C-obstacles need more nodes.

C. Collision Detection

The dominant operation in the creation of the roadmap is collision detection: it is heavily used both in the node generation and in the roadmap connection phases. Thus, its efficiency is of vital importance to the overall efficiency of the method. In our implementation, collision detection was implemented by checking each link of the robot with each obstacle using a line segment intersection algorithm [12]. For our two-dimensional workspaces, collision detection is carried out by checking each link of the robot with each edge of each polygonal obstacle. For fixed-base robots, this is sufficient since the base of the robot is outside of any obstacle by default. For free-flying robots, we also need to check that the base is outside all obstacles. This can be done by using the line segment intersection algorithm without increasing the total complexity of algorithm. Generally, the complexity of this collision detection algorithm is

$$T_{collision-detection} = O(n_{dof} * n_{obs} * n_{vert})$$

where n_{dof} is the number of links of the robot, n_{obs} is the number of obstacles, and n_{vert} is the total number of vertices of the obstacles.

Better results might have been obtained if we had used C-space bitmaps for each link of the robot as was done in [22]. However, as mentioned before, the bitmap method has draw-backs like memory explosion and coarse resolution, especially in three-dimensional workspaces.

In our method, the collision detection algorithm could be any algorithm which meets the computation time, memory, and resolution requirements. We could even switch between different types of algorithms for different obstacles.

D. Interconnection Strategy

In the interconnection phase, we would like to connect all the roadmap candidate nodes into a single connected component, if possible. Clearly, the interconnection strategy chosen can greatly affect how close we come to this goal. Recall that we have a set V_i of roadmap candidate nodes for each obstacle s_i in the workspace. The basic approach we took was to attempt to connect each node $v \in V_i$ to its k nearest neighbors in $V_j \in V$, for $1 \leq j \leq n$, where k is a constant. Clearly, the larger k, the greater chance of connection, but also the more computation involved; for our experiments $k = 10$.

Note also that a significant amount of computation is required to determine the k closest nodes for *every* candidate node in *every* candidate set. In an attempt to reduce the cost of finding the closest nodes, we tried a heuristic method to identify 'close' nodes. We partitioned V_j into k equal-sized subsets, and found the closest node in each subset; in this way we are guaranteed to find the closest node, and will also probably get some other close nodes. Besides, since the distance metric is a heuristic itself, the partitioning method may help overcome the limitations of the pure Euclidean distance metric. By using this heuristic method, the complexity of choosing the closest nodes is reduced from $O(n * k\log k)$ (insertion sort) to $O(nk)$, where n is the number of nodes in the candidate set and k is the out degree of each node. The experimental results show that this method works well.

E. Local Planner

The local planner is used in both the roadmap connection phase of the preprocessing and in the path planning queries. Great care should be taken in selecting the local planner and different methods will be needed in different situations. 'Local' here means the inputs of this planner are relatively near each other as compared with the global method.

The local planner used in the roadmap connection phase (called the *connection planner*) must be a deterministic method since the paths connecting nodes in the roadmap will not be stored. Our experiments show that the efficiency of this local planner is of crucial importance since the roadmap connection phase required orders of magnitude more time than any other part of the preprocessing. So it must be simple and fast rather than powerful and slow.

We used a simple straight-line planner as the connection planner. Let x and y be the two configurations we wish to connect. The local planner we tried was to move directly along the straight line segment connecting x and y in C-space, performing collision-detection checks at uniform intervals on the line segment. The distance between subsequent points checked is determined by the resolution needed for the current problem instance.

The requirements for the local planner used in the query phase (called the *query planner*) are different. The goal of this planner is to connect the start and goal configurations to the roadmap and it is possible to store these two paths. After the candidate nodes in the roadmap are chosen, we hope to connect the start and goal configurations to them with the least number of tries. Hence, we can use a powerful and expensive local planner as the query planner.

In our implementation, the query planner is the same as connection planner and is a simple straight-line planner. We found this to be sufficient, since in our test examples, the majority of the query configurations can be connected to the roadmap with a simple straight-line planner. For further enhancement, one could include the potential field and random walk methods in the query planner to improve the success rate.

Like the collision detection method, the local planners used in our implementation are fairly independent. Better local planners could be adopted without much effort and the performance of the method would be improved.

CHAPTER V

EXPERIMENTAL RESULTS AND ANALYSIS

A path planner for planar articulated manipulators in a two-dimensional workspace was implemented and evaluated for different robot configurations and varying environments. The code was written in C. All measurements were taken on an SGI Indy workstation running IRIX Version 5.3; the CPU was a 100 MHz MIPS R4600 with 32 MB of RAM. All I/O requests were serviced remotely using NFS. The planner was implemented as an integrated graphic environment with a Motif interface.

We used the heuristic interconnection strategy mentioned in Chapter IV to pick $k = 10$ candidates from each set V_i for every roadmap node. Even with this simple strategy, in most cases almost all the nodes were contained in one large connected component.

A. Experiment Scenarios

Robots. Three types of robots were used in the experiments. They were:

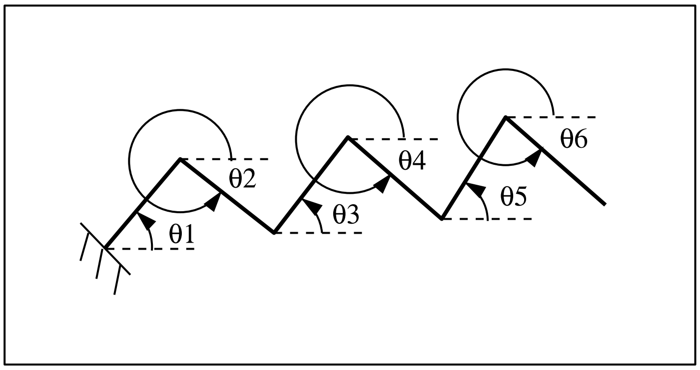


Fig. 11. A 6-link fixed-base planar manipulator.

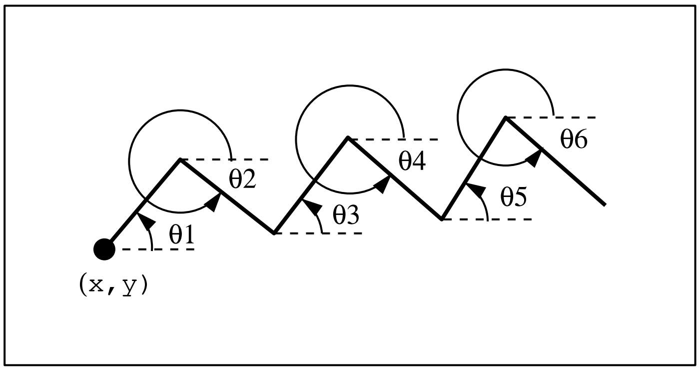


Fig. 12. A 6-link 8-dof free-flying planar robot.

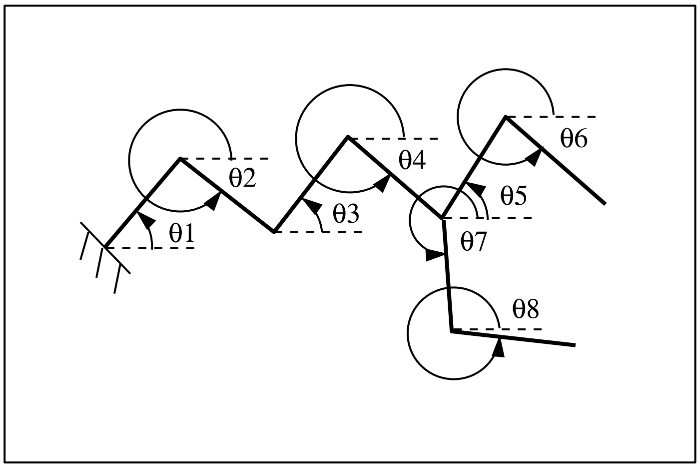


Fig. 13. A 8-dof fixed-base planar manipulator with fingers.

1. *Fixed-base planar robots.* A 6-link fixed-base manipulator is shown in Figure 11. Its base is fixed and a configuration is given by specifying the values θ_1, θ_2, ... θ_6 of the 6 joint angles. In our experiments, each link has the same length. We varied the dof (number of links) of the robot from 6 to 20.

2. *Free-flying planar robot.* A 6-link 8-dof free-flying robot is shown in Figure 12. It has a moving base (x, y) contributing 2 dof and 6 additional dof represented by the 6 joint angles θ_1, θ_2, ... θ_6 of its 6 links.

3. *Fixed-base planar robot with fingers.* As shown in Figure 13, an 8-link fixed-base manipulator has its base fixed and θ_1, θ_2, ... θ_8 represent the joint angles of its 8 links. The robot has two fingers consisting of links 5 and 6 and links 7 and 8.

Workspaces. The obstacles in the 2D workspace are rectangles with different sizes and positions. In the cases of the fixed-base manipulators, the workspace is open. In the case of the free-flying robot, the workspace is bounded by a rectangle which is specified as a special obstacle called the frame obstacle. All candidate nodes are generated on the inside surface of the frame obstacle instead of on the outside surface as with the normal obstacles. The specific environments studied are described with the experiment results.

B. Fixed-Base 6-dof Planar Robot – Varying Environments

This set of experiments shows the performance of the preprocessing stage for a fixed-base 6-dof planar robot in 2D environments which differ in terms of the width of the free corridors in C-space. The environments are shown in Figure 14. The obstacles are four 0.6in.$\times$0.6in. squares and the gap between them varies from 0.6 inches to 0.24 inches. The robot is a fixed-base 6-dof planar robot. The length of each link of the robot is 0.2 inches. When the gap between the obstacles in the workspace becomes narrower, so does the free corridor in C-space. These long, narrow free corridors in C-space are hard to find using the probabilistic roadmap method proposed by Kavraki and Latombe [22] and Overmars and Švestka [34] because:

1. Since the roadmap candidate nodes are generated using uniform random sampling of C-space it is difficult to generate nodes in the free corridors. It is important to find such nodes because these free corridors in C-space are most likely bridges between major connected components of the roadmap.

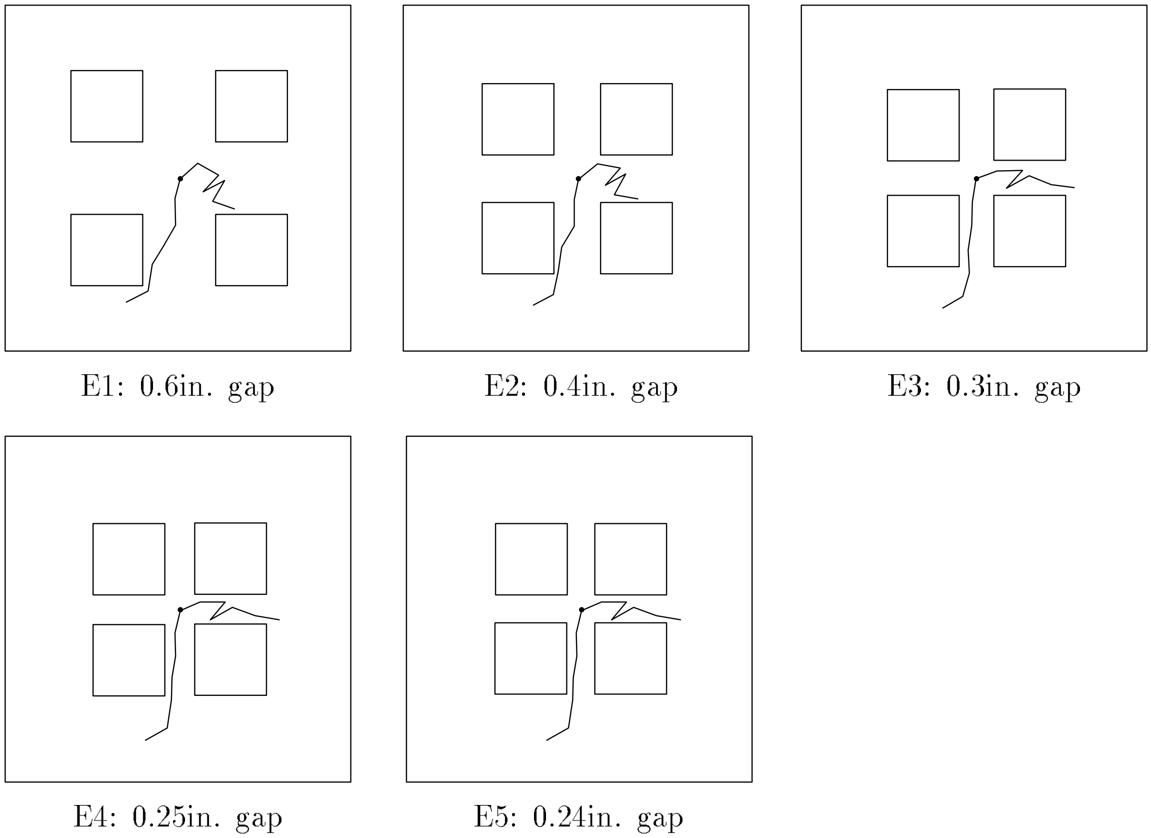
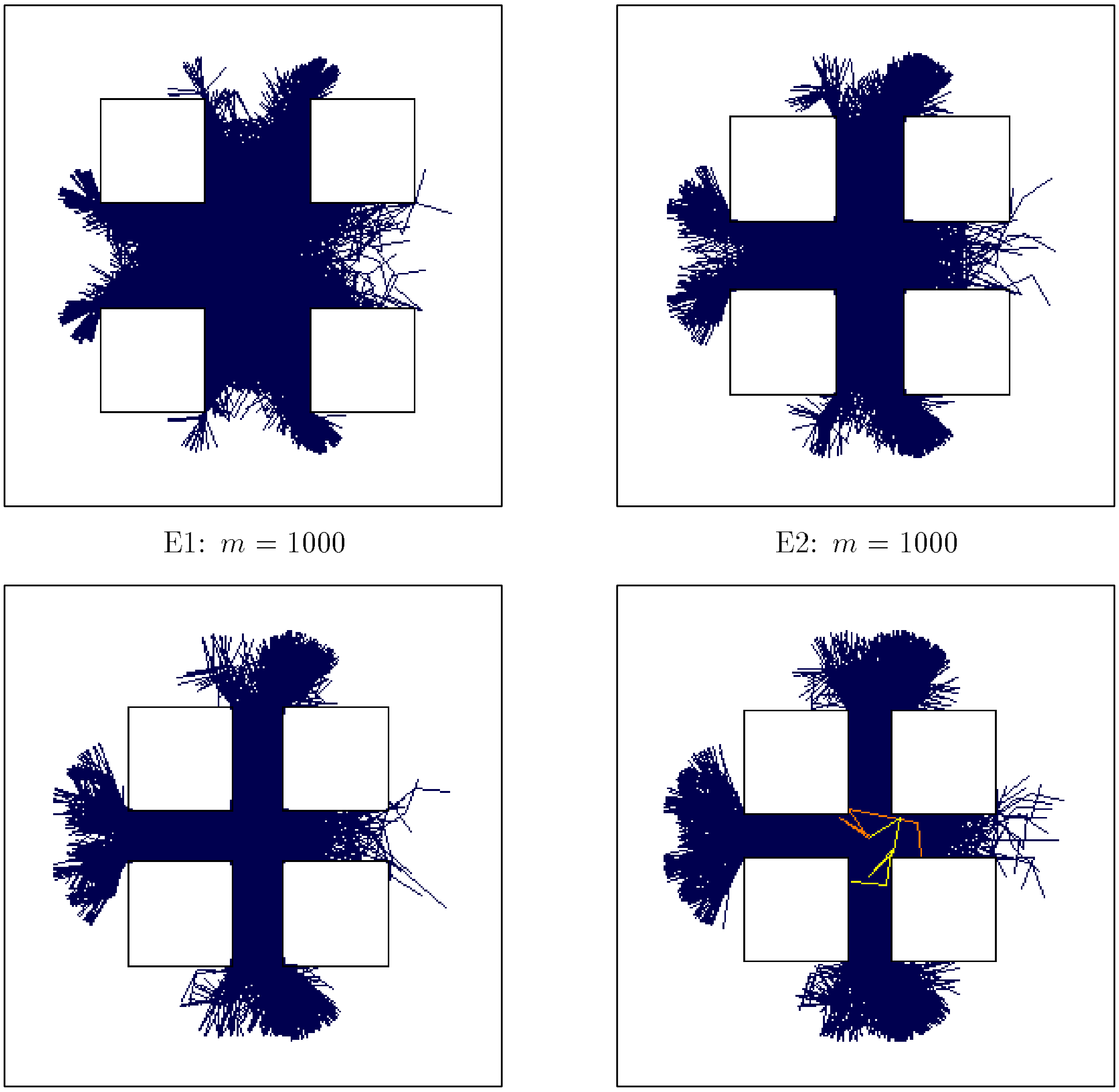


Fig. 14. Environments with four 0.6in. $\times$ 0.6in. obstacles and varying gaps between them.

2. It is difficult to enhance the roadmap after it is generated since it is hard to find an accurate metric to indicate which part of the C-space is 'difficult' or 'crowded'.

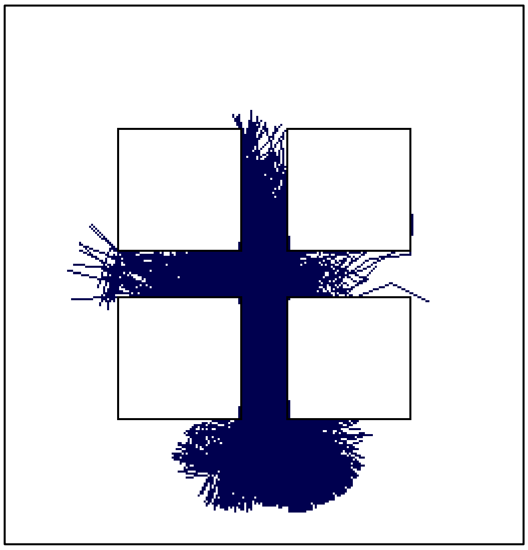
The obstacle-based probabilistic roadmap method handles narrow free corridors in C-space much better. The roadmaps generated for E1, E2, E3 and E4 using the OBPRM method are shown in Figure 15. In the figures, the roadmap nodes are displayed one on top of another in the workspace and different connected components are drawn in different gray scale values. The roadmaps for E1, E2 and E3 are fully connected and the roadmap for E4 (m=2000, which is the number of directions for each obstacle) has three connected components: a big component with 4627 nodes



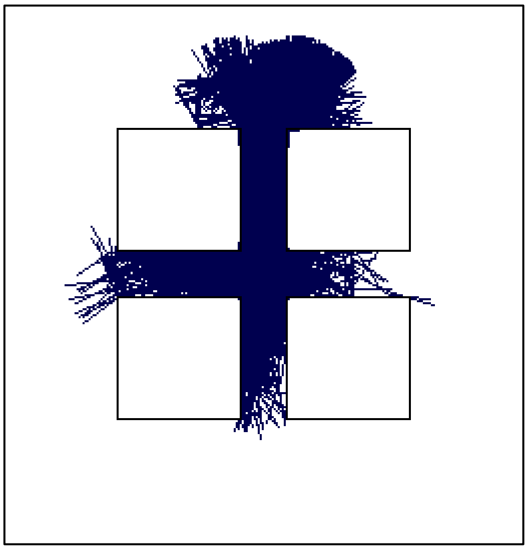
E1: $m = 1000$ E2: $m = 1000$

E3: $m = 1000$ E4: $m = 2000$

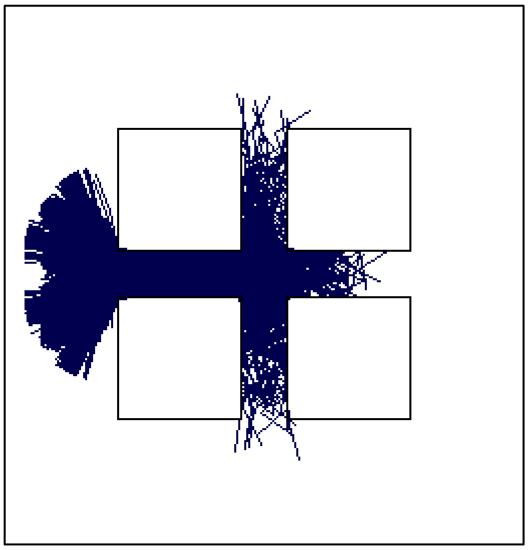
Fig. 15. Roadmaps for E1, E2, E3 and E4.



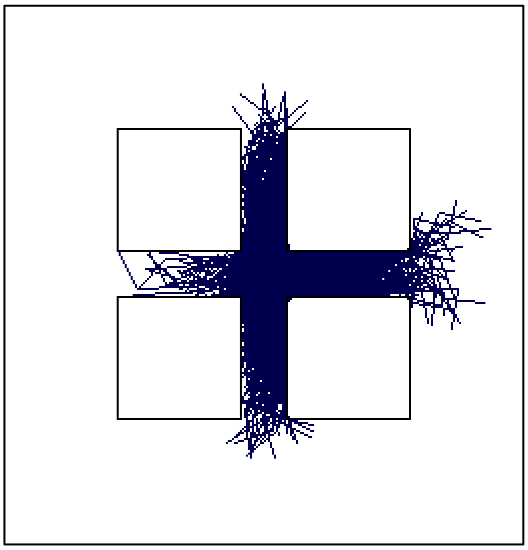
E5: Connected component 1



E5: Connected component 2



E5: Connected component 3



E5: Connected component 4

Fig. 16. Connected components of roadmap for E5, $m = 4000$.

TABLE I
PREPROCESSING TIMES (SECONDS) AND STATISTICS
FOR E1, E2, E3, E4 AND E5

Env.	m	Generation		Conn.	Roadmap Structure		
		#nodes	time	time	#cc	sizes	time
E1	200	799	1	139	1	799	1
E1	500	2000	1	338	1	2000	3
E1	1000	4000	3	688	1	4000	10
E2	200	733	1	136	1	733	1
E2	500	1838	1	334	1	1838	2
E2	1000	3690	3	659	1	3690	7
E3	200	616	1	104	1	616	1
E3	500	1535	1	265	1	1535	1
E3	1000	3038	3	535	1	3038	4
E4	200	458	1	53	35	403, 5(3), 3(2), ...	1
E4	500	1141	1	147	12	1129, 2, 1(10)	1
E4	1000	2349	3	321	13	2332, 6, 1(11)	2
E4	1500	3491	5	498	3	3489, 1(2)	4
E4	2000	4629	7	691	3	4627, 1(2)	7
E5	200	456	1	51	29	169, 150, 71, 31 4(2), ...	1
E5	500	1123	1	136	29	823, 185, 57, 11, 10, ...	1
E5	1000	2183	3	285	15	830, 808, 376, 156, 2(2), 1(9)	2
E5	1500	3200	5	443	17	2361, 578, 246, 2, 1(13)	4
E5	2000	4391	7	606	5	3265, 793, 331, 1(2)	5
E5	3000	6611	10	965	10	2466, 2445, 1181, 513, 1(6)	11
E5	4000	8681	14	1363	5	3290, 3180, 1535, 675, 1	21

and two components each consisting of a single node (the gray ones in E4: m=2000). No fully connected roadmap could be generated for E5. Figure 16 shows the four biggest connected components in the roadmap for E5 (with $m = 4000$). Below we discuss the results for each environment in more detail.

The preprocessing times (seconds) and other statistics related to the roadmap construction are shown in Table I. The first column is the environment, the second column is m, the number of randomly selected directions used to generate candidate nodes for each C-obstacle, the third column lists the number of configurations successfully generated as roadmap candidate nodes, the fourth and fifth columns show the times (seconds) for the node generation and roadmap connection phases, respectively, the sixth column lists the number of connected components in the roadmap after the interconnection process (using only the straight-line in C-space planner), the seventh column lists the number of nodes in each connected component (if needed, the number of connected components of a certain size is indicated in parenthesis and the smaller connected components are omitted if there are too many of them), and the eighth column shows the time (seconds) for roadmap structure creation.

From the table, we can see that the obstacle-based probabilistic roadmap method yields good roadmaps for E1, E2 and E3. Even when m is 200, the roadmaps generated for E1, E2 and E3 are fully connected. For environment E4 (4 obstacles with a 0.25 inch gap), however, the roadmaps generated are not fully connected when m=200, 300, 500 or 1000. The roadmaps become almost fully connected when m=1500, 2000. This shows the roadmap becomes better connected when m is increased. Hence m can be used to adaptively guide the preprocessing. We can assign an initial value to m and keep increasing m until a well connected roadmap is obtained, or the limit of m is reached.

For environment E5 (4 obstacles with a 0.24 inch gap), the roadmaps generated consist of several big connected components but they are isolated from each other. The roadmaps remain unconnected even when m is increased to 4000. Experiments with m = 5000 and 6000 have been done on machines with bigger memory (more than the Indy's 32MB) and the results are similar.

Thus we can say that the 0.25 inch gap is near the critical point for this robot (with link lengths of 0.20 inches). If the obstacles are nearer, it is hard to generate a fully connected roadmap even with very big m. One way to overcome this problem is human assistance, e.g., the operator could input a critical path to bridge the isolated components of the roadmap if he knows that one exists. Another way would be to use more powerful local planners to connect those components.

Notice that when the obstacles are nearer (free C-space becomes narrower), the actual number of candidate nodes in the roadmap becomes smaller (for the same m) and the running time of connection phase becomes smaller and so does the total running time. Thus, our method has faster running times when C-space is crowded.

To show the effect of the parameter m on the running time and the size of roadmap, Figures 17 and 18 show the time and roadmap size as functions of m for E4 and E5.

The graphs show that the node generation time is trivial compared with the total preprocessing time. This means that a more complex node generation method could be used to balance the time distribution and hence improve the performance. The graphs also show that when the environment becomes more 'crowded' (from E4 to E5) the preprocessing time and the size of the roadmap decrease. This is understandable since fewer candidate nodes are generated in more 'crowded' environments.

In the graphs both the running time and the size of roadmap are almost linear functions of m. It seems m is the parameter we can use to control the size of the roadmap and the preprocessing time. The actual relation between the size of the roadmap and m remains to be investigated, since it differs from environment to environment and is related to the 'crowdedness' of C-space.

C. Free-Flying 8-dof Planar Robot – Varying Environments

This set of experiments shows the performance of the preprocessing stage for a free-flying 8-dof planar robot in 2D environments which differ in terms of the width of the free corridors in C-space. The environments are shown in Figure 19. The obstacles are four rectangles within a 3.0in. $\times$ 1.6in. rectangle frame. The size of the obstacles

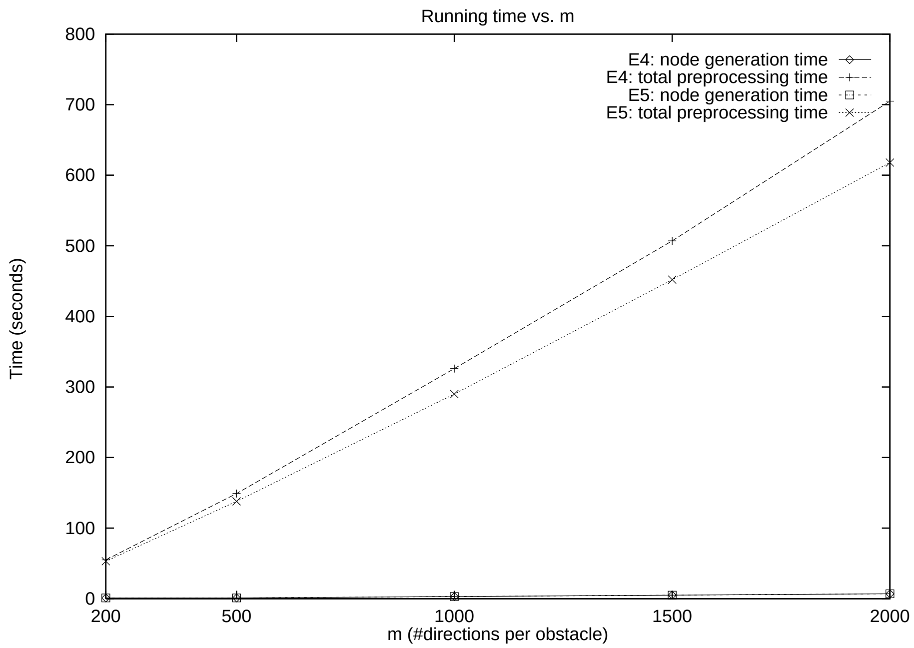


Fig. 17. Running time vs. m.

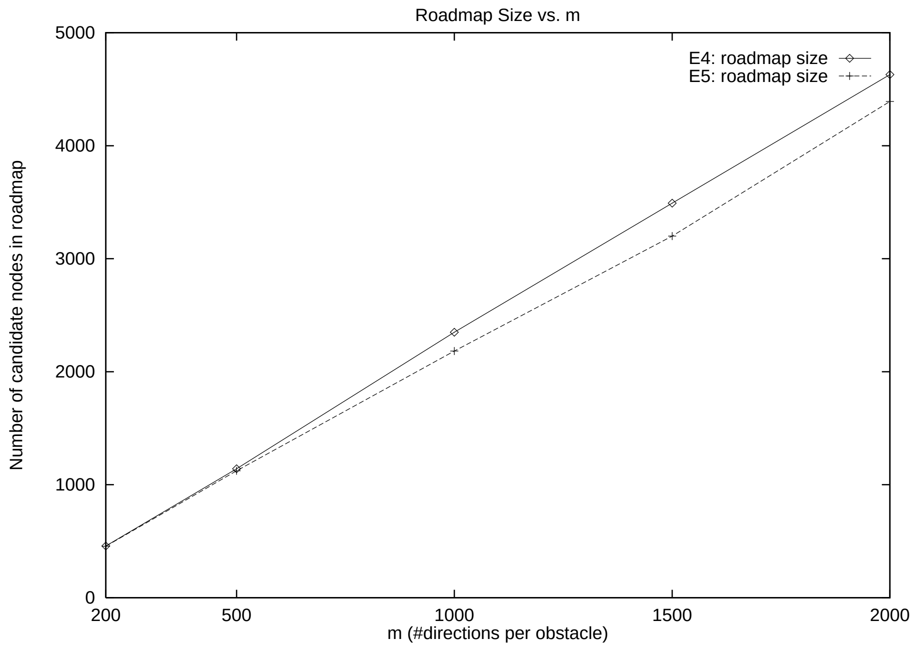


Fig. 18. Roadmap size vs. m.

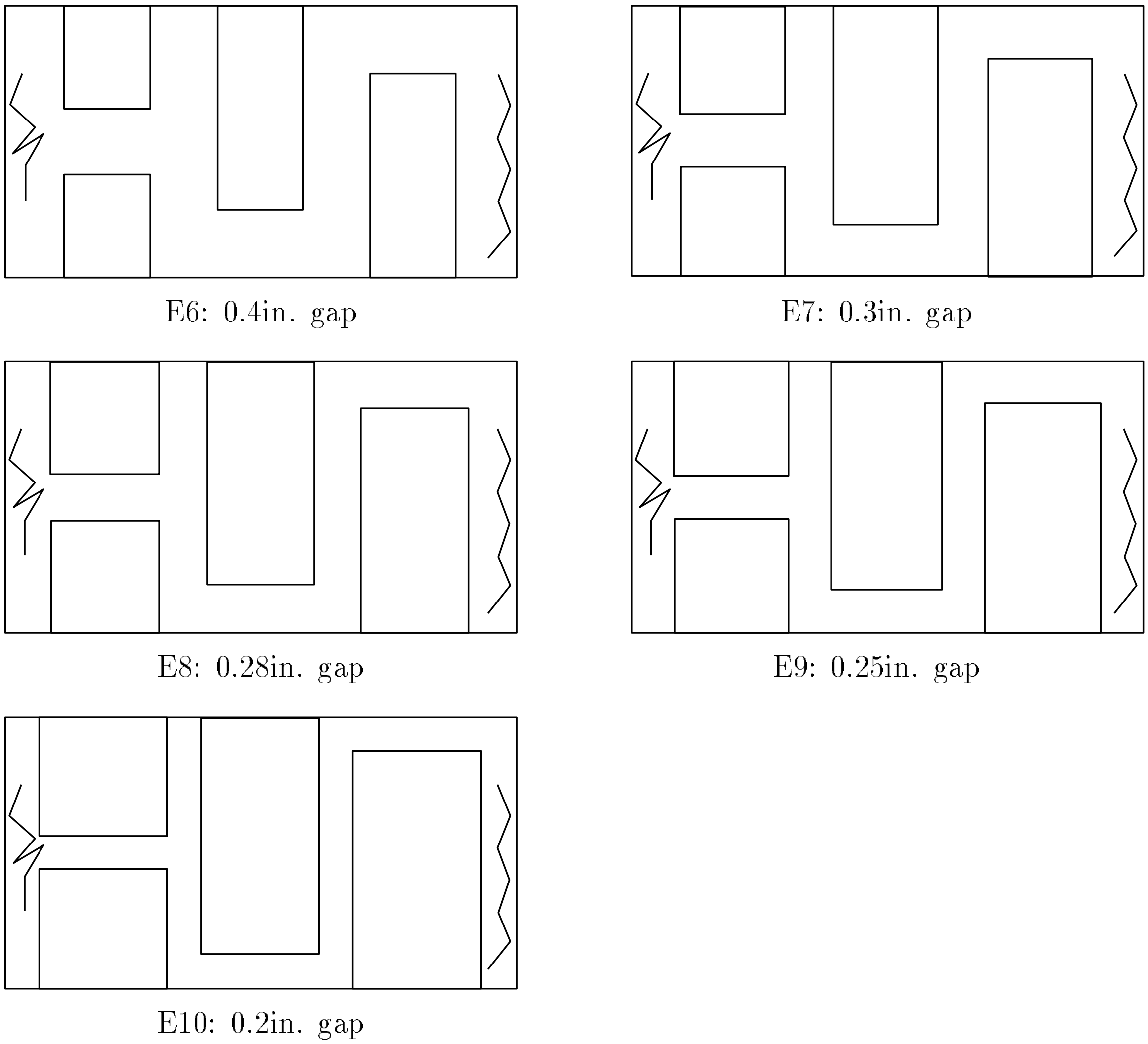


Fig. 19. Environments with five obstacles and varying gaps between them.

TABLE II
Preprocessing Times (Seconds) and Statistics
for E6, E7, E8, E9 and E10

Env.	m	Generation		Conn.	Roadmap Structure		
		#nodes	time	time	#cc	sizes	time
E6	500	567	2	130	8	553, 8, 1(6)	1
E6	1000	1000	4	260	3	998, 1(2)	1
E6	1500	1749	6	450	3	1748, 1	2
E6	2000	2271	8	556	9	2263, 1(8)	3
E7	1000	518	4	86	32	166, 140, 65, 51, 48, ...	1
E7	1500	2131	6	399	11	1785, 336, 2, 1(9)	3
E7	2000	1476	8	289	28	1234, 213, 4, 1(27)	2
E7	2500	1202	11	235	19	605, 441, 140, 1(16)	1
E7	3000	2042	13	446	13	2030, 1(12)	2
E7	3500	2397	15	452	16	2378, 4, 2, 1(13)	2
E7	4000	3253	17	650	14	3240, 1(13)	11
E8	2000	908	8	226	12	897, 1(11)	1
E8	3000	798	13	167	25	367, 185, 149, 68, ...	1
E8	4000	1338	17	236	13	1323, 3, 2, 1(10)	2
E8	5000	2838	22	617	17	2821, 2, 1(15)	5
E8	6000	2089	26	387	24	1869, 194, 3(2), 1(20)	3
E8	7000	3560	30	671	28	3530, 2(3), 1(23)	15
E9	2000	276	8	33	26	90, 47, 45, 31, 28,...	1
E9	2500	473	10	78	34	200, 149, 55, 18, 8(2),...	1
E9	3000	510	13	86	36	169, 167, 62, 52, 25,...	1
E9	3500	2021	15	410	22	1829, 169, 3, 2, 1(18),	2
E9	4000	1323	17	277	16	845, 409, 53, 3, 2, (11)	1
E9	4500	1089	19	209	36	472, 463, 93, 9,...	2
E9	5000	776	21	132	37	409, 258, 38, 22, 7, ...	1
E9	5500	1124	24	180	42	308, 264, 215, 158, ...	1
E9	6000	1367	26	254	29	638, 618, 60, 22, 3, ...	1
E9	7000	1471	30	273	43	1361, 53, 9, 6, 2(3), 1(36)	1
E9	8000	923	34	158	37	644, 97, 70, 40, 29, ...	1
E9	9000	1889	38	330	58	630, 457, 419, 275, 49,...	2
E10	7000	3188	30	487	35	1555, 1106, 249, 111, ...	20
E10	8000	4301	35	831	58	2285, 2036, 268, ...	64
E10	9000	3600	39	629	45	1765, 1383, 286, ...	21

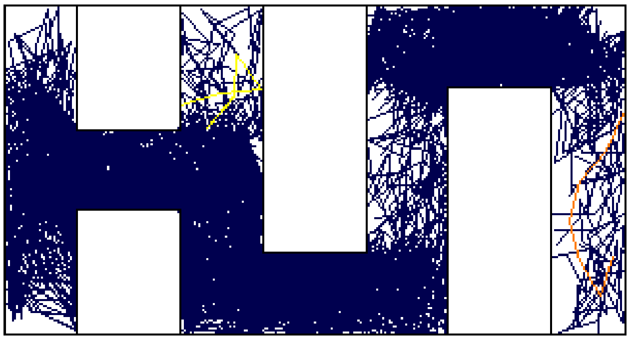


Fig. 20. Roadmap for E6 with $m = 1000$.

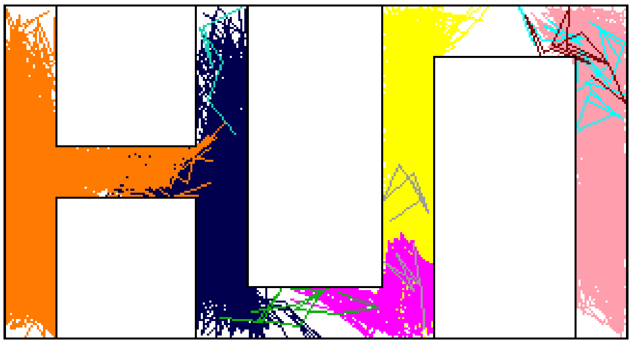


Fig. 21. Roadmap for E9 with $m = 9000$.

are changed so that the gap between them varies from 0.4 inches to 0.2 inches. The robot is a free-flying 6-link planar robot which has 8 dof. The length of each link of the robot is 0.2 inches. When the obstacles are near enough (E10), the rotation of the links of the robot is limited. The only possible move would be sliding in E10. In C-space, this is regarded as a long, narrow corridor.

The preprocessing times (seconds) and other statistics are shown in Table II, where m is the number of randomly selected directions used to generate candidate nodes for each obstacle. In this case, as previously mentioned, the frame box is processed as a special obstacle. As with the fixed-base robot, the roadmaps generated for the free-flying robot in relatively uncrowded environments are well connected even with small m.

For E6, the gaps between obstacles are 0.4 inches and that is far enough to generate well connected roadmaps for m = 500, 1000, 1500 and 2000. The roadmap generated for E6 when m = 1000 is shown in Figure 20. The roadmap is almost fully connected. There are three connected components in the roadmap: one component has 998 nodes and the other two components each consist of a single node.

For E7, the gaps are reduced to 0.3 inches and it takes at least m = 3000 to obtain a well connected roadmap. Since the algorithm is a randomized one, a non-stable case like m = 2500 can occur, in which fewer nodes are generated than were generated with smaller values of m.

For E8, the gaps are 0.28 inches wide and the smallest value of m to obtain a well connected roadmap is 4000. Similarly, for E9, the gaps are 0.25 inches wide and it becomes difficult to generate a well connected roadmap even with large m. The roadmap generated for E9 (with m= 9000) are shown in Figure 21. It has 58 connected components, which is not a well connected roadmap. The five biggest components of size 630, 457, 419, 275 and 49 are shown in the graph with different gray scale values.

For E10, the rotation of the links is limited since the gap size equals the link length of 0.2 inches. The preprocessing cannot generate a well connected roadmap even with m = 7000, 8000, or 9000. The same methods as for the fixed-base robot could help to improve the performance in this situation.

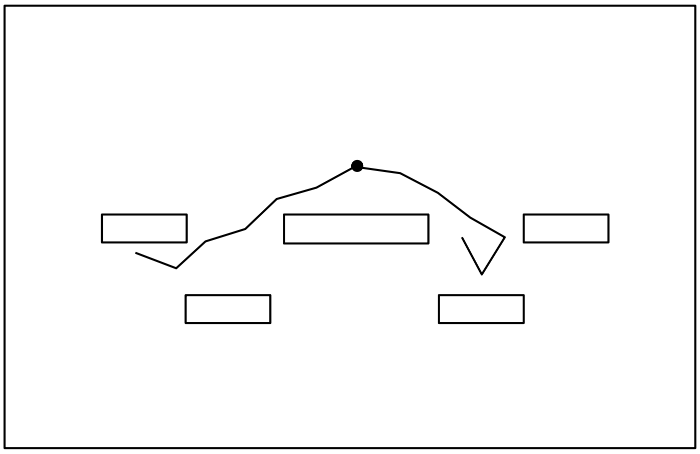


Fig. 22. E11: fixed-base 6-dof robot and five rectangular obstacles.

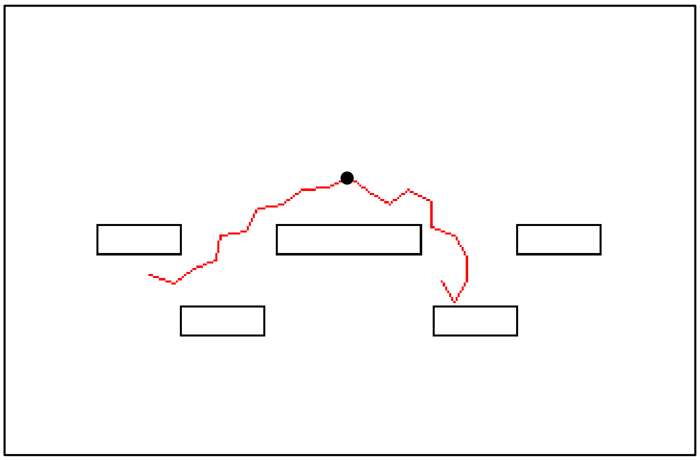


Fig. 23. E12: fixed-base 10-dof robot and five rectangular obstacles.

D. Fixed-Base Planar Robot – Varying DOF

This set of experiments shows the performance of the preprocessing stage for fixed-base planar robots when the number of links is varied from 6 to 20. The environments are shown in Figure 22 (6 dof) and Figure 23 (10 dof). The obstacles are five rectangles (one 0.7in. $\times$ 0.14in. rectangle and four 0.4in. $\times$ 0.14in. rectangles). The robots are fixed-base planar robots with 6, 8, 10, 12, 16 and 20 links of length 0.2, 0.15, 0.12, 0.10, 0.075 and 0.06 inches, respectively. Hence, the total length of the links is the

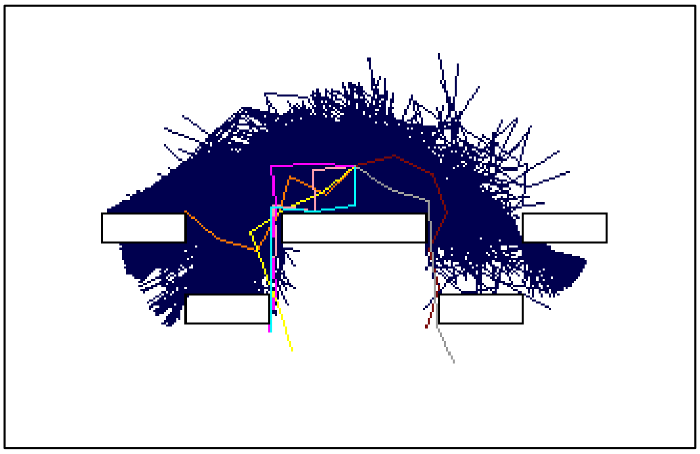


Fig. 24. Roadmap for E11 and 6-dof robot with $m = 1000$.

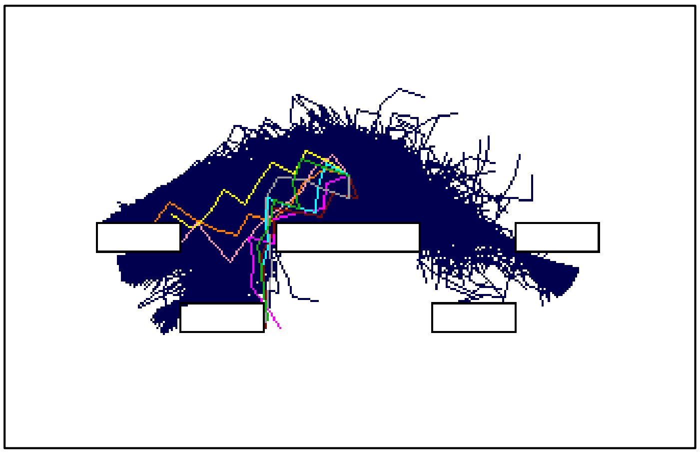


Fig. 25. Roadmap for E12 and 10-dof robot with $m = 1500$.

same for each robot (1.2 inches) and the workspaces are also the same.

The roadmap generated for the 6-dof robot when $m = 1000$ is shown in Figure 24. The roadmap is well connected and has eight connected components: a big component with 2848 nodes (shown in black in the graph) and seven components each of which has only a single node (shown in gray in the graph). Note that some of the single node components lie in the 'difficult' areas and more nodes could potentially be generated in these areas with a larger m. The roadmap generated for the 10-dof robot when

TABLE III
PREPROCESSING TIMES (SECONDS) AND STATISTICS FOR 5 OBSTACLE ENVIRONMENT AND ROBOTS WITH DOF 6, 8, 10, 12, 16 AND 20

DOF	m	Generation		Conn.	Roadmap Structure		
		#nodes	time	time	#cc	sizes	time
6	300	870	1	202	2	869, 1	1
6	500	1445	2	344	1	1445	3
6	1000	2885	4	680	8	2848, 1(7)	6
8	300	864	1	252	2	863, 1	1
8	500	1458	2	430	2	1457, 1	2
8	1000	2938	5	856	3	2936, 1(2)	7
8	1500	4392	8	1287	8	4383, 2(2), 1(5)	15
10	300	894	2	328	1	894	1
10	500	1464	3	560	5	1460, 1(4)	2
10	1000	2945	7	1132	7	2939, 1(6)	8
10	1500	4387	11	1617	9	4379, 1(8)	14
12	300	906	2	398	3	904, 1(2)	1
12	500	1478	4	628	3	1476, 1(2)	2
12	1000	2985	8	1361	6	2980, 1(5)	6
12	1500	4428	13	1979	12	4416, 2, 1(10)	12
16	300	892	3	507	3	890, 1(2)	1
16	500	1515	5	820	8	1508, 1(7)	1
16	1000	3001	11	1810	15	2987, 1(14)	5
16	1500	4537	17	2692	33	4505, 1(32)	11
20	300	904	4	609	19	886,1(18)	1
20	500	1530	7	1000	6	1525, 1(5)	1
20	1000	3013	14	2127	38	2976, 1(37)	3
20	1500	4600	21	3257	85	4514, 2(2), 1(82)	8

$m = 1500$ is shown in Figure 25. The roadmap is also well connected and has nine connected components: a big component with 4379 nodes (shown in black in the graph) and seven components each of which has only a single node (shown in gray in the graph).

The preprocessing times (seconds) and other statistics are shown in Table III, where m is the number of randomized directions used to generate candidate nodes for each obstacle.

The results show that our method yields well-connected roadmaps when the dof

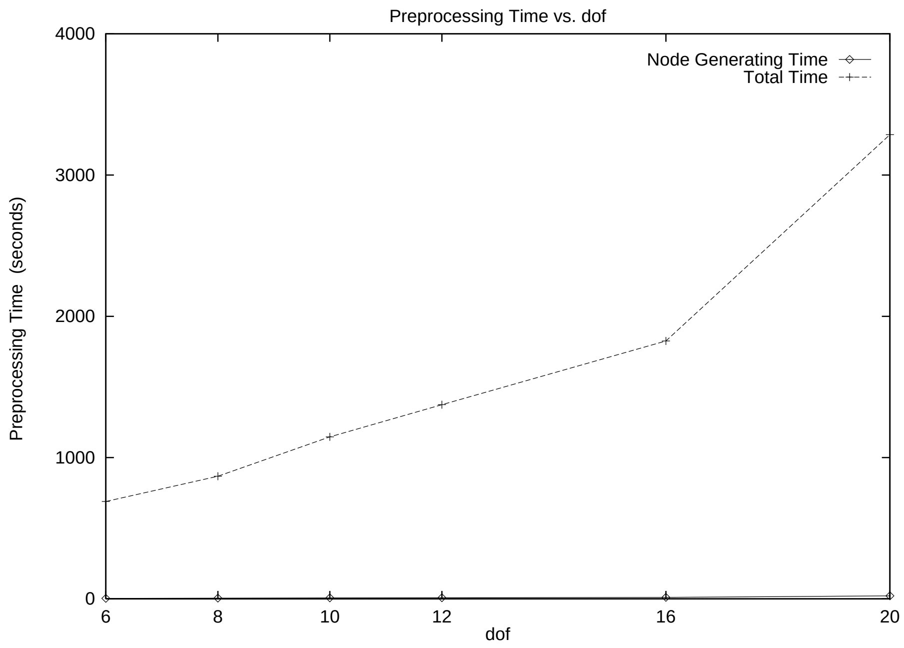


Fig. 26. Preprocessing time vs. dof with $m = 1000$.

of the robot is varied from 6 to 20.

The preprocessing time vs. dof (with $m = 1000$) is shown in Figure 26. Our results are in accordance with our theoretical analysis which indicates that the complexity of preprocessing is a polynomial function of the dof. Again, the graph shows the dominating part of the preprocessing is the roadmap connection whose main components are the collision detection and connection planning. A simple and fast local planner could improve the performance dramatically. However, when the environment needs a more subtle and powerful planner, the complexity of preprocessing will increase.

We note that the collision detection and the connection planner are external methods to our approach. Users should choose algorithms that achieve the best performance for their problems.

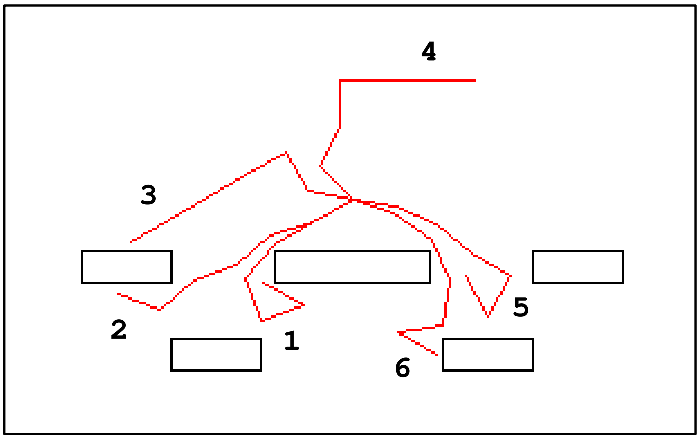


Fig. 27. Queries of fixed-base robot.

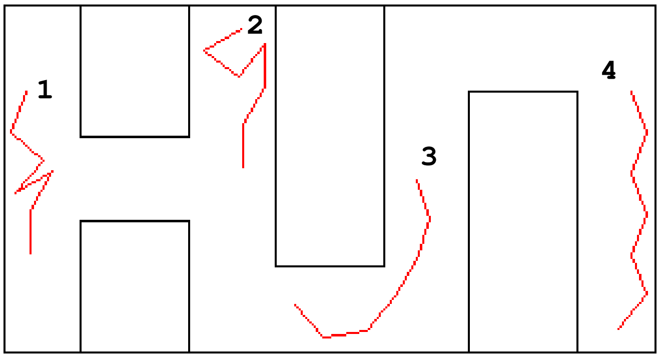


Fig. 28. Queries of free-flying robot.

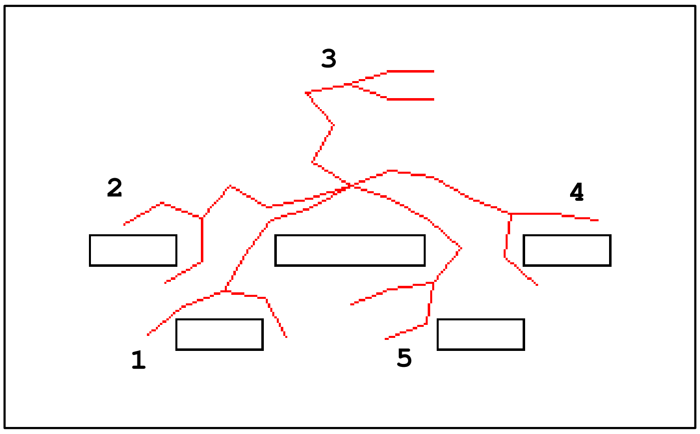


Fig. 29. Queries of fixed-base robot with fingers.

TABLE IV
TIMES (SECONDS) TO CONNECT CONFIGURATIONS TO THE LARGEST CONNECTED COMPONENT USING THE STRAIGHT-LINE PLANNER

Env.	Robot Type & m	Configuration Number					
		1	2	3	4	5	6
E11	Fixed-base, m=1000	.013	.011	.015	.012	.011	.022
E6	Free-flying, m=1000	.014	.012	.020	.055	–	–
E11	Fixed-base with fingers, m=1000	.016	.014	.015	.634	.149	–

E. Planning Queries

After a well connected roadmap is constructed for an environment during preprocessing, the actual path planning queries can be carried out very quickly. In the following we examine some example queries for the three types of robots in three environments. Figure 27 shows six configurations of a 6-dof fixed-base robot in E11, Figure 28 shows four configurations of an 8-dof free-flying robot in E6, and Figure 29 shows five configurations of an 8-dof fixed-base robot with fingers in E11.

The times (seconds) required to connect the example configurations to the corresponding roadmap for each environment are shown in Table IV. The times include both the time of the simple planner and the time to search the roadmap for candidates for connection (again, using the heuristic partitioning method). The only planner used in this stage was the straight-line in C-space planner, i.e., if the straight-line planner failed we did *not* try the enhanced approach of executing a random walk and then attempting to connect the terminus of the random walk to the roadmap. Even so, for the examples shown above, all configurations are connected to the roadmaps and the query times are all less than one second.

The time for path planning between any two configurations would add the time to find the path in the roadmap between the connection points to the connection times reported above. For the environments considered here, the search in the roadmap for these paths is very fast (e.g., less than a second). For example, a path was generated in 0.079 seconds to connect configuration 2 to configuration 6 in Figure 27. Serial snapshots of the path are shown in Figure 30. In another example, a path was generated in 0.706 seconds to connect configuration 1 to configuration 4 in Figure 29. Serial snapshots of the path are shown in Figure 31.

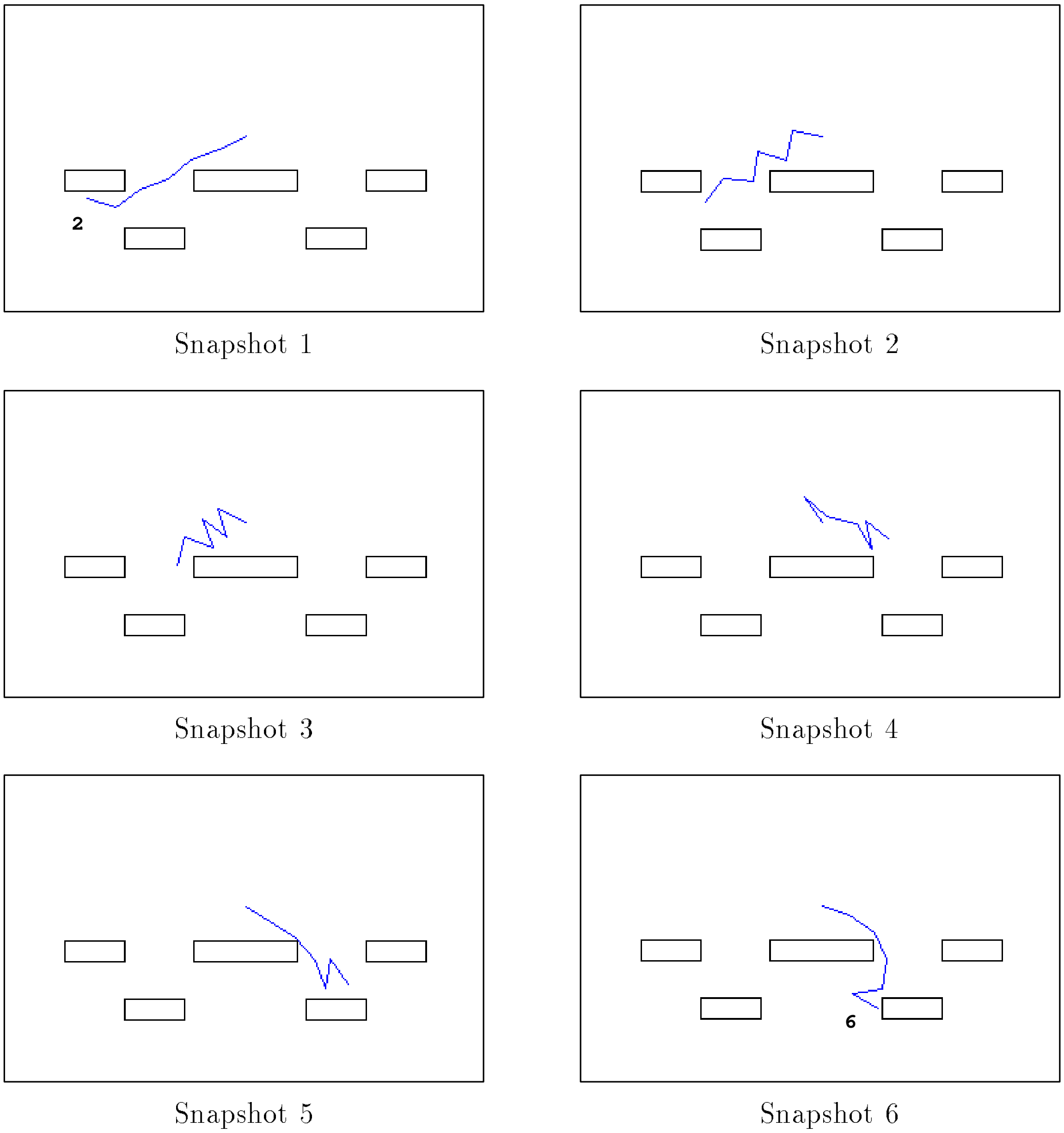


Fig. 30. Snapshots of the path between configurations 2 and 6 in Figure 27.

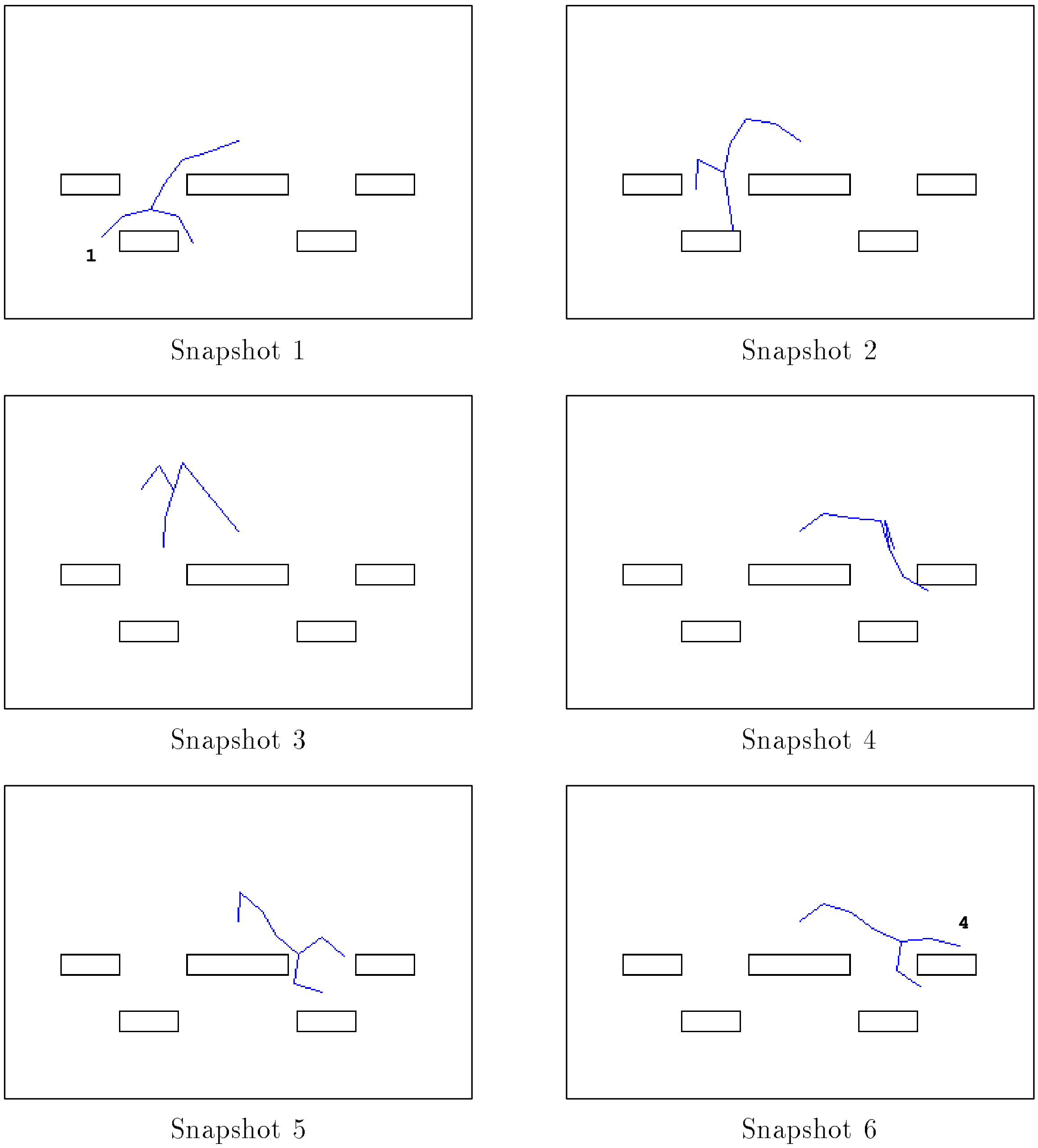


Fig. 31. Snapshots of the path between configurations 1 and 4 in Figure 29.

CHAPTER VI

CONCLUSION

A. Summary

A new obstacle-based probabilistic roadmap method is described for motion planning that can be used for collision-free path planning. The method presented is a two-step method. During a preprocessing stage a roadmap is constructed in C-space. The planning stage involves connecting the start and goal configurations to the roadmap and then finding a path in the roadmap between the connection points using breadth-first search. The main novelty of this method is the generation of candidate nodes in the preprocessing. Instead of sampling configurations randomly in C-space [33, 34], the roadmap candidate nodes are generated on the surface of C-obstacles. Using this method, high quality roadmaps can be built even in crowded C-space. Previous methods performed poorly in such environments.

A path planner was implemented for 2D environments with polygonal obstacles and articulated robots using the obstacle-based probabilistic roadmap method. Extensive experiments were performed for various types of robots and for environments which have crowded C-spaces with long, narrow passages. The results show that the method works well on sample problems which were designed to have high-dimensional crowded C-space and are generally regarded as difficult to solve using previous techniques.

Our method has somewhat higher complexity in the candidate node generation phase than previous methods [22, 34]. However, the extra cost is seen to pay off in the roadmap connection phase which is by far the dominating component of the preprocessing. Thus the total complexity of the method is expected to be lower than other methods. The other advantage of this method is C-obstacle dependency. Since the candidate nodes are generated according to each C-obstacle, it is rather easy to make the method adaptive to deal with each individual C-obstacle, e.g., to generate more nodes on a 'difficult' C-obstacle, which is hard for previous methods which do not utilize the C-obstacle dependency.

A potential drawback of the method is that it is hard to generate candidate nodes 'uniformly' on the surfaces of the C-obstacles. One reason is the difficulty of defining a 'uniform' metric in high-dimensional, non-Euclidean C-space. Although we could borrow some concepts from Euclidean space and use them locally in C-space, applying them in our environments is highly complicated and their implementation would be expensive. Another difficulty is that the binary search used to find the candidate nodes on the C-obstacle surfaces is not perfect. This is also due to the fact that C-space is high-dimensional and non-Euclidean. More subtle methods should be used for environments when this is seen to be a problem.

Another disadvantage of the method is that the numerical parameters such as the number of directions for each obstacle (m), the number of attempts in roadmap connection (k), etc., are chosen manually and there is no obvious method to predict the values of these parameters. An obvious enhancement is to make the preprocessing iterative so that it adapts to the difficulty of the problem.

B. Future Work

Future extensions of the OBPRM method are planned for contact tasks, three-dimensional workspaces, and dynamic environments (in which obstacles may move). Since all the candidate nodes in the roadmap are on the surfaces of C-obstacles, the configurations they represent must have basic contact with some obstacle(s). It would be interesting to see how this method performs for planning contact tasks and object manipulation. In this case local planning will be done on C-obstacle surfaces.

When extending the planner to a 3D workspace, the only module that needs to be changed is the collision detection method. Since collision detection is a primitive operation in path planning, the efficiency of the collision detection algorithm will determine the efficiency of the path planner. In a 3D workspace, the representation of the robot and the obstacles is much more complex than with a 2D workspace. Since the collision detection method is external to the OBPRM method, a similar performance gain is expected for the 3D version of the path planner.

Parallelization of the OBPRM method is natural due to its inherent parallelism. Preliminary results from an implementation of a parallel path planner on an SGI Power Challenge (a distributed shared-memory machine with 16 processors) are encouraging. In particular, we have achieved linear speedup with a simple implementation of the OBPRM method.

For dynamic environments, the OBPRM method could still be used as long as the change in the environment is incremental. In this case, the majority of the roadmap could be reused and additional candidate nodes would be generated and connected to maintain the connectivity of the roadmap.

REFERENCES

[1] N. M. Amato and Y. Wu, "A randomized roadmap method for path and manipulation planning," in *Proc. IEEE Internat. Conf. Robot. Autom.*, Minneapolis, MN, Apr.22–28, 1996, vol.1, pp.113–120.

[2] T. Asano, L. Guibas, J. Hershberger, and H. Imai, "Visibility-polygon search and Euclidean shortest path," in *Proc. of the 26th Annual Symposium on the Foundations of Computer Science*, Portland, OR, Oct.21–23, 1985, pp.155–164.

[3] F. Aurenhammer, "Voronoi diagrams – A survey of fundamental geometric data structure," *ACM Computing Surveys*, vol.23, no.3, pp.345–405, Sept., 1991.

[4] J. Barraquand, B. Langlois, and J.-C. Latombe, "Numerical potential field techniques for robot path planning," *IEEE Trans. Sys., Man, Cybern.*, vol.22, no.2, pp.224–241, Mar./Apr., 1992.

[5] J. Barraquand and J.-C. Latombe, "Robot motion planning: A distributed representation approach," *Internat. J. Robot. Res.*, vol.10, no.6, pp.628–649, Dec., 1991.

[6] J. F. Canny, "A new algebraic method for robot motion planning and real geometry," in *Proc. of the 28th Annual Symposium on the Foundations of Computer Science*, Los Angeles, CA, Oct.12–14, 1987, pp.39–48.

[7] J. F. Canny and J. Reif, "New lower bound techniques for Robot motion planning problems," in *Proc. of the 28th Annual Symposium on the Foundations of Computer Science*, Los Angeles, CA, Oct.12–14, 1987, pp.49–60.

[8] D. J. Challou, M. Gini, and V. Kumar, "Parallel search algorithms for robot motion planning," in *Proc. IEEE Internat. Conf. Robot. Autom.*, Atlanta, GA, May 2–7, 1993, vol.2, pp.46–51.

[9] H. Chang and T. Y. Li, "Assembly maintainability study with motion planning" in *Proc. IEEE Internat. Conf. Robot. Autom.*, Nagoya, Japan, May 21–27, 1995, pp.1012–1019.

[10] P. C. Chen and Y. K. Hwang, "SANDROS: A motion planner with performance proportional to task difficulty," in *Proc. IEEE Internat. Conf. Robot. Autom.*, Nice, France, May 12–14, 1992, pp.2346–2353.

[11] Y. C. Chen and M. Vidyasagar, "Optimal trajectory planning for planar n-link revolute manipulators in the presence of obstacles," in *Proc. IEEE Internat. Conf. Robot. Autom.*, Philadelphia, PA, Apr. 24–29, 1988, pp.202–208.

[12] T. H. Cormen, C. E. Leiserson and R. L. Rivest, *Introduction To Algorithms*, McGraw-Hill, New York, 1990.

[13] B. R. Donald, "A search algorithm for motion planning with six degrees of freedom," *Artif. Intell.*, vol.31, no.3, pp.295–353, Mar., 1987.

[14] B. Faverjon, "Hierarchical object models for efficient anti-collision algorithm," in *Proc. IEEE Internat. Conf. Robot. Autom.*, Scottsdale, AZ, May 14–19, 1989, pp.333–340.

[15] B. Faverjon and P. Tournassoud, "A local approach for path planning of manipulators with a high number of degrees of freedom," in *Proc. IEEE Internat. Conf. Robot. Autom.*, Raleigh, NC, Mar. 31–Apr. 3, 1987, pp.1152–1159.

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

[17] M. R. Garey, D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, New York, 1979.

[18] M. Herman, "Fast, three-dimensional, collision-free motion planning," in *Proc. IEEE Internat. Conf. Robot. Autom.*, San Francisco, CA, Apr. 7–10, 1986, pp.1056–1063.

[19] J. Hopcroft, D. Joseph and S. Whiteside, "On the movement of robot arms in 2-dimensional bounded regions," *SIAM J. Comput.*, vol.14, no.2, pp.610–629, May, 1985.

[20] Y. Hwang and N. Ahuja, "Gross motion planning – a survey," *ACM Computing Surveys*, vol.24, no.3, pp.219–291, Sept., 1992.

[21] D. H. Joseph and W. H. Plantiga, "On the complexity of reachability and motion planning questions" in *Proc. of the First ACM Symposium on Computational Geometry*, Baltimore, MD, 1985, pp.62–66.

[22] L. Kavraki and J. C. Latombe, "Randomized preprocessing of configuration space for fast path planning," in *Proc. IEEE Internat. Conf. Robot. Autom.*, San Diego, CA, May 8–13, 1994, pp.2138–2145.

[23] L. Kavraki, P. Švestka, J.-C. Latombe, and M. Overmars, "Probabilistic Roadmaps for Path Planning in High Dimensional Configuration Spaces," to appear in *IEEE Trans. on Robotics and Automation*, (Also available as STANFORD-CS-TR-94-1519).

[24] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Proc. IEEE Internat. Conf. Robot. Autom.*, St. Louis, MO, Mar. 25–28, 1985, pp.500–505.

[25] P. Khosla and R. Volpe, "Superquadric artificial potentials for obstacle avoidance and approach," in *Proc. IEEE Internat. Conf. Robot. Autom.*, Philadelphia, PA, Apr. 24–29, 1988, pp.1778–1784.

[26] K. Kondo, "Motion planning with six degrees of freedom by multistrategic, bidirectional heuristic free space enumeration," *IEEE Trans. on Robotics and Automation*, vol.7, no.3, pp.267–277, June, 1991.

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

[28] T. Lozano-Pérez, "Automatic planning of manipulator transfer movements," *IEEE Trans. on Sys. Man, Cybern.*, vol.11, no.10, pp. 681–698, Oct., 1981.

[29] T. Lozano-Pérez, "Spatial Planning: A configuration space approach," *IEEE Trans. on Computers*, vol.32, no.2, pp.108–120, Feb., 1983.

[30] T. Lozano-Pérez, "A simple motion planning algorithm for general robot manipulators," *IEEE J. Robotics Auto.* , vol. RA-3, no. 3, pp.224–238, June 1987.

[31] A. A. Maciejewski and C. A. Klein, "Obstacle avoidance for kinematically redundant manipulators in dynamically varying environments," *Internat. J. Robot. Res.*, vol.4, no.3, pp.109–117, Fall, 1985.

[32] W. Newman and N. Hogan, "High speed robot control and obstacle avoidance using dynamic potential function," in *Proc. IEEE Internat. Conf. Robot. Autom.*, Raleigh, NC, Mar. 31–Apr. 3, 1987, pp.14–247.

[33] M. Overmars, "A random approach to path planning," Technical Report RUU-CS-92-32, Computer Science, Utrecht University, The Netherlands, 1992.

[34] M. Overmars and P. Švestka, "A probabilistic learning approach to motion planning," in *Proc. Workshop on Algorithmic Foundations of Robotics*, San Francisco, CA, 1994, pp.19–37.

[35] B. Paden, A. Mees and M. Fisher, "Path planning using a Jacobian-based freespace generation algorithm," in *Proc. IEEE Internat. Conf. Robot. Autom.*, Scottsdale, AZ, May 14–19, 1989, pp.1732–1737.

[36] J. Reif, "Complexity of the mover's problem and generalizations," in *Proc. of the 20th Annual Symposium on the Foundations of Computer Science*, San Juan, Puerto Rico, Oct. 29–31, 1979, pp.421–427.

VITA

Yan Wu was born on May 2, 1968, in Shandong, China. He received his Bachelor and Master degrees in Computer Engineering from Xian Jiaotong University, Shaanxi, China, in 1989 and 1992, respectively. He was employed by the China Aviation Technology Import/Export Company (CATIC), Shenzhen, China, for two years before he began his studies at Texas A&M University in 1995.

Mr. Wu's current address is 1 Hensel, #Y3E, College Station, TX 77840 and he can be reached by email at yanwu@cs.tamu.edu.

The typist for this thesis was Yan Wu.