Motion Planning For Serial Manipulators With Many Degrees Of Freedom: Implementation Of A Sequential Search Strategy

by

Zhenping Guo

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in the School

of

Engineering Science

Simon Fraser University

January, 1992

APPROVAL

NAME: Zhenping Guo

DEGREE: Master of Applied Science (Engineering Science)

TITLE OF THESIS: Motion Planning For Serial Manipulators
With Many Degrees Of Freedom:
Implementation Of A Sequential Search Strategy

EXAMINING COMMITTEE:

Chairman: Dr. John Dill

Dr. Kamal K. Gupta
Senior Supervisor

Dr. John Jones
Supervisor

Dr. Shahram Payandeh
Supervisor

Dr. William Havens
Examiner

DATE APPROVED: 21 January, 1992

PARTIAL COPYRIGHT LICENSE

I hereby grant to Simon Fraser University the right to lend my thesis, project or extended essay (the title of which is shown below) to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users. I further agree that permission for multiple copying of this work for scholarly purposes may be granted by me or the Dean of Graduate Studies. It is understood that copying or publication of this work for financial gain shall not be allowed without my written permission.

Title of Thesis/Project/Extended Essay

MOTION PLANNING FOR SERIALS MANIPULATORS WITH MANY DEGREES OF FREEDOM:

IMPLEMENTATION OF A SEQUENTIAL SEARCH STRATEGY

Author:

(signature)

GUO, Zhenping

(name)

January 21, 1992

(date)

Abstract

In this thesis, we deal with the robot motion planning problem, i.e., planning collision-free motions for a manipulator arm in an environment filled with stationary obstacles. We have developed and implemented a motion planner that can plan collision-free motions for serial manipulator arms with many degrees of freedom among stationary obstacles. Our planner is based on a sequential search strategy that exploits the serial nature of manipulator arms. The basic idea behind this approach is to plan the motion of each link successively starting from the base link. Given that the motion of links until link i (including link i) has been planned, that is, the path of one end (the proximal end) of link $i+1$ is determined, the motion of link $i+1$ is now planned along this collision-free path by controlling the degree of freedom associated with it – a two-dimensional motion planning problem. Therefore, in the simplest case, our strategy results in one one-dimensional (the first link is degenerate) and $n-1$ two-dimensional planning problems instead of one n-dimensional problem for an n-link manipulator arm. The search strategy leads to a fast and efficient algorithm, and is especially suited for manipulator arms with many degrees of freedom. Furthermore, a backtracking mechanism has been incorporated in our motion planner that makes it more effective (but slower) in cluttered environments. This mechanism provides a parameter which can be used to trade off completeness of the planner with its speed. Our motion planner runs on SUN Sparc stations and is written in C. The run times are in the order of a few minutes. We have demonstrated the effectiveness of our strategy with some interesting and difficult examples of motion planning for several arms with many degrees of freedom.

To My Family

Acknowledgements

My sincere thanks to my supervisor Dr. Kamal K. Gupta for providing me the opportunity to work on this project, guiding and encouraging me to work in very challenging research topics and also supporting me throughout the course of this research. I am very grateful to my committee members, Dr. John Dill, Dr. John Jones, Dr. Shahram Payandeh, and Dr. William Havens, for their assistance and helpful hints. In addition, I was lucky enough to obtain useful help from other individuals. Dr. Binnay Bhattacharya gave me guidance to solve the computational geometry problems. Doug Girling helped me to transfer simulation results from the Sun station to the PUMA controller. Edgar Velez gave me timely help to revise the thesis. Also, many thanks to the graduate secretary, Brigitte Rabold, for her kind help.

Finally, I would like to express my appreciation to my colleagues, Zhu XinYu, Zhu XiaoMing and Xu ZhuKang, for their valuable discussions during this research. Special thanks to my friends for their encouragement and support during the preparation of my thesis.

CONTENTS

LIST OF FIGURES

CHAPTER 1

Introduction

Future generations of robots will be considerably more autonomous than present robotic systems. A main objective of research in robotics is to endow robotic systems with basic capabilities they will need to operate in an intelligent and autonomous manner. These improved capabilities fall into three broad categories, (i) *Sensing* – The robot system should be able to gather information about its workspace through a variety of sensing devices (visual, tactile, or proximity sensing) and analyze and transform the raw sensory data into a model of the environment; (ii) *Planning* – This model is then used to plan tasks the robot is commanded to execute; (iii) *Control* – The robot system will finally obtain a low-level control loop that monitors the actual execution of each planned substep of the task.

The aim at the planning stage is to allow the robot's user to specify a desired activity at a very high level (called top-level planning, or task-level planning) and have the robot system fill in the missing low-level details [10]. A task-level robot planning system is one that can be instructed in terms of task-level goal, such as

"Grasp part A and place it inside box B." This type of specification contrasts sharply with that required for existing industrial robot systems, which insist on a complete specification of each motion of the robot and not simply a description of a desired goal.

To automatically perform such tasks, the robot needs the capability to plan collision-free motions. The most basic version of the motion planning problem is to find a collision-free path for a manipulator arm in a three-dimensional environment filled with static obstacles. Note that there are a number of variations of the basic problem, e.g., the moving object may be arbitrary arms or collections of arms, as well as objects (e.g., mobile robot like spacecraft) moving freely in space [2], [3]. In this thesis, we focus on the static and purely geometric version of the motion planning problem for a manipulator arm. We have developed and implemented a motion planner that can plan collision-free motions for a manipulator arm with many degrees of freedom. In the next section, we formally define the motion planning problem, and briefly discuss some aspects of it, and then give a brief overview of our motion planner.

1.1 The Problem

In its simplest form, the motion planning problem considered in this thesis, is defined as follows [30]. Let A be a manipulator arm comprising a collection of rigid links which are attached to each other at certain joints. Suppose A has a total of n degrees of freedom, i.e., each placement of A can be specified by n real parameters, each representing a joint angle of the manipulator arm. Suppose further that A is free to

move in a workspace amidst a collection of obstacles whose geometry is known to the robot system.

The motion planning problem for A is

Given an initial placement P_i and a desired target placement P_f of A, determine whether there exists a continuous obstacle-avoiding motion of A from P_i to P_f, and, if so, plan such a motion.

We assume that the robot manipulator arms considered in this thesis, are articulated devices made up of a series of rigid *links* connected by one-degree-of-freedom joints, the joint motions are either rotational or translational. Such manipulators are also called serial manipulators, however, for brevity we will use the term manipulator through out the thesis. The position of all links of the rigid robot arm are completely specified by the values of the joint parameters, known collectively as the *joint angles.* Any set of parameters that uniquely specifies the position of every part of the arm is called a *configuration*, and the space defined by those parameters is the C-space. Also, we call the collection of configurations that produce collisions the C-space obstacles. It is clear that, for a manipulator arm with n degrees of freedom, the robot's joint space is an n-dimensional C-space [9], [11], [12]. Once such a representation has been obtained, the actual motion planning problem reduces to searching for a path from the initial configuration to the final configuration in the C-space.

Most of the approaches to motion planning implicitly or explicitly compute the C-space of the manipulator and then search it for a path. There are two major problems with the configuration space based approaches: (i) the configuration space

obstacles are highly complex to represent, and (ii) the dimensionality of the configuration space is fairly high for most practical manipulators, mostly greater than six. An answer to the first problem is a discrete representation of the configuration space ([9], [14], [15], [16]). However, according to many complexity results, the "big barrier" is the high number of degrees of freedom. The complexity of the basic motion planning problem is exponential in the degrees of freedom of the manipulator [9]. For manipulator arms with a large number of degrees of freedom, the motion planning problem becomes computationally intractable, and it isn't likely that efficient worst-case solutions will ever be found. Many approaches are hopelessly inefficient in the worst case, and although their development is significant from a theoretical point of view, their implementations do not exist. For example, Lozano-Pérez [9] implemented only a three degrees of freedom example since he treated the last few degrees as one (using a bounding box to cover them); Kondo [16] mentioned that for a high number of degrees (more than six) of freedom arm and the worst cluttered environment, it needs more memory and more computer time. Recently, Barraquand and Latombe ([17], [18], [19]) have suggested a planner based on numerical potential fields and random search for manipulators with many degrees of freedom. Random search, however, has its problems. For instance, the run times may vary drastically even for the same example. From all the discussion above we conclude that to solve the motion planning problem fast and efficiently, one needs to devise general yet efficient search strategies.

A deterministic approach to this problem of high dimensionality has recently been suggested by Gupta [1] – a sequential search strategy. The basic idea of this approach is to sequentially plan the motion of each link, starting from the base link. To briefly recapitulate, suppose that the motion of links until link i (including link i)

has been planned. This already determines the path of one end (the proximal end) of link $i+1$. The motion of link $i+1$ is now planned along this path by controlling the degree of freedom associated with it – a single link motion planning problem. This strategy (without backtracking) results in n single link motion planning problems instead of one n-dimensional problem for an n link manipulator arm. If along the path (determined by the previous links) no motion exists for the current link, the planner backtracks and chooses another path for the previous link. Along this new path for the previous link, it searches for a path for the current link. This approach completely avoids the problem of representing and searching a high-dimensional C-space. However, in some situations the planner may fail to find a collision-free path in very cluttered situations even though such a path may exist, i.e., the algorithm is not complete.

Based on this approach, we have developed and implemented a motion planner for several types of robot manipulator arms to plan collision-free motions in different kinds of cluttered robot working environments. The motion planner is written in C and its run time is in the order of a few minutes on a Sparc station. The asymptotic complexity of the motion planner with backtracking mechanism is given by

$$O(nm^{2k+2}),$$

where n is the number of degrees of freedom, m the number of nodes in the two-dimensional subspace for each robot link, and k the level of backtracking. Certainly, for a given k, the complexity will be polynomial in m.

In summary, our planner has a number of advantages: (i) it completely avoids the difficulty of representing n-dimensional C-space obstacles, (ii) it is efficient and practical – runs at the most in the order of a few minutes, even for manipulators

with large number of degrees of freedom, (iii) it is deterministic (in the sense that it does not use stochastic search techniques), and (iv) it provides a mechanism for trading off execution speed of the planner with its completeness.

We emphasize that the main contribution of this thesis lies in implementing the sequential search approach of [Gupta 90] in 3-D. With this implementation, we have thoroughly tested the efficiency of the sequential approach.

1.2 Organization Of The Thesis

This thesis is organized as follows. We begin in chapter 2 by overviewing algorithms for robot motion planning in the recent years, and briefly give a basic catalogue of the algorithms. Chapter 3 contains the overview of our sequential search strategy approach. In chapter 4, we concentrate on the implementation of our algorithm, including how to construct the sub-spaces for each of the robot links; how to search for a collision-free path, how to parameterize the generated path and how to perform backtracking search. Chapter 5 will show the experimental results. Finally, chapter 6 gives a summary and suggestions for the future research.

CHAPTER 2

Survey

Techniques for automatic planning of robot motions have been extensively studied over the last several years as one of the central problems in task level robot programming, both from theoretical interests and from practical requirements. In this chapter, we will first give the basic catalogue, and then present a brief survey of recent developments in motion planning algorithms and their implementations. Our survey concentrates on practical approaches to motion planning for manipulator arms.

2.1 Complexity

The complexity of motion planning problems has been studied quite extensively in past fifteen years or so ([32], [30]). In a recent fundamental work, Canny [37] has improved upon the previous result of Schwartz and Sharir [28]. Canny shows that

the motion planning problem for a manipulator arm with n degrees of freedom under m geometric constraints (usually the number of faces, edges, vertices etc. in the obstacles and manipulator arms), can be solved in time $O(m^n \log m)$. Numerous other results [9] show that the complexity of motion planning is exponential in the degrees of freedom of the robot. Therefore, for a manipulator arm with an arbitrarily large number of degrees of freedom, the problem becomes computationally intractable, and it isn't likely that efficient worst-case solutions will ever be found. Intuitively, this is because when n is large, the C-space is a high-dimensional space with irregular boundaries, making it hard to build and search efficiently.

Clearly, to develop practical motion planning algorithms, one needs to devise efficient representations and search strategies to break this barrier of complexity. Following are several practical approaches to develop motion planners for manipulator arms.

2.2 Practical Approaches

Many planners for manipulator arms have been developed ([12], [22], [31]). Some of the more recent ones are Lozano-Pérez's Configuration Space Approach ([9], [13]), Faverjon and Tournassoud's Octree Approach ([14], [15]), Barraquand and Latombe's Potential Field Approach ([17], [18], [19]), and Kondo's Enumeration Approach [16]. The major differences among these approaches are in two aspects: (i) how to build and represent the configuration space, and (ii) how to search for a collision-free path. The methods used in these approaches can be roughly classified into two categories, (i) Configuration Space based approaches, and (ii) Potential Field based approaches.

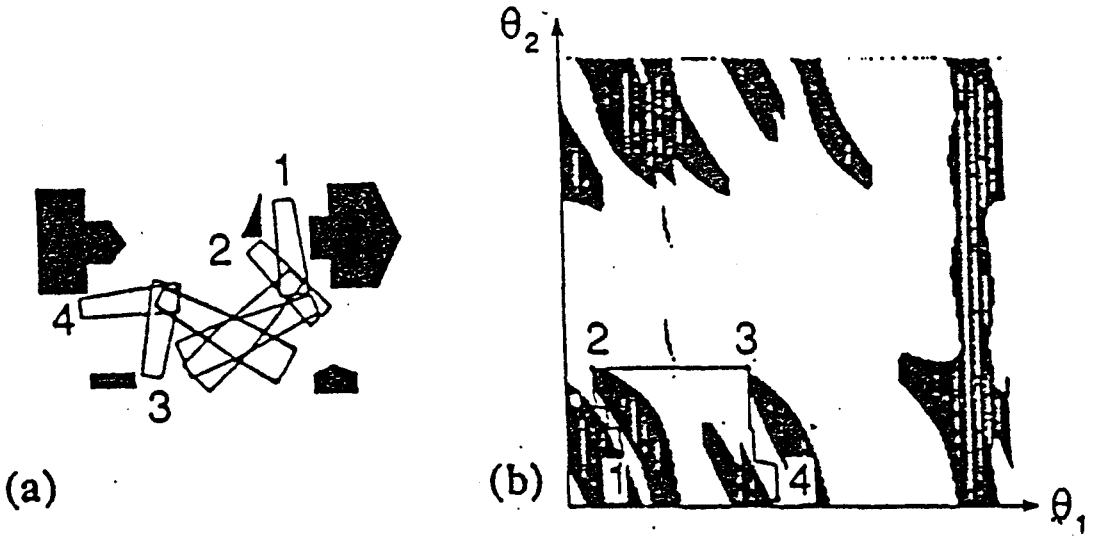


Figure 2.1: C-space obstacles for a simple robot. (a) A robot with two rotational joints, and its obstacles. (b) The corresponding C-space obstacles (the hatched regions). The joint angles are also the labels on the axes of the C-space. A collision-free path in this C-space is shown, together with four configurations of the arm along the path.

2.3 Configuration Space Based Approaches

Planning a collision-free path for an n-degree of freedom manipulator arm is equivalent to calculating a continuous path within a collision-free region in an n-dimensional space defined in terms of the degrees of freedom. Figure 2.1 (this figure is presented in [9]. Author uses it to explain the configuration space) gives a two-dimensional example.

Motion planning based on the C-space approach requires that the obstacles in the robot's workspace should be mapped into its C-space, called C-space obstacles, which represent the configurations of the manipulator arm that would cause collisions. Having formed the C-space obstacles, there are two main approaches to search for a collision-free path, (i) free-space based and (ii) V-graph based methods.

For the free-space based method, the free C-space, which is defined to be the complement of the C-space obstacles, should be characterized explicitly. Having found the free-space regions, a region graph is built where the nodes are regions and links indicate regions with a common boundary. Then, a collision-free path is searched in the region graph from the region containing the start point to the region containing the target point. For the V-graph based method, a visibility graph should be built, which is composed of nodes and edges. The set of nodes is the set of vertices of the C-space obstacles, including the source node (as the initial position) and the target node (as the final position), and each edge between two nodes represents that these two nodes can be visible to each other. Then the planner searches this V-graph for a collision-free path from the source node to the target node.

Many planners have been developed using the C-space approach ([9], [14], [15], [16]). We briefly review three important representative planners that have been developed and reported in the literature.

2.3.1 A Simple Motion Planning Algorithm

In [9], Lozano-Pérez showed the simplicity of computing approximate characterizations of the free space for simple manipulators. A conservative approximation

of a C-space obstacle was introduced, which was called a *slice projection.* For an n-dimensional case, a C-space obstacle is a n-dimensional volume, which is represented by a union of (n-1)-dimensional *slice projections.* A *slice projection* of a one-dimensional C-space obstacle is defined by a range of values for one of the defining parameters of the C-space (the joint angles for a manipulator arm with rotary joints). The key step in this simple approach, therefore, is to compute one-dimensional slice projections of C-space obstacles, that is, to determine the range of forbidden values of one joint parameter, given ranges of values for all previous joint parameters.

Using this approach, a planner has been implemented successfully. It has solved problems for manipulator arms with four degrees of freedom. For the case of manipulator arms with more degrees of freedom, Lozano-Pérez suggested that the last few links and the end-effector be replaced by a simple bounding box [9]. For a simple two-link manipulator arm in two-dimensional space (see Figure 2.1), Lozano-Pérez mentioned that the run time is six seconds on a Symbolics 3600 Lisp Machine.

Theoretically, the motion planning problems with n degrees of freedom can be solved. In practice, however, it is difficult to represent a free space in an n-dimensional C-space explicitly. His approach will fail to find a collision-free path in the more cluttered case, such as in the example of PUMA 560, simply because the bounding box covers quite a big free volume.

2.3.2 Octree Approach

Faverjon has used octrees to represent the C-space for a manipulator arm [14]. An octree is a tree of degree eight which describes hierarchically the space contained in a cube that forms the root. The sons of a node are the eight cubes obtained by cutting the father node by planes parallel to the faces and containing its center. The nodes can be labeled as Full, Empty, or Mixed depending on whether they lie entirely in an obstacle, out of all obstacles, or partly in an obstacle. Only the Mixed nodes will be cut again until the minimum size for a cube is reached. Using collision-detection type algorithms, physical obstacles in the robot workspace can be transformed into its joint space. Faverjon implemented this approach for the first three links of a manipulator arm. So the joint space is a three-dimensional C-space. He divided the whole space into $64 \times 64 \times 64$, and then filled it up using the transforming algorithm. The scheme of searching a short collision-free path is also based on the well known A^* algorithm [42]. Unfortunately, the approach can not be directly extended to planning a collision-free path for manipulator arms with large number of degrees of freedom, since the memory requirements grow exponentially with the degrees of freedom.

In 1987, Faverjon and Tounassoud [15] implemented a two-level local-global planner. The global planner is essentially a very coarse discretized representation of the full C-space. Transitions between the adjacent cells are weighted (and updated) by the probability for the local planner to succeed in moving the system from one cell to another. These probabilities are used by the global planner (a minimum cost finding algorithm) to generate sub-goals for the local planner. The local planner is based on a variant of the potential field technique. They have reported some good results

for a full six degrees of freedom arm carrying a bulky load, and for a ten degrees of freedom manipulator among vertical obstacles. Although it is possible to apply this technique to manipulators with many degrees of freedom, the global planner resolution would be very coarse for a given memory size and the probabilities may not converge. For example, they use a resolution of 16 degrees for a six degree of freedom problem; in fact, for much larger number of degrees of freedom problems, the global planner discretization will be so coarse that it will be virtually useless.

2.3.3 Enumeration Approach

Kondo [16] has suggested another approach to motion planning for a robot manipulator arm. The basic idea in this approach is to restrict the free space concerning a free path and to avoid executing unnecessary collision detections. Kondo implemented this approach for a six-degree of freedom manipulator arm. In the approach, the six-dimensional C-space is, initially, equally quantized into cells by placing a regular grid, and then, the free-space cells connecting the initial and final configurations are enumerated based on the heuristic graph search algorithm, which is similar to the A^* algorithm [42]. These cells are categorized into three types: a free space cell belonging to a collision-free configuration, an obstacle cell which belongs to a configuration that causes collisions, and an unknown cell for which the collision status is not yet known.

Not all cells are enumerated in this approach. Which cell will be enumerated depends on the evaluated strategy. There are different strategies of searching a collision-free path defined by heuristic functions. The efficiency of each strategy is evaluated during free-space enumeration, and the more promising one is automat-

ically selected and executed. A collision detection procedure is independent of the search procedure and is called every time the necessity arises for checking whether the cell is collision-free or not. However, the total number of necessary collision detections for free-space enumeration mainly depends on the most efficient search strategy among the evaluated strategies, and therefore, the free-space cells are efficiently enumerated for an arbitrary moving object in all kinds of workspaces.

Initially, all cells are unknown cells except for the free-space cells containing the initial and final configurations. These two free-space cells are expanded based on a bidirectional heuristic graph search algorithm using multiple search strategies, and the configurations of the expanded unknown cells are checked for collision. Free-space cells are expanded until the initial and final configurations are connected by free-space cells. Finally, the collision-free path can be planned.

The approach is fast when only a few free-space cells are enumerated in the search process. But in the more cluttered workspace, the efficiency of this approach is drastically reduced. Furthermore, for an n degrees of freedom manipulator arm and the number of levels, m, the C-space will contain $2^{n \times m}$ cells. It needs a huge memory, especially for a larger n. Hence the approach can not be directly extended for a large number of degrees of freedom manipulator arm.

2.4 Potential Field Based Approaches

Khatib [20] first used the idea of an artificial potential field for obstacle avoidance. In this approach, the *manipulator moves in a field of artificial forces.* The obstacles exert a repulsive force and the goal position exerts an attractive force on the ma-

nipulator. In this field of forces, the manipulator end-effector (treated as a point) moves to its goal position. This field is called the artificial potential field. The core of this approach is to set up artificial potential fields over the robot workspace, each applying to a specific point on the robot, and the robot moves along the gradient direction generated by this artificial potential field. Unfortunately, this approach eventually leads the search to local minima of the potential and mechanisms need to be devised to escape these minima. In the next section, we review a recent advance in the artificial potential field approach, called numerical potential field approach.

2.4.1 Numerical Potential Field

Barraquand and Latombe ([17], [18], [19]) use a bitmap of the workspace and construct numerical potential fields over the workspace. Each of these potentials applies to a selected point on the robot, called a control point. The overall C-space potential field is a combination of these workspace potential fields. The C-space potential field generally has several local minima, and for a high-dimensional C-space, the Monte-Carlo procedure (which consists of generating random motions) is applied to escape a local minimum. The planner has been successfully applied to manipulators with many degrees of freedom; however, there are some intrinsic limitations in the stochastic approach. For example, the execution times may vary drastically even for the same example for different runs of the planner. Also, the success of probabilistic search seems to stem from the fact that for manipulators with many degrees of freedom, the solution space is also very large. However, this may not be the case in cluttered environments.

From the brief survey given in the previous sections, it is apparent that no

approach has yet satisfactorily overcome the difficulty of solving the motion planning problem for manipulator arms with a larger number of degrees of freedom (greater than 6). In the next chapter, a new approach, called *Sequential Search Approach* [1], will be introduced, which offers one way to develop practical motion planners for manipulator arms with large number of degrees of freedom.

CHAPTER 3

Sequential Search Algorithm Overview

In this chapter, we give a brief overview of the sequential search approach to motion planning, a contribution of Gupta ([1], [4]). Based on this approach, we have developed and implemented a motion planner for several types of robot manipulator arms to plan collision-free motions in different kinds of cluttered robot working environments.

The idea behind the approach is to exploit the serial nature of manipulator arms. A manipulator arm has an inherent ordering of links, from the base link to the last link. The strategy of this approach is to plan, successively, the motion of each link starting from the base link. For example, consider the three-link (planar

manipulator) arm shown in Figure 3.1 (this figure[1] is from [1]). The links have been labeled in increasing order from the base, i.e., the base link is labeled 1, the next link is labeled 2, and the last (free) link is labeled 3. Suppose that the motion of link 1 has been planned. Now, as the first link moves the second link can be treated as a moving robot with another degree of freedom (pure rotation for revolute joints). Thus, its motion (rotation) can be planned to avoid the obstacles. Once the motion of link 2 has been planned, the motion of link 3 can be planned, and so on. In general, suppose that the motion of links till link i (including link i) has been planned. This already determines the path of one end (the proximal end) of link $i+1$. The motion of link $i+1$ is now planned along this path by controlling the degree of freedom associated with it – a single-link motion planning problem. As a result, for a manipulator arm with n degrees of freedom (n revolute or prismatic joints), this strategy converts one n-dimensional motion planning problem into n single-link planning problems (planning for the first link is a degenerate case)

In addition, Gupta and Guo [5] have also developed a backtracking search mechanism for our motion planner to make it more effective in planning collision-free paths in cluttered situations, although the sequential planner described above does solve the difficult problems. In general, the planner must backtrack and re-plan the path for the previous link (say link $i-1$) in case there is no collision-free path for link i. Furthermore, the planner may backtrack more than just the previous link, say k previous links and search again. The parameter k is called the level of backtracking. This backtracking mechanism has been incorporated in our motion planner and has remarkably improved the capabilities of our motion planner. Figure 3.2 shows the

[1]This figure courtesy of Dr. K. K. Gupta

complete structure of our motion planner with the backtracking search mechanism.

Not only is the resulting planner efficient and fast, but we completely avoid the difficulty of representing n-dimensional $t \times \theta$ space obstacles. Our motion planner is, therefore, especially suited for highly redundant arms. The complexity of our planner is given by

$$O(nm^{2k+2}),$$

where n is the number of degrees of freedom, m the number of nodes in two-dimensional sub-space for link i and k the number of level of backtracking searches. Clearly, the run time is polynomial in m, for a given k. Thus, our approach provides a "tuning" parameter k, that can be used to trade off speed with completeness of the planner.

3.1 Terminologies

The definitions used throughout the thesis are given next. The manipulator joint parameters are represented by θ_i, the corresponding links by l_i, $i = 1, 2, \cdots, n$, l_1 being the most proximal link, and l_n the most distal link. The initial and goal configurations of the arm are given as θ_i^s, and θ_i^g, $i = 1, \cdots, n$. $\mathbf{q_i}$ denotes the i-dimensional joint space vector $(\theta_1, \theta_2, \cdots, \theta_i)$. Thus $\mathbf{q_i^s}$ denotes the vector of start configurations of the first i joints, i.e., $\mathbf{q_i^s} = (\theta_1^s, \theta_2^s, \cdots, \theta_i^s)$, and $\mathbf{q_i^g}$ denotes the vector of their goal configurations, i.e., $\mathbf{q_i^g} = (\theta_1^g, \theta_2^g, \cdots, \theta_i^g)$. Let $\mathbf{r_i}$ be the reference vertex (in three-dimensional Cartesian space) for link l_i. The base of the first link is assumed to be fixed at $\mathbf{r_1}$. Let $\mathbf{r_i}(t)$ denote the trajectory of the reference vertex $\mathbf{r_i}$, however, we will never need to represent it explicitly. Instead, we will represent

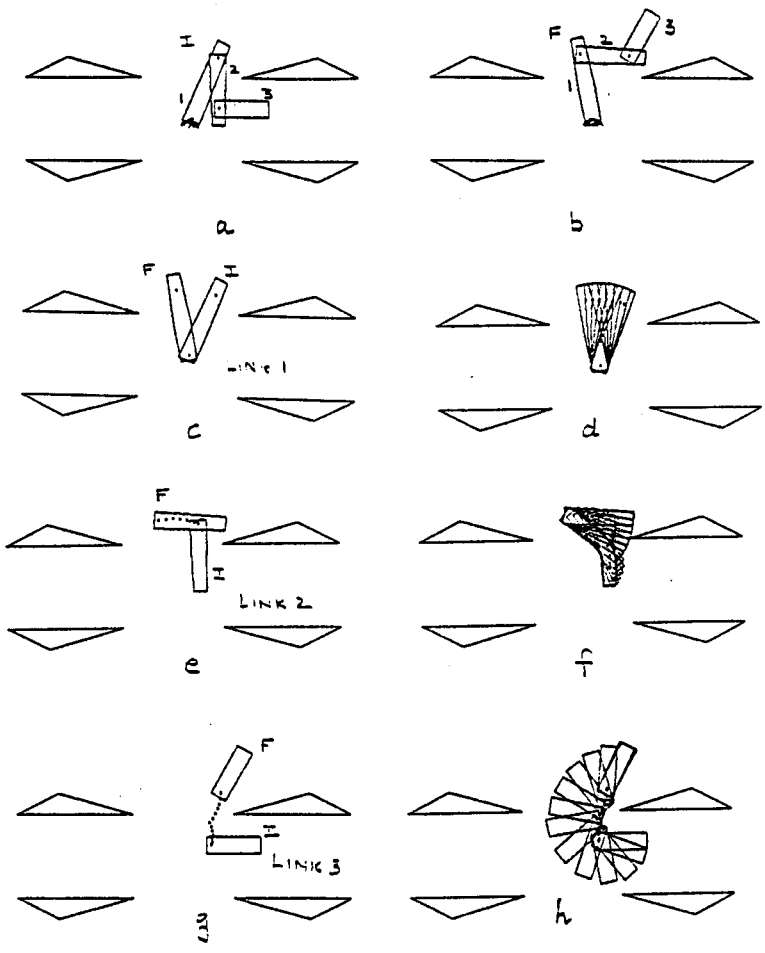


Figure 3.1: Decomposing the motion planning problem for an n link manipulator into n simpler single-link problems. The figure shows a three link example. The initial configuration is shown in (a), and the final configuration in (b). Planning proceeds from the base link (link 1) to the last link (link 3). First the motion of link 1 is planned (c) and (d). This motion determines the motion of a reference point (the point where link 2 is attached to link 1) on link 2. The angle of link 2 is now planned to avoid collisions with obstacles (e) and (f). This motion of link 2 determines the motion of a reference point (the point where link 3 is attached to link 2) on link 3. The angle of link 3 is now planned (g) and (h). This process is repeated n times for an n-link arm, resulting in n single-link motion planning problems.

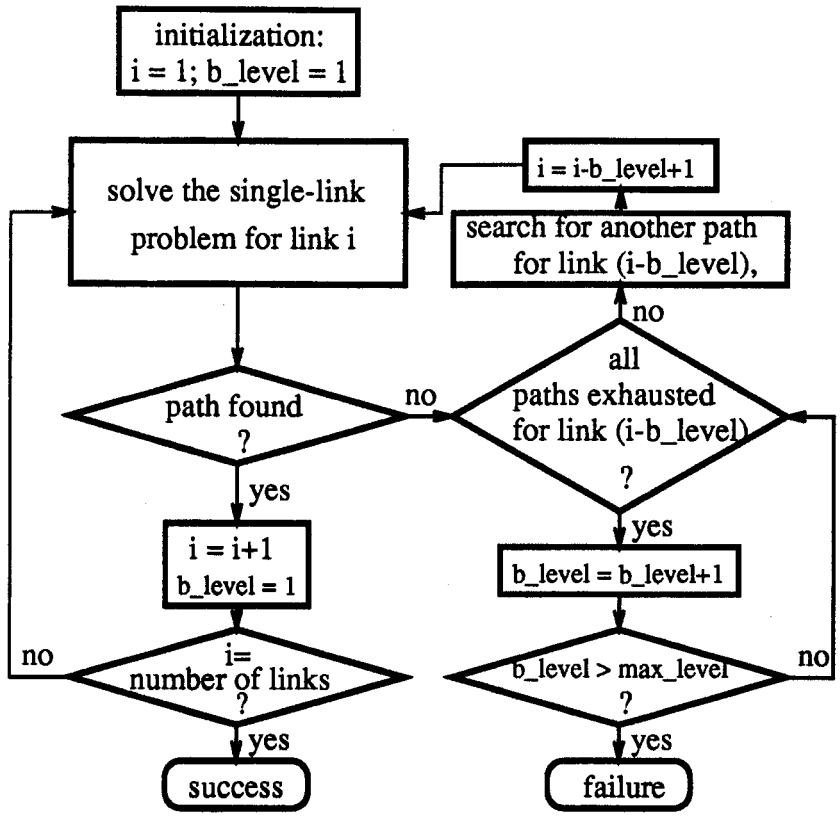


Figure 3.2: The structure of our motion planner with the backtracking search mechanism. In the figure, *b_level* represents the current level of backtracking; *max_level*, the maximum allowed level of backtrackings. Note that $i - b\_level > 1$ should be satisfied since the planner doesn't backtrack to the first link.

the i-dimensional joint space trajectory as a vector valued function $\mathbf{q}_i(t) = (\theta_1(t), \theta_2(t), \cdots, \theta_i(t))$: $[0,1] \rightarrow [\mathbf{q}_i^s, \mathbf{q}_i^g]$. Note that given $\mathbf{q}_{i-1}(t)$, $\mathbf{r}_i(t)$ is easily computed using forward kinematics. Further, we will distinguish between the mapping $\mathbf{q}_i(t)$ and the graph (trace) $\mathbf{Q}_i$ of the mapping, where $\mathbf{Q}_i = (\mathbf{q}_i(t), t)$.

Now we concentrate on the motion planning scheme used in our approach for the single-link problem. Assume that the motion for the previous $i-1$ links has already been planned and is specified as a function of a discretized parameter $t_i = [0,1]$, i.e., $\theta_{i-1}(\frac{k}{M})$ is known for $k = 0, 1 \ldots M-1$. M is the total number of samples of the free path, and is given by user. Note that this completely determines the path of $\mathbf{r}_i$, and as it moves along this path, the problem now is to adjust the angle θ_i of link i to avoid the obstacles in its way. In general, for a single link, the scheme adapted here includes four stages: (1) computing forbidden regions, (2) building a visibility graph, (3) searching a collision-free path, and (4) parameterizing the collision-free path for the next link.

- Stage 1: Computing $t_i \times \theta_i$ space obstacles

The physical obstacles give rise to constraints on the angle θ_i of the link as a function of t_i. These constraints appear as forbidden regions in $t_i \times \theta_i$ space. Thus, the physical obstacles are transformed into forbidden regions in $t_i \times \theta_i$ space, and the rigid solid link, link i, becomes a moving point in the $t_i \times \theta_i$ space. Hence, we always build a two-dimensional $t_i \times \theta_i$ space for each single link of a manipulator arm. The technique of computing $t_i \times \theta_i$ space obstacles will be discussed in chapter 4, and is based on methods in [9].

- Stage 2: Building a visibility graph

The forbidden regions are approximated as polygons in $t_i \times \theta_i$ space. A visibility graph (V-graph) is built in $t_i \times \theta_i$ space. Note that we could also use the free-space based methods.

- Stage 3: Searching for a collision-free path

We use breadth-first search [44] as our V-graph searching algorithm.

- Stage 4: Discretizing the collision-free path

The path obtained after the V-graph search is piece-wise linear. However, to plan the path for the next link, we need to parameterize the path, i.e., given $\mathbf{Q}_i(t_i)$, we compute $\mathbf{Q}_i(t_{i+1})$. The parameterization of the collision-free path will be discussed in chapter 4.

There is no doubt that our sequential search strategy leads to significant improvement in computation time simply because, for a manipulator arm with n degrees of freedom, it solves several single-link motion planning problems rather than one n-dimensional problem. The run time for the planner increases linearly with n (n is the number of joints of a manipulator arm) for a given level of backtracking. This explains the apparent success of our planner for a high number of degrees of freedom problem. Note that our planner is incomplete and may not find a collision-free path even if one exists. However, we believe that in most cases it will.

CHAPTER 4

Implementation

In this chapter, we describe our detailed implementation of the sequential motion planner. In section 1, we briefly review the robot coordinate system and the geometric models and the representations of robot links and obstacles. Section 2 overviews the method of calculating forbidden ranges (the range of angles in which the contact occurs). Section 3 will describe the method of forming $t_i \times \theta_i$ space obstacles. In section 4, it will be explained how to search for a collision-free path. In section 5, we explain how the extended segments work. In section 6, the parameterization of the free path will be discussed. Section 7 shows efficiency of the backtracking mechanism. Section 8 gives a brief discussion of the complexity of our motion planner.

4.1 Geometric Representations

In this section, we briefly review the robot coordinate system, define the geometric models of the robot links and obstacles, and then give the basic representations of the robot links and obstacles.

In order to describe the position and orientation of a robot link in the workspace, we attach a coordinate system, called a *frame*, rigidly to that link. Then we proceed to describe the position and orientation of this frame with respect to some reference coordinate system. In this thesis, we set the base frame of the robot coincident with its workspace coordinate system, the reference frame. Figure 4.1 shows the definition of various frames. Appendix A gives more details of the coordinate systems and the transformation between them.

We define the geometric models of the robot links and obstacles as convex polyhedra, which may have an arbitrary number of faces and vertices. For some other models of robot links and obstacles, such as cylinders and spheres, we may use a "bounding box" to cover those models. The models are represented by the list of faces and vertices as given in Appendix A. Note that more complicated polyhedral shapes can be described as union of these convex primitives.

4.2 Solving The Single Link Problem

To solve the motion planning problem for a manipulator arm, the sequential search strategy will first plan a collision-free path for the first link of the manipulator arm, and then the second link, and so on. The key step therefore is to solve the motion

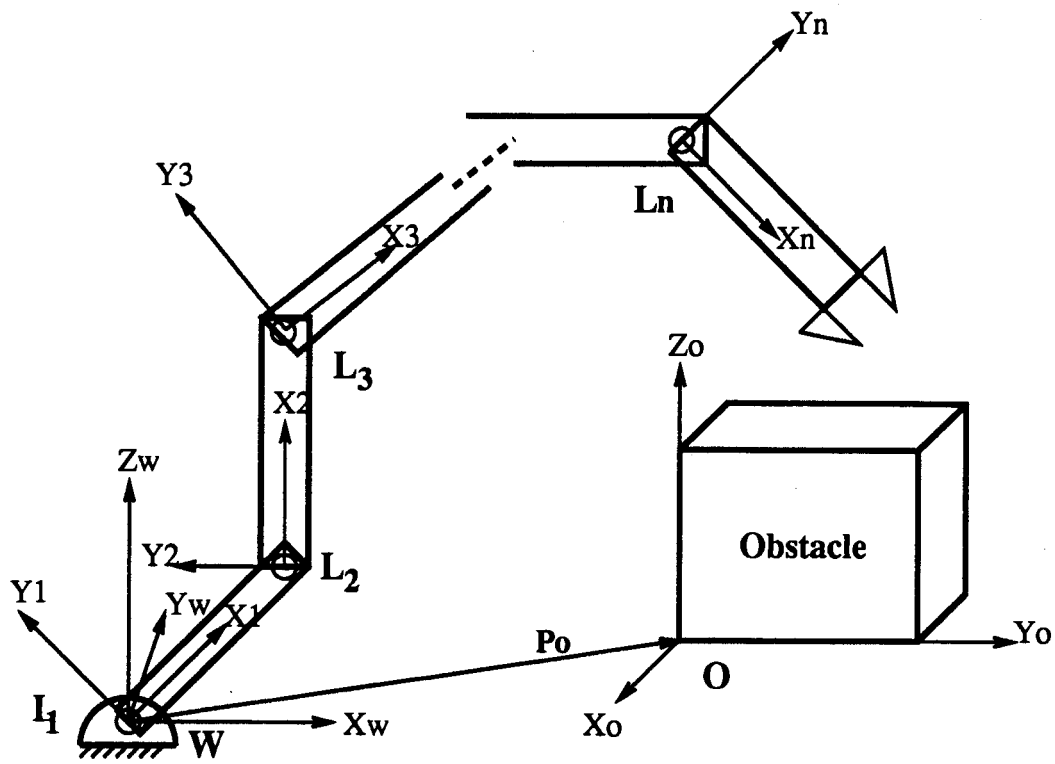


Figure 4.1: Robot coordinate systems

planning problem for a single link. Now assume that the motion of the previous $i-1$ links has already been planned and is specified as a function of a discretized parameter $t_i = [0, 1]$, i.e., $\theta_{i-1}(\frac{k}{M})$ is known for $k = 0, 1, \cdots, M-1$, and M is the total number of samples of the whole collision-free path given by user. Note that this process completely determines the path of $\mathbf{r_i}$, one end of link i which is connected with link $i-1$, and as it moves along this path, the problem now is to plan a collision-free path for link i (or to adjust the values of θ_i) to avoid the obstacles in its way. We call this the single link problem (please see Figure 3.1). To solve the single link problem, there are four major steps, as mentioned in chapter 3. Figure 4.2 shows the steps for the single link problem. The following sections give a detailed discussion of each step.

Having planned the path for link i, we parameterize the path for link i with a new discrete parameter, t_{i+1}. The motion for link $i+1$ now can be planned in $t_{i+1} \times \theta_{i+1}$ space. This process can be iterated until the last link. Clearly, using the sequential search strategy, we always solve the single link problem in a simple two-dimensional $t \times \theta$ space until we find a path or fail to find it.

4.2.1 Calculating Forbidden Regions

The first step is to compute the joint angle values for a given link which result in collision with the obstacles, as one end of the link moves along a discretized path. At each discrete point of this path, the set of angle values that result in a collision with the obstacles are called forbidden angles. Since we assume that the obstacles and links are polyhedra, the fundamental and geometric problem is to determine the set of angles at which a rotating polyhedron intersects another static polyhedron. For

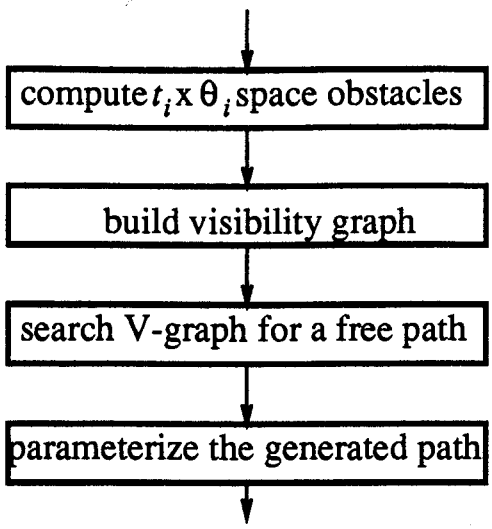


Figure 4.2: The diagram shows the steps in solving the single link motion planning problem.

the case of convex polyhedra, these angle values are interval sets, called forbidden ranges, bounded by values for which the two polyhedra are in contact. A forbidden range is therefore, an interval, composed of two angle values, *start* and *end*. *start* indicates that a change in rotation angle θ will move further into the contact, and *end* away from the contact. There are two steps in computing these contact values. The first step is to compute all possible potential contact angles, and the second step is to see which ones satisfy certain constraints.

4.2.2 Potential Contact Angle

Lozano-Pérez [9] defined three basic contacts, called Type A: *link-vertex with obstacle-face*, Type B: *link-face with obstacle-vertex* and Type C: *link-edge with obstacle-edge.*

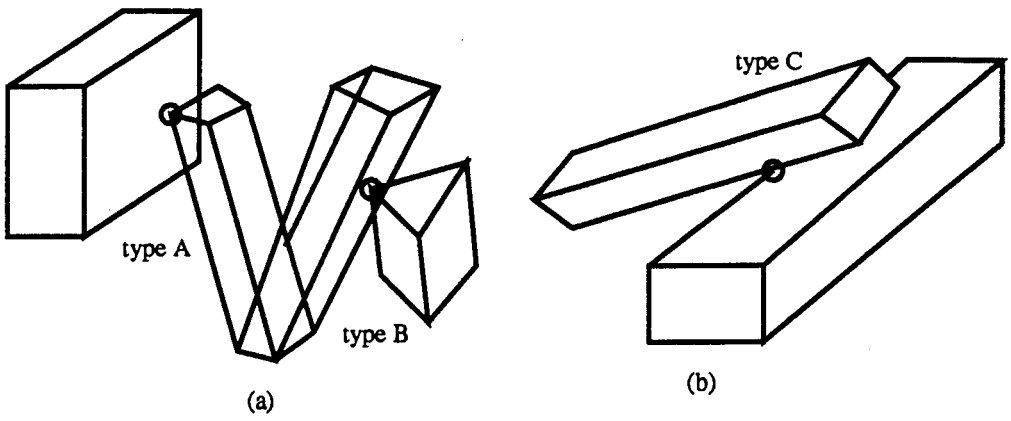


Figure 4.3: Three types of contacts. (a) type A or type B; (b) type C.

The definitions of three types of contacts are shown in Figure 4.3. Here the detailed derivation is ignored (see Appendix B for detailed derivation), which essentially follows [9]. As a summary, the complete algorithms are shown in Figures 4.4, 4.5 and 4.6.

4.2.3 Constraints

A potential contact does not necessarily imply a real contact. It must satisfy certain constraints. Based on Lozano-Pérez's definition (see Figure 4.3), there are three constraints in general: (1) *in-surface* constraint, (2) *orientation* constraint and (3) *reachability* constraints. The precise mechanisms for computing these constraints differ for type A, B and type C contacts. Lozano-Pérez presented a detailed derivation of the constraint detections for a two-dimensional case (see [9]). For the three-dimensional case, we give a more detailed discussion in Appendix C.

We make use of these constraints to determine whether a potential contact angle is a *real* contact angle. These computations are used at each discrete point of

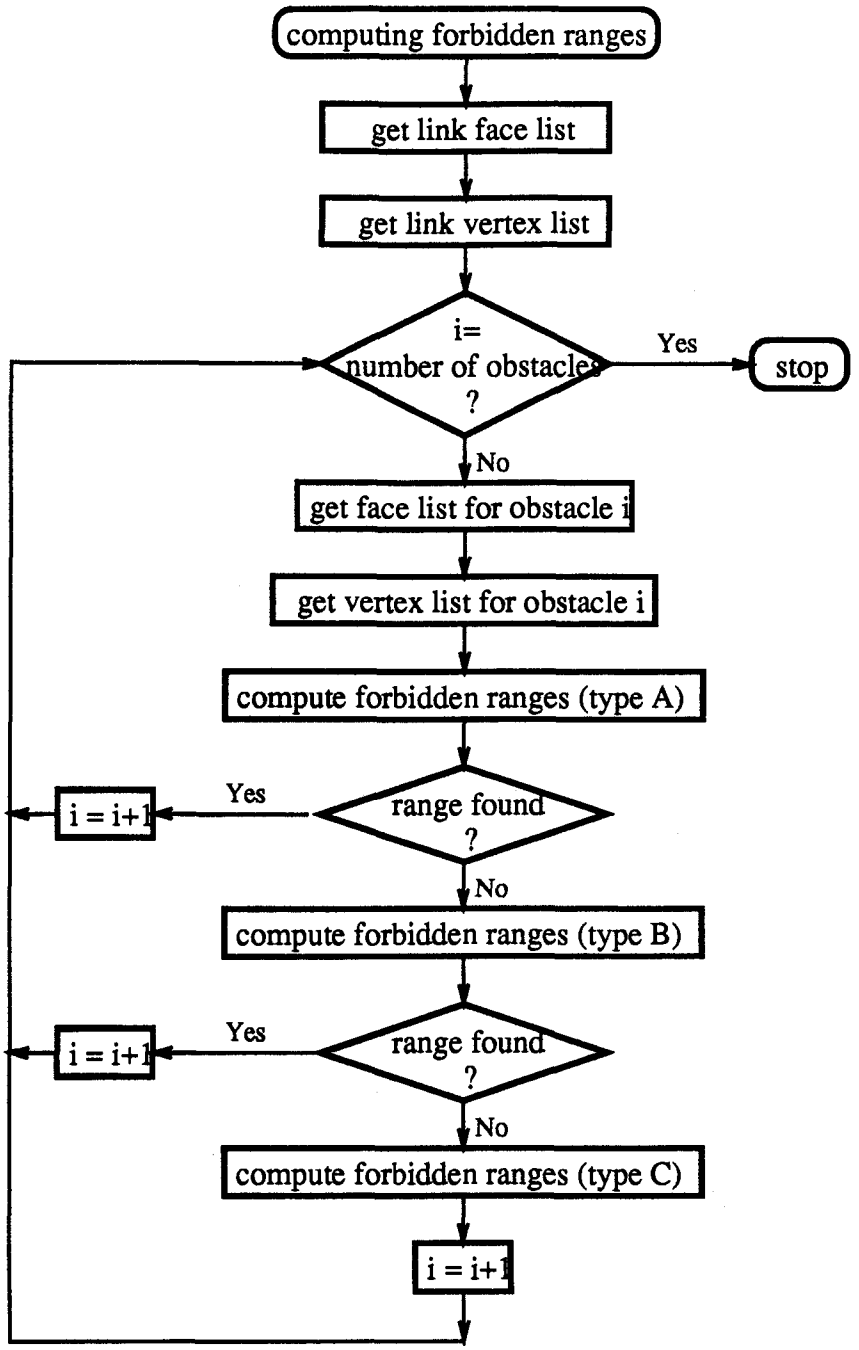


Figure 4.4: The diagram shows the procedure of computing forbidden ranges.

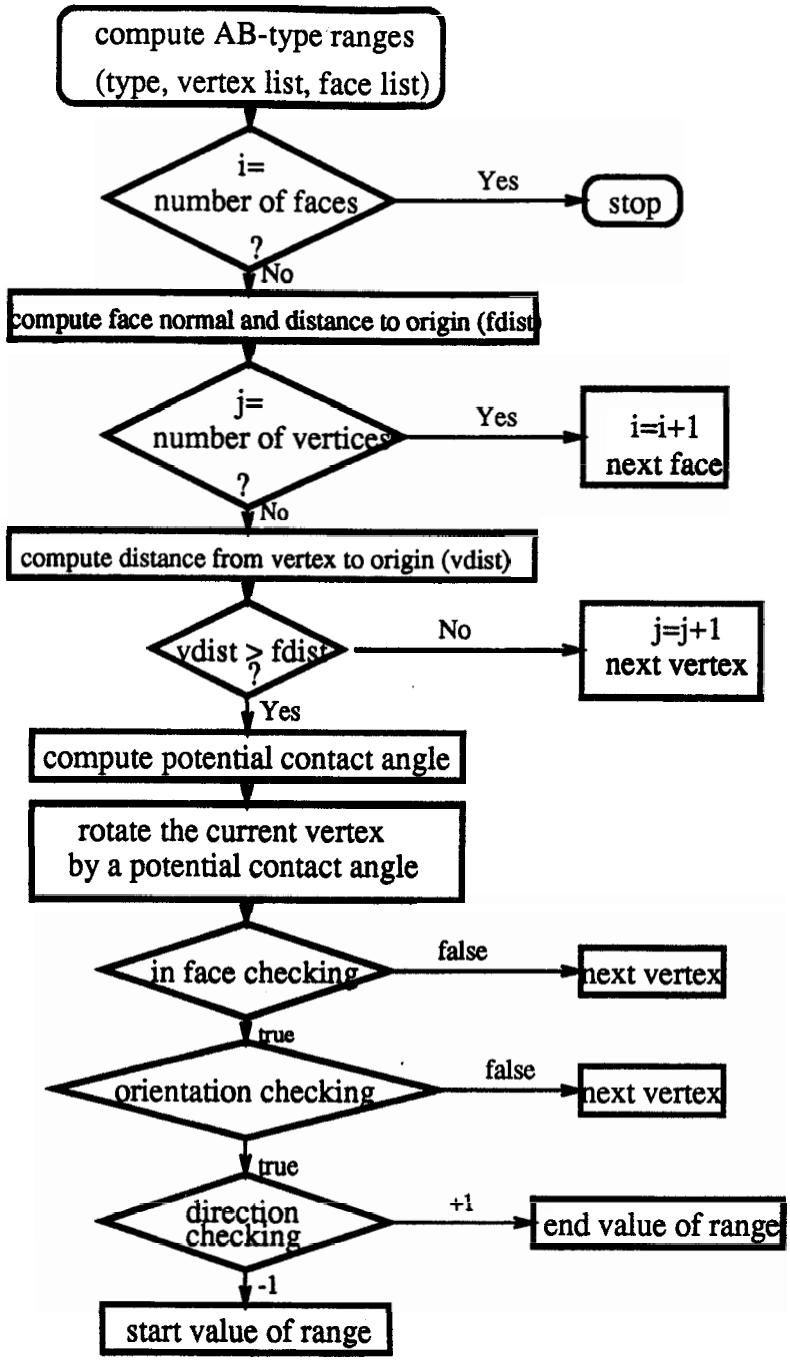


Figure 4.5: The diagram shows how to compute forbidden ranges for type A and B contact.

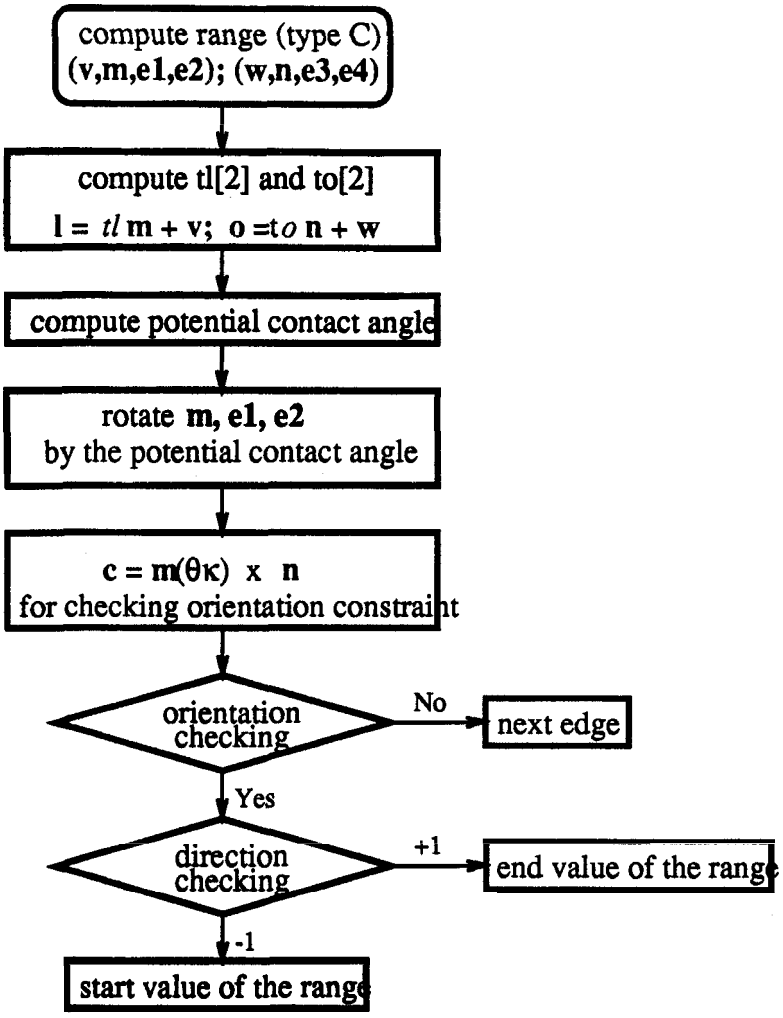


Figure 4.6: The diagram shows the procedure of computing type C contact. Please see Figure C.3 for explanation of symbols used in this diagram.

the path to compute the forbidden ranges. These forbidden ranges, in $t_i \times \theta_i$ space, appear as regions. Note that for each discrete parameter value a number of forbidden ranges may exist. The set of ranges at a given discrete point is called a forbidden slice.

4.3 Building Forbidden Regions in $t_i \times \theta_i$ Space

Having obtained forbidden ranges in $t_i \times \theta_i$ space, the next step is to group those ranges into regions in $t_i \times \theta_i$ space. A *forbidden region* is made up of linear ranges from a set of adjacent slices such that all the ranges in the region overlap. These regions are then approximated by simple polygons, however, these polygons may have too many vertices. Hence we further approximate the polygons to reduce the number of vertices of a *forbidden region* to an acceptable number.

4.3.1 Grouping Forbidden Ranges into Regions

The regions are formed by iterating over the forbidden slices from left to right, that is, from *time-parameter* $t = 0$ to $t = 1$. The basic idea is to group contiguous and overlapping forbidden ranges in one region. We have used a very conservative scheme as follows: each forbidden range in the first slice starts a new region (obstacle). We keep the current range into a *bounding range* and update it every time after a new range is added to the region. As each slice is considered, the range overlapping the *bounding range* of a given region is added to that region. A region is terminated when no range in a slice overlaps the bounding range of this region. The grouping

procedure is outlined in Figure 4.7 . Figure 4.8 shows a grouping example in our implementation. Note that this simple procedure will always yield regions that increase monotonically in parameter t, albeit the regions may overlap.

4.3.2 Polygonal Approximation

Having grouped the overlapping ranges in a region, our next step is to further simplify the representation of a region by approximating each region with a polygon. Note that by tracing the end points of each range in the region, we immediately have a simple polygon. However, there are too many vertices on this polygon, two for each range. This would make the subsequent V-graph computations computationally intensive. We would like to have fewer vertices in the approximating polygon. We have implemented two approaches to it. The first approach is a straightforward intuitive approach, and the second is a more systematic and better approach. However, we discuss both approaches here.

The first approach is based on obtaining the convex hull, i.e., removing the concave vertices from the original polygon until all such vertices are removed and a convex polygon is obtained. This is easily achieved as shown in Figure 4.9. The process is illustrated on the upper boundary of the original polygon. Tracing the boundary from left to right, we compute the internal angle at the vertex. If the internal angle is greater than 180 degrees, the vertex is removed from the polygon, and a new edge connecting the previous and the next vertex is added. This process is repeated iteratively until there is no concave vertex left. A similar process is carried out for the lower boundary of the polygon.

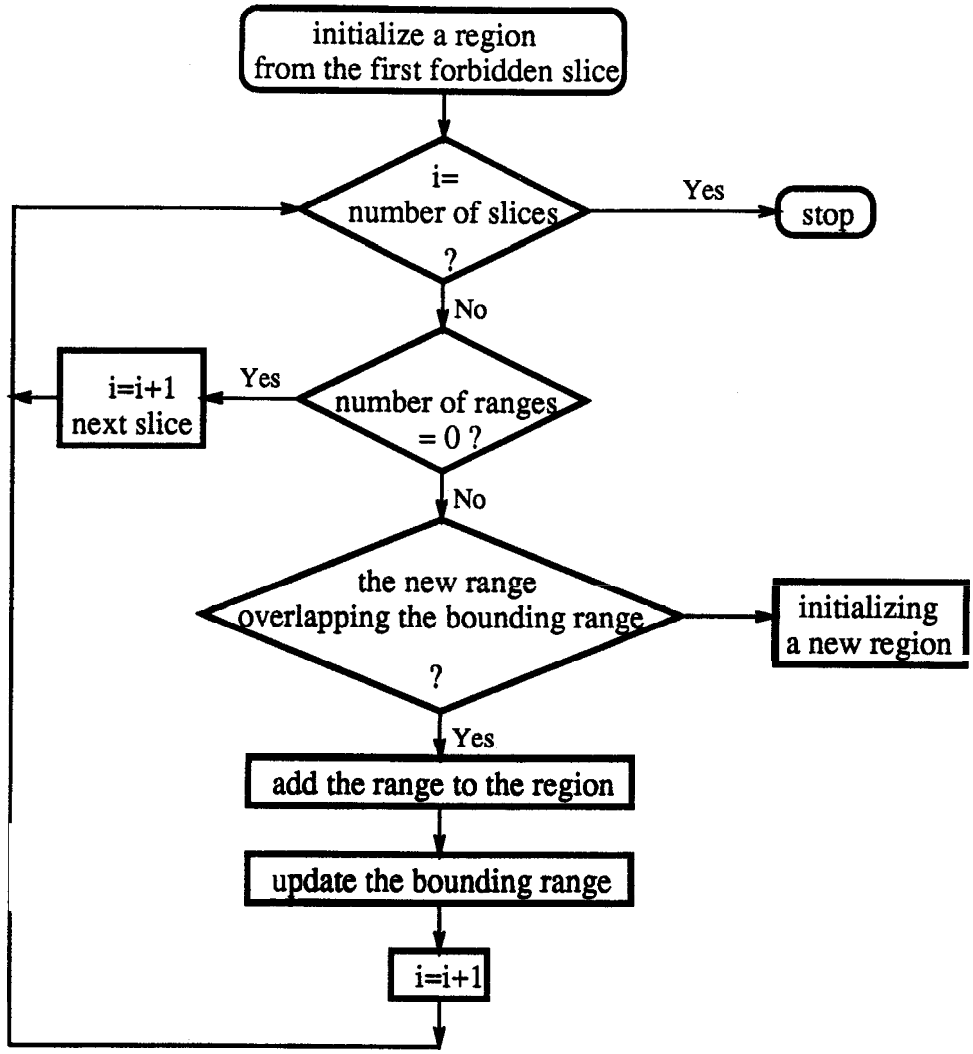


Figure 4.7: Diagram shows how to group contiguous and overlapping ranges into forbidden regions.

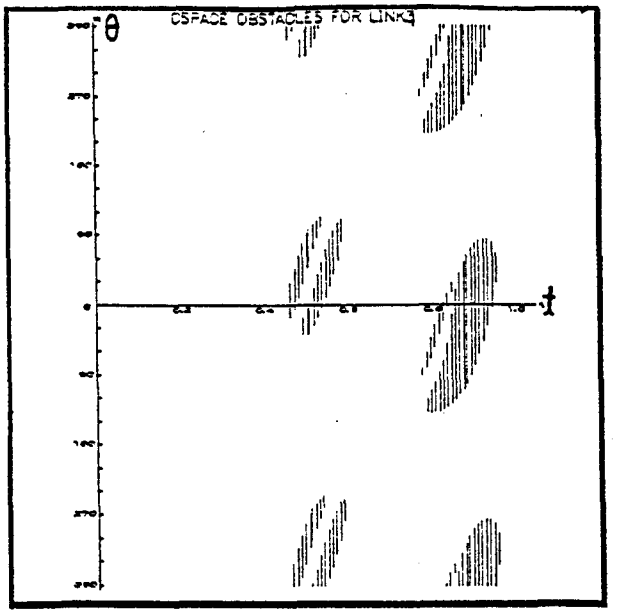


Figure 4.8: An example of grouping forbidden slices into forbidden regions in our implementation.

There may still be too many vertices on the boundary of the polygon. To further reduce the number of vertices, we can prune short edges of the polygon. Figure 4.10 shows the pruning process. This process stops when the number of vertices of the polygon meets the desired number.

It is easily seen that the convex polygon approach does reduce the number of vertices, however, it may waste much of the free-space, which is treated as obstacles and a collision-free path may not exist. For example, the initial, or final, or both positions, may be trapped in the forbidden region (Figure 4.11 shows an example for this undesired case).

A variation of the above approach is to reduce the number of vertices by merging adjacent edges if the difference of their slope values is small. Note that with this approach the resulting polygon may not be convex. The process is iterated until

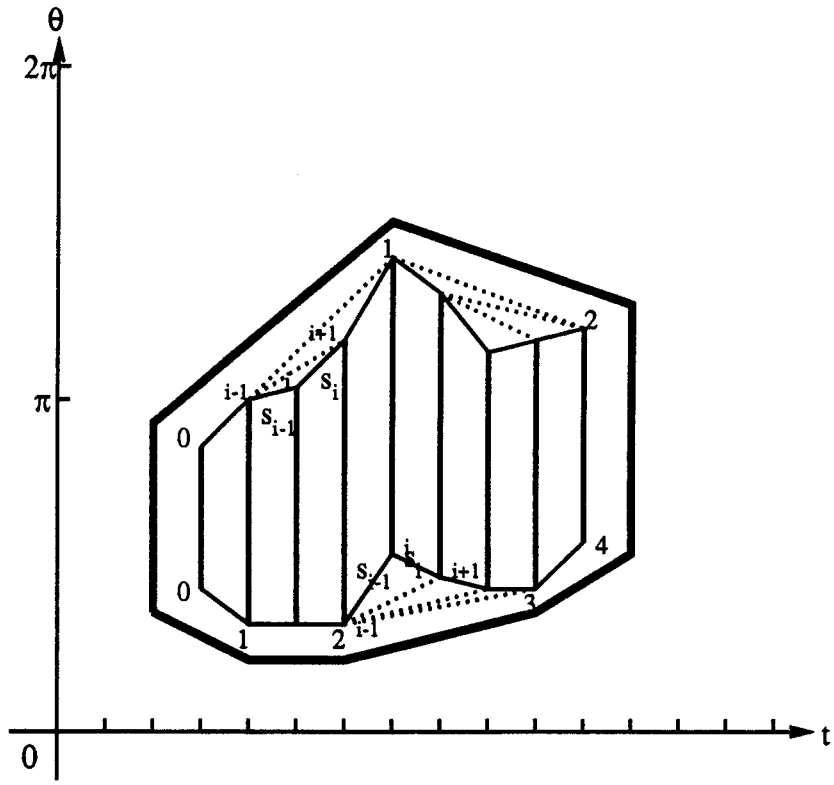


Figure 4.9: Convex polygon approach works recursively: a vertex is removed at a time according to the updated internal angle value.

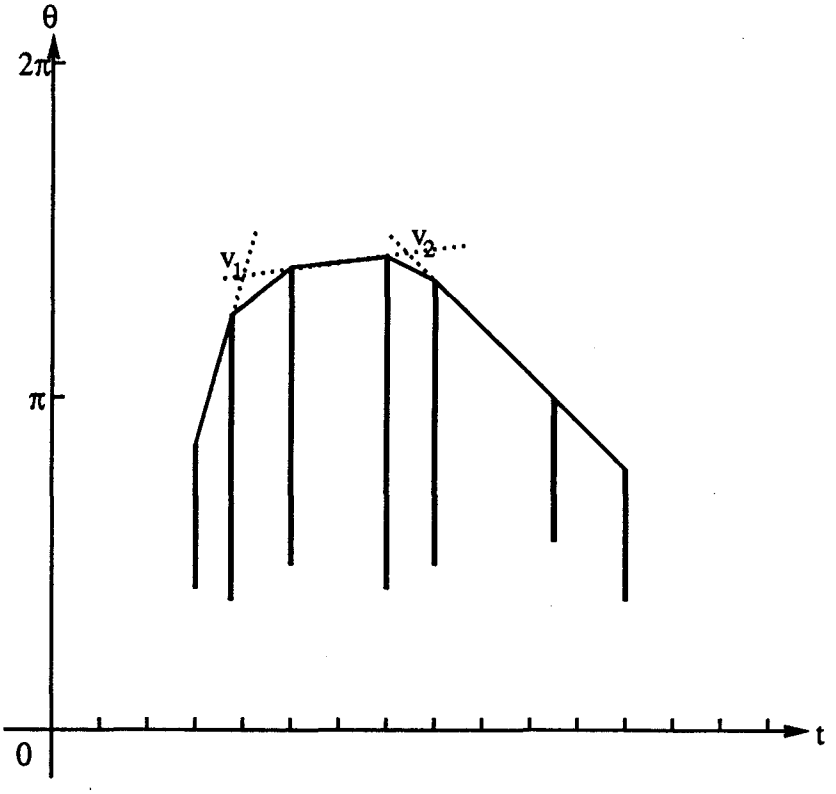


Figure 4.10: Two short edges have been removed by the pruning process, and two new vertices, $\mathbf{v_1}$ and $\mathbf{v_2}$ are added. Note that the number of edges in the resulting polygon is less than that in the original polygon.

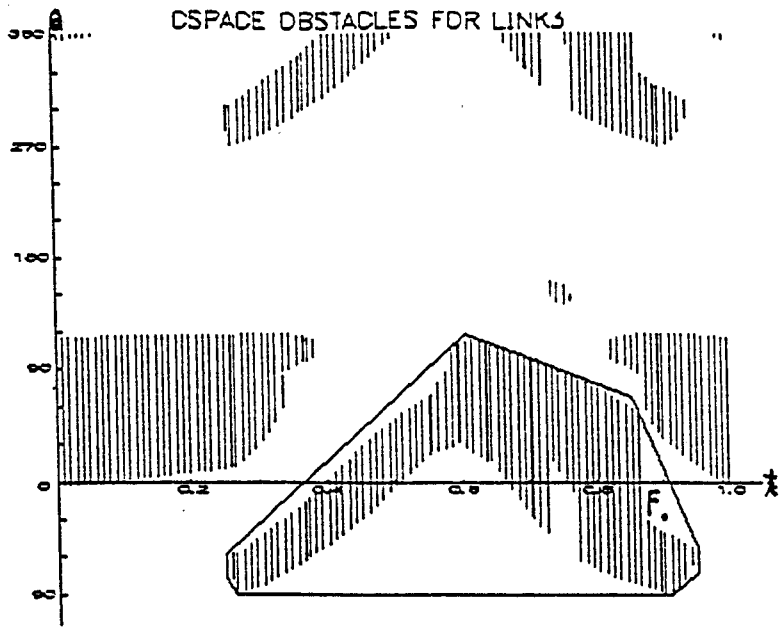


Figure 4.11: The advantage: total number of vertices is eight; disadvantage: final position is trapped in the approximated region, but not in the exact forbidden area.

the number of vertices of the polygon is below a pre-specified limit. Figure 4.12 gives an example of this approach, and Figure 4.13 shows a perfect result obtained by this approach, which is much better than that generated by the convex polygon approach. However, this approach may also, in some cases, waste much of the free-space. The fundamental problem is that we are arbitrarily specifying number of vertices rather than specifying a limit on how much free-space we can afford to waste. In the next section, we describe an approach based on the latter strategy.

4.3.3 Piecewise Linear Function Approach

This approach is based on polygonal approximation, a sub-area in computational geometry that deals with the problem of approximating a piecewise linear curve by another coarser one. In general, there are two types of optimization problems

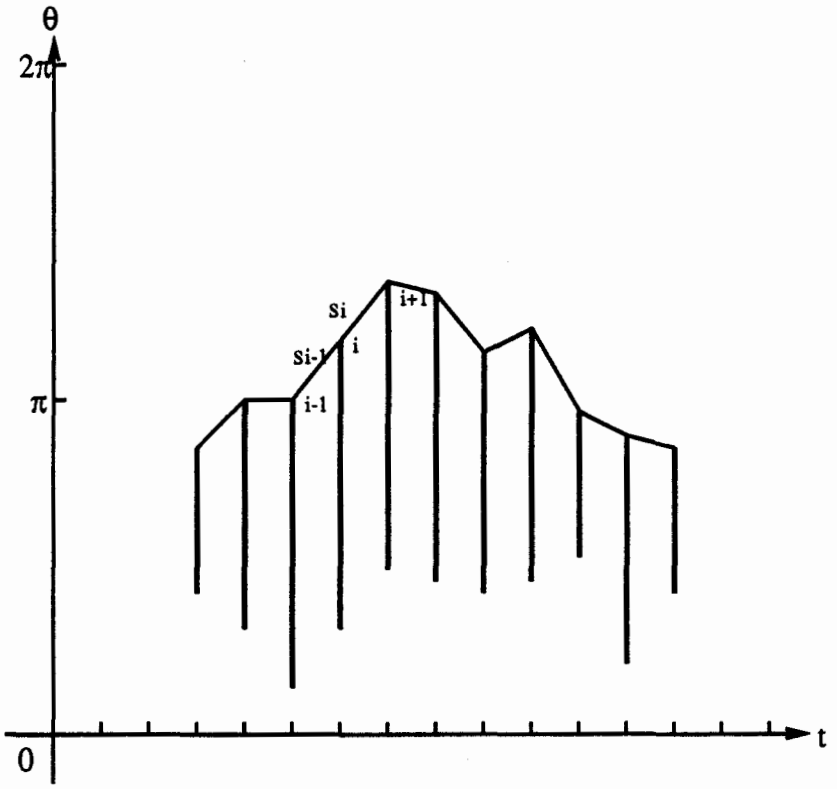


Figure 4.12: Nonconvex polygon approach: vertex i will be removed since the difference between slope s_{i-1} and s_i has a small value.

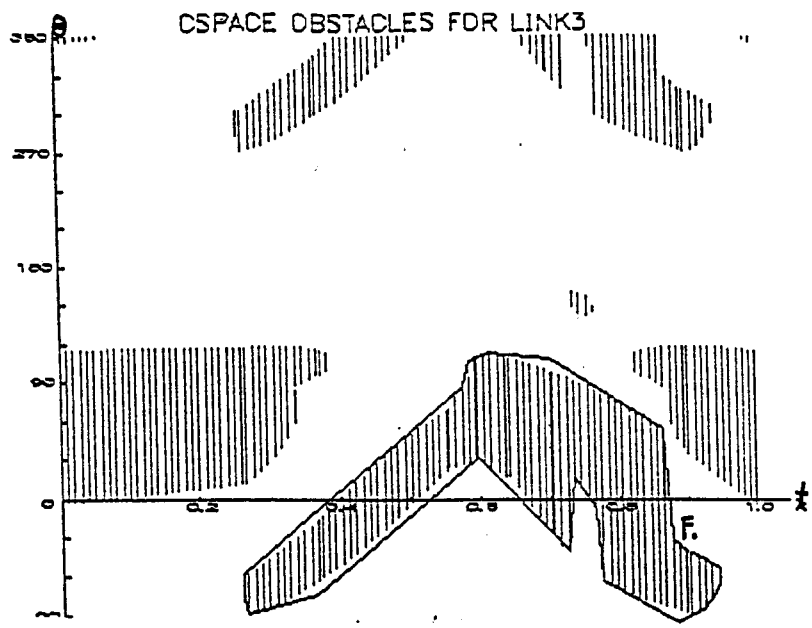


Figure 4.13: There is not much free space lost, but the number of vertices is not smaller than in the convex polygon approach.

related to polygonal approximation: (1) minimizing the number of vertices of an approximate curve with the error within a given bound; (2) minimizing the error of an approximate curve consisting of a given number of vertices [36]. For our purpose, we make use of type (1) to coarsely approximate forbidden regions by polygons with a smaller number of vertices within a reasonable error ϵ.

The details of the algorithm are given in [36]. We briefly discuss the basic idea behind the algorithm. Suppose that the lower boundary of a forbidden region is considered. Based on the polygonal line with seven vertices, $p_1p_2\cdots p_7$, and a given error ϵ, polygon $\mathbf{P}(\epsilon)$ is formed by sliding this polygonal line downwards by ϵ, shown in Figure 4.14. First of all, edge e_1 is selected to be a *window* (the definition is given in [36]). Then we detect whether an edge can be seen from the *window*. If the last edge, edge e_7, can be seen from the *window*, edge e_1, the approximate polygonal line can be a straight line connecting edge e_1 and e_7, that is, a polygonal line with seven

vertices becomes a straight line with only two vertices. But in practice, the case is not that simple. In the example of Figure 4.14, edge e_4 is not visible from the *window*. Therefore, the *window* (e_1 at this moment) is updated. After updating, the *window* is w_2 (see Figure 4.14), and then we continue the visibility detection from the updated *window* until the last edge becomes visible from the *window*. Finally, a new polygonal line with four vertices, $v_1v_2v_3v_4$, is generated by this approach. In our implementation, we use discretized windows and edges.

4.3.4 Growing Obstacles in $t_i \times \theta_i$ Space

Since we are using V-graph based approaches to search for a path, the path consists of several *nodes* which are vertices of $t_i \times \theta_i$ space obstacles. From this point of view, when the robot manipulator moves to any *node* along the free path it means that the robot has touched an obstacle in its workspace. In order to overcome this problem, we *grow* them. For the upper boundary of the region, the vertical coordinate value of each vertex is *grown* upwards by a given resolution r, and for the lower boundary, it is *grown* downwards by r (usually we set $r = 1°$). For two vertical boundaries, the left one moves one interval to left side and the right one interval to the right side. In Figure 4.9, the thick solid line shows the grown forbidden region based on this method.

4.3.5 Joint Limits Treated As $t_i \times \theta_i$ Space Obstacles

In practice, a revolute joint can not rotate a full circle due to joint limits. For each joint, the limit values form a forbidden range in which the link can not move. For

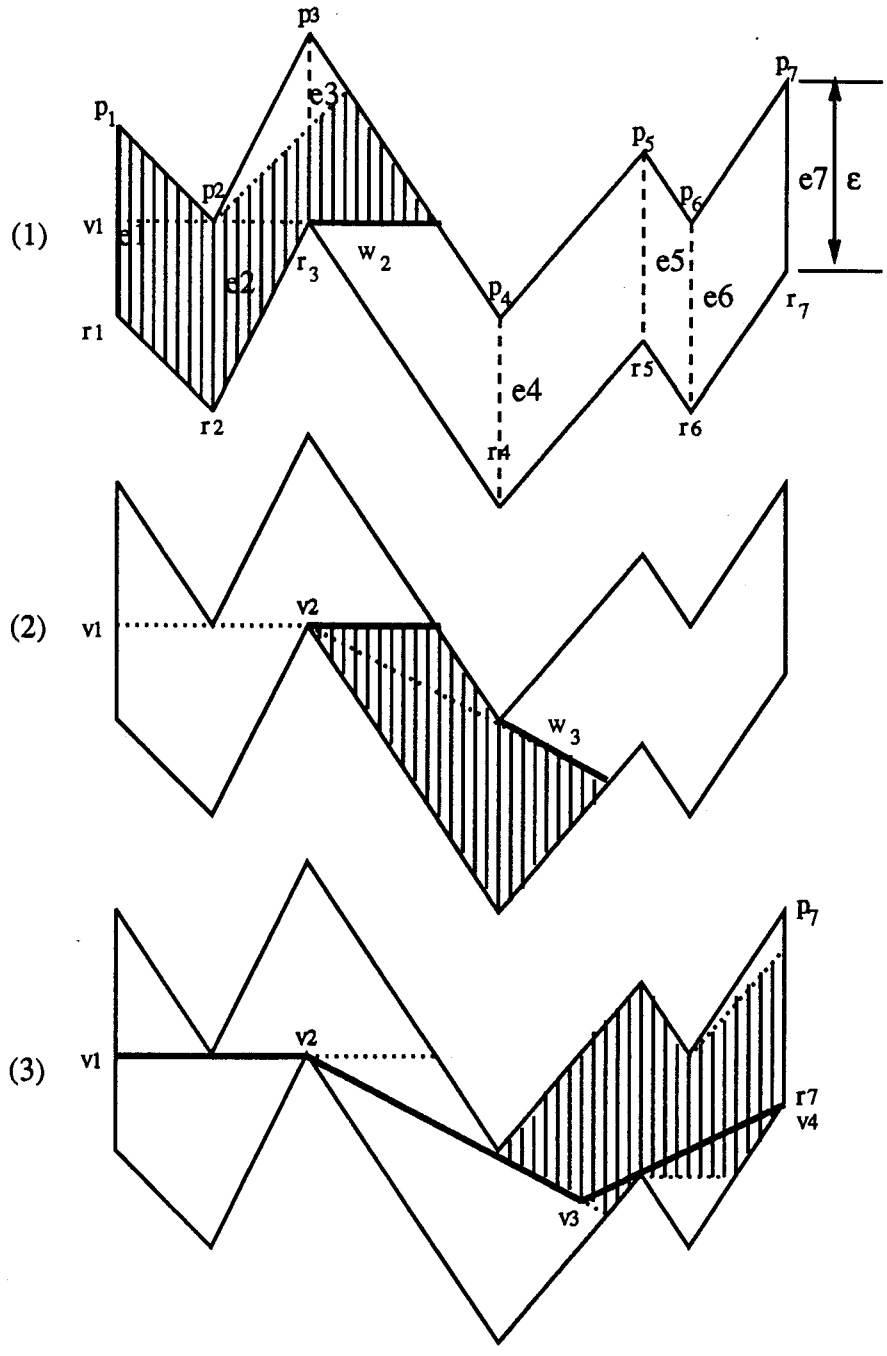


Figure 4.14: Reducing the number of vertices with a piecewise linear function approach. After this procedure the number of vertices becomes four.

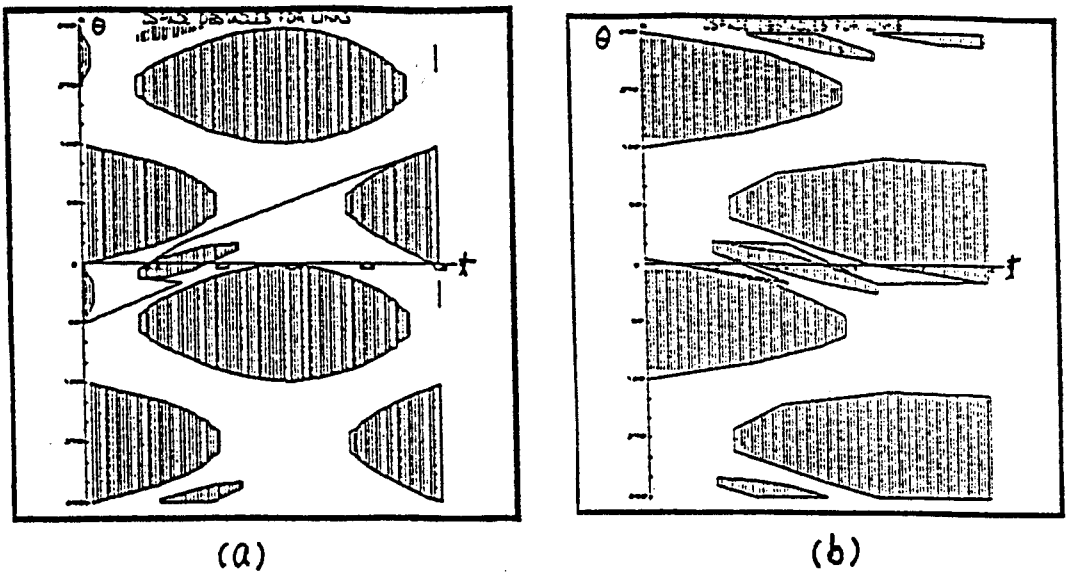


Figure 4.15: A result of the piecewise linear function approach (a) compared to that of the convex polygon approach (b).

link i, a joint angle limit is treated as a rectangular obstacle in its $t_i \times \theta_i$ space. The horizontal extent of the rectangle spans the whole parameter interval [0, 1].

4.4 Searching For A Collision-Free Path

Having obtained the polygonal approximations of the obstacles in $t_i \times \theta_i$ space, the next step is to find a collision-free path from the initial position (t_i^s, θ_i^s) to the final position (t_i^g, θ_i^g) in $t_i \times \theta_i$ space. We have used V-graph based techniques [12]. The V-graph is formally defined as $\mathbf{VG} = (\mathbf{N}, \mathbf{E})$, where $\mathbf{N} = (\mathbf{I}, N_1, N_2, \cdots, N_m, \mathbf{F})$, is the set of nodes of the V-graph, **I**, the source node corresponding to the initial position of the robot link, **F**, the target node corresponding to the final position of the robot link, and $\mathbf{N_i}$, the vertices of the polygonal obstacles for $i = 1, 2, \cdots, m$,

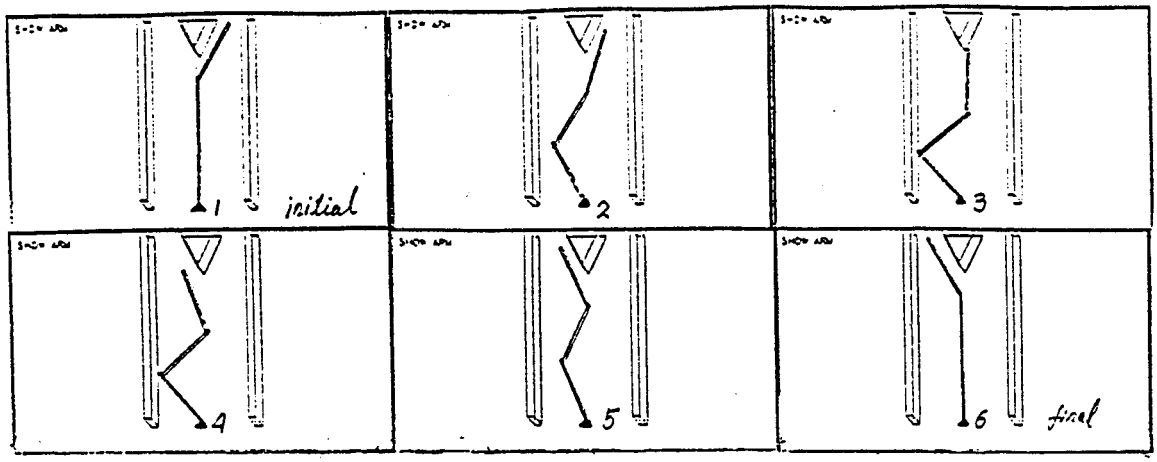
m is the total number of vertices in $t_i \times \theta_i$ space. Any two nodes that can "see" each other, i.e., the line segment joining the two nodes does not intersect any of the obstacles, are connected by an edge with weight equal to the length of the edge. **E** is the set of these edges.

Efficient techniques for building the V-graph are well documented in the literature and we omit details here [12].

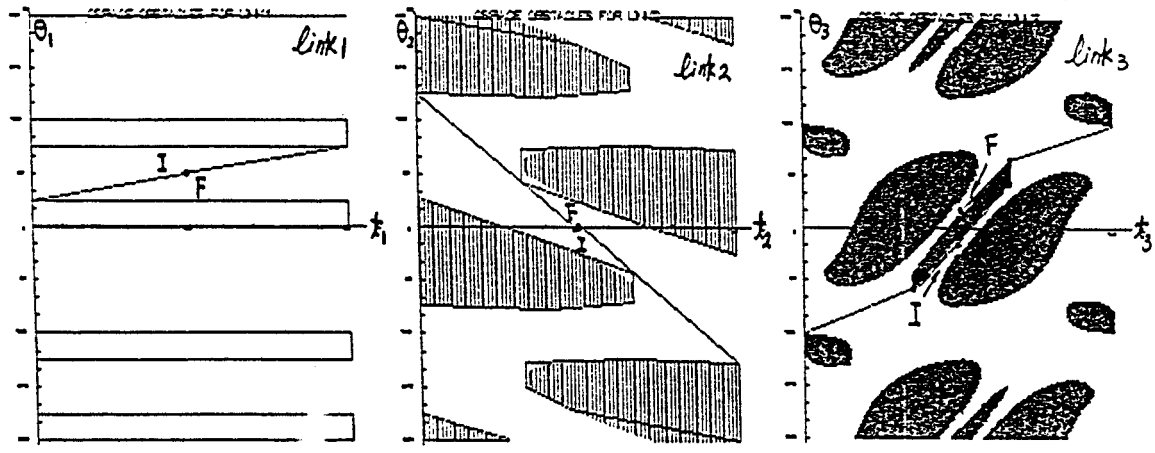
A main reason for choosing the V-graph approach is that in a two-dimensional space, a necessary condition for a *shortest* collision-free path length is that it is composed of straight line segments connecting a subset of the vertices of the polygonal obstacles. The shortest collision-free path from the initial position to the final position is thus given by the minimum weight path from **I** to **F**. We then use breadth first search to find the shortest collision-free path in $t_i \times \theta_i$ space [44].

4.5 Extension Segments And Backup Movements

In many situations (we will illustrate them in chapter 5), it may be crucial for a given link to back up away from the start or the goal position and then come back to it, in order for the next link to maneuver. We call such motions backup motions and they may be crucial in finding a collision-free path. Consider the example in Figure 4.16. Suppose that the robot links are not allowed to cross over. With this constraint, the only way for the arm to reach the final configuration is to allow the first two links to move beyond their final position so that link 3 can move and then allow the first two links to move back to the final position again.



(a)



(b)

Figure 4.16: A three-link example illustrates the backup motions for link 1 and link 2 beyond their start to goal intervals. (a) shows the motion of the arm. It is clear that the backup movements beyond the start to goal interval happen on the first two links. (b) shows the $t_i \times \theta_i$ spaces for each link. For the first two links, the initial and the final are located at the same position in their $t_i \times \theta_i$ space. When link 3 moves forward along the generated path from the initial node **I** in $t_3 \times \theta_3$ space, the first two links will move beyond their final positions and then move back to the final positions again. Note that $\mathbf{I} = [t_i, \theta_i]$ and $\mathbf{F} = [t_f, \theta_f]$

A straightforward modification, as shown in Figure 4.16, leads to such a mechanism. In the initial parameterization for link 1, the algorithm considers the sub-interval from the initial value θ_1^s to the goal value θ_1^g, i.e., θ_1^s, θ_1^g for the first link. In general, this sub-interval will be a subset of a whole free interval. We parameterize the whole interval (instead of only the sub-interval from initial position to final position) and then plan the path for the second link. Note that the beginning and the end of motion of link 1 now do not correspond to $t = 0$ and $t = 1$, but to some intermediate values, $t = t_1^s$ and $t = t_1^g$. The beginning and goal points for link 2 are also not at $t = 0$ and $t = 1$, but at $t = t_2^s$ and $t = t_2^g$.

For the first link there is a natural mechanism for the link to move beyond its final (or initial) configuration – by considering the whole free-interval in which the initial and the final configuration values for the first joint lie. There seems to be no natural way of extending such "beyond the final (or initial) configuration motion" for subsequent links.

The mechanism of backup movements beyond the beginning to goal interval is realized by searching two extension segments of a collision-free path. Clearly, these two extension segments have to be in the free $t_i \times \theta_i$ space for each of the robot links. In our approach, we first select two nodes, one node with parameter value $t = 0$ and denoted by $\mathbf{F}_b'$, and the other node with parameter value $t = 1$, denoted by $\mathbf{F}_f'$. We then search for a collision-free path from $\mathbf{F}_b'$ to $\mathbf{I}$, called the backward extension segment; and a collision-free path from $\mathbf{F}$ to $\mathbf{F}_f'$, called the forward extension segment. To perform an effective backup movement, so that the robot link can move beyond the initial to final interval within a big enough range of the joint angle, $\mathbf{F}_b'$ (or $\mathbf{F}_f'$) is selected such that the difference between θ_b' and θ_i (or

θ_f' and θ_f) is as large as possible. The same graph search technique is used to search for the extension segments as for the main path from **I** to **F**. See Figure 4.18 as an example. Segment $\mathbf{F}_b'\mathbf{I}$ is the backward extension segment and $\mathbf{F}\mathbf{F}_f'$ the forward extension segment.

Thus the path for link 2 can go beyond the $[t_2^s, t_2^g]$ interval. Similarly, the path for link 3 also can go beyond the $[t_3^s, t_3^g]$ interval. Such motions correspond to backup movements beyond the start to goal interval. Hence the first two links can now back up away from their final position and then move back to their final position. With this modification, this three-link example is easily solved. The motion is shown in (a) of Figure 4.16.

4.6 Parameterizing A Collision-Free Path

Having planned a collision-free path, the next step is to reparameterize this path. The new parameters will form the horizontal axis of the two-dimensional $t \times \theta$ space for the next robot link. For example, suppose we have planned a collision-free path and discretized it for link $i-1$. The discretized path is then reparameterized with the parameter t_i. Figure 4.17 shows this procedure.

The method adopted in our motion planner is called equal-length parameterization, and uses discrete normalized arclength along the path as the new parameter. The collision-free path for every robot link is sampled in the same number of samples, M. Suppose that the motion of the previous $i-1$ links has already been planned and is specified as a function of a discretized parameter $t_i = [0, 1]$, i.e., $\theta_{i-1}(\frac{k}{M})$ is known for $k = 0, 1, \cdots, M-1$. Further, suppose that the length of the

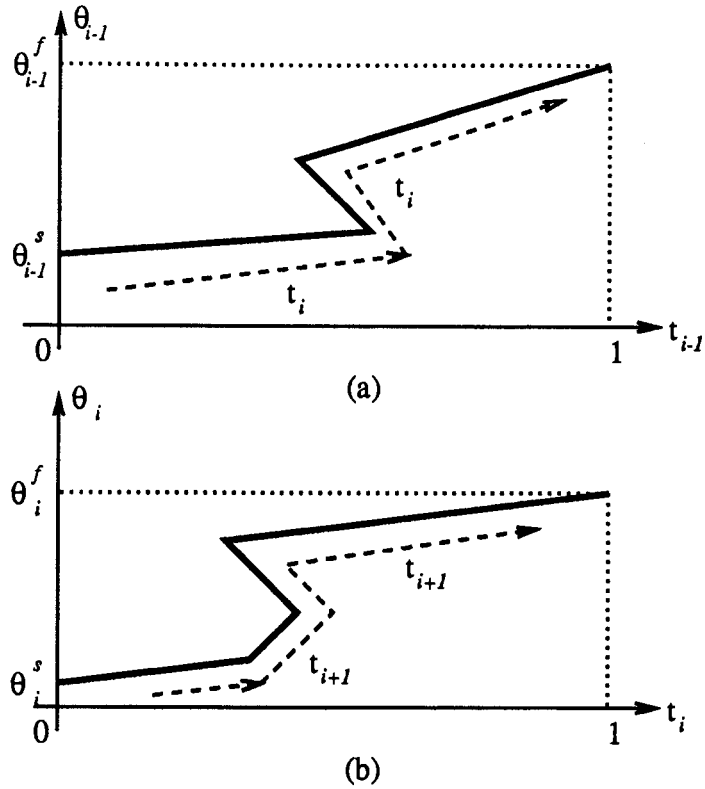


Figure 4.17: The path for link $i-1$ is shown as a piecewise linear path in its $t_{i-1} \times \theta_{i-1}$ space. This path is parameterized (along its length) with the parameter t_i. The parameter t_i is then drawn as the horizontal axis in (b) as if the path in (a) has been "straightened out". The path for the next link (link i) is then computed as a piecewise linear path in the $t_i \times \theta_i$ space for link i.

path equals L. The path is then divided into M samples of equal length, the length of each sample being $\Delta T_{i+1} = \frac{L}{M}$. The joint angle corresponding to k^{th} sampled point is then easily determined by linear interpolation and represents $\theta_i(\frac{k}{M})$. Note that for each discrete value $\frac{k}{M}, k = 0, 1, \cdots, M-1$, we can calculate the corresponding $\mathbf{q}_i$. Figure 4.18 shows this method.

A main drawback of constant number of samples is that a "jump" may happen when the robot arm moves along the vertical segments (see Figure 4.18). To overcome this, we may choose a larger M, but this results in spending a longer time to build the $t_i \times \theta_i$ space. Therefore, a more adaptive sampling scheme that guarantees a minimum resolution for every joint angle is more desirable.

4.7 Backtracking Mechanism

The main intention of backtracking search mechanism is to make the motion planner more powerful so that it becomes closer to a complete planner. Although the sequential planner described above does solve difficult problems such as the one illustrated in the previous section, nevertheless, there are situations where (say, while planning for link i) no collision-free path is found in the $t_i \times \theta_i$ space. In such situations, the planner must backtrack and re-plan the path for the previous links. i.e., a backtracking search mechanism is required. The planner may go back to the previous link, link $i-1$, and search for another free path, Q'_{i-1}, in the $t_{i-1} \times \theta_{i-1}$ space (which has already been built previously). Along the new path, Q'_{i-1}, the planner should reconstruct the $t_i \times \theta_i$ space and search for a collision-free path Q'_i for link i again. This is repeated till either (i) a path is found for link i, or (ii) all

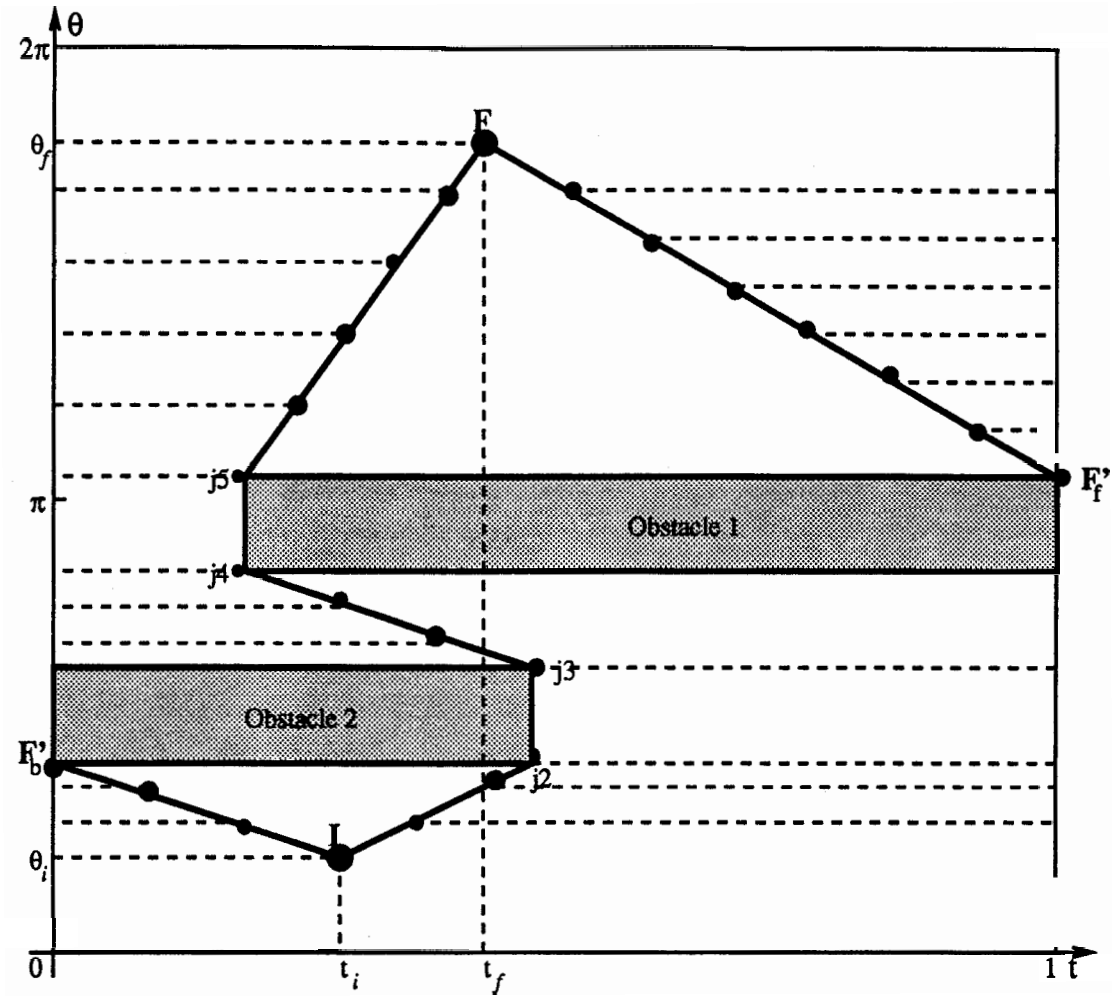


Figure 4.18: The path is parameterized by "Equal Length" based approach.

the paths are exhausted in $t_{i-1} \times \theta_{i-1}$ space. In case (i), the planner goes on to the next link, link $i+1$. In case (ii), the planner has exhausted all the paths for link $i-1$ and still does not find a path for link i. Therefore, the planner backtracks another level, i.e., to link $i-2$, and searches for another path, Q'_{i-2}, for link $i-2$ in $t_{i-2} \times \theta_{i-2}$ space (which has been built previously). Along this new path Q'_{i-2}, the planner builds and searches the $t_{i-1} \times \theta_{i-1}$ space for a new path Q'_{i-1} for link $i-1$. Along this path, the planner will try to find a path for link i. In general, the planner may backtrack more than just the previous link, say k previous links and search again. The parameter k is the backtracking level, and it is an adjustable parameter. Indeed, with greater k, the planner will be more complete but slower as is shown in the worst case complexity analysis of the planner in section 4.8.

In the following paragraphs, we present a precise backtracking mechanism based on the above approach.

Suppose that while planning for link i, there is no collision-free path found in the corresponding $t_i \times \theta_i$ space, a main question is how to modify or re-plan a new path for link $i-1$, and how many such paths should be tried during backtracking. Since our methodology to find a path in $t_i \times \theta_i$ is V-graph based, our approach is to first determine the node with the maximum parameter value that is reachable from the start node, i.e., the maximum parameter value of t_i at which the "block" occurs for link i in $t_i \times \theta_i$ space. A portion of the path Q_{i-1} in $t_{i-1} \times \theta_{i-1}$ space, that corresponds to this maximum value, is then deleted. Since our representation is V-graph based, we remove that edge segment from the V-graph that is on the

path and corresponds to the blocked parameter value [1]. This pruned V-graph is then searched again for a new path Q'_{i-1}. Along this new path for link $i-1$, the single-link problem is solved again for link i, i.e., forbidden regions are constructed and approximated by polygons, and a V-graph search is carried out for a path, Q'_i in $t_i \times \theta_i$ space.

The structure of the new planner with the backtracking mechanism is shown in Figure 3.2 in chapter 3. Several examples have been implemented and the experimental results are shown in chapter 5.

To clarify our discussion, we would like to illustrate the backtracking mechanism with an example shown in Figure 5.5 in chapter 5. We redraw the $t_2 \times \theta_2$ space for link 2 in Figure 4.19, and it shows the V-graph for link 2. The first path chosen by the planner (the shortest path in the V-graph) is segment **IF**. Along this path **IF**, there is no free path for link 3 – the initial position is trapped in a forbidden region as shown in the $t_3 \times \theta_3$ space for link 3 in Figure 4.20. Physically, it implies that link 3 can not move out of its initial position if links 1 and 2 move as chosen by the planner.

Without the backtracking mechanism, the planner would have failed to find a path in this situation. With the backtracking mechanism, the planner determines

[1] We may consider a range of the parameter values (instead of a single "blocking" parameter value) over which the current link is blocked. This range could be determined by a forward search from the start node to the node with the maximum reachable parameter value of t_i, and a backward search from the goal node to the minimum reachable parameter value of t_i. Then the edge segments on the path, which correspond to this range of the parameter t_i, are deleted from the V-graph.

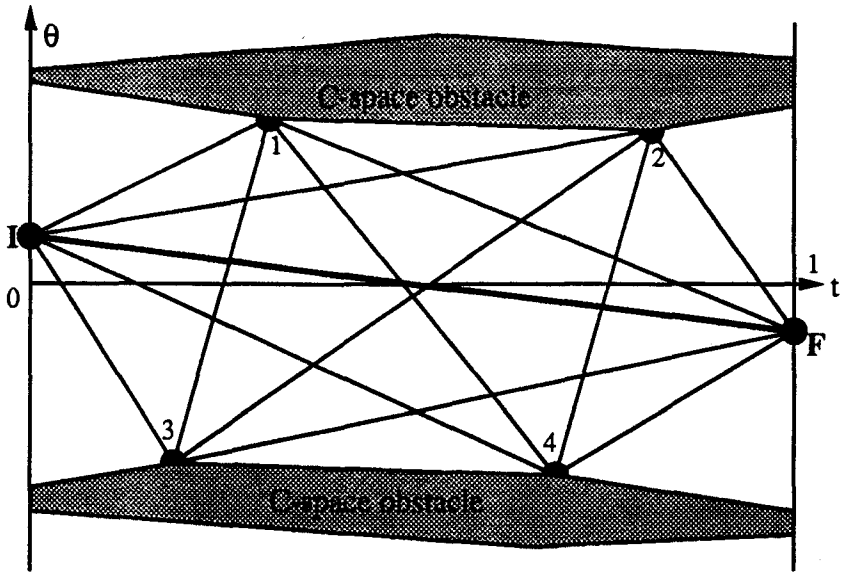


Figure 4.19: The V-graph for link 2. The shortest path is **IF**; the planner chooses it, but does not find a free path for link 3 (See Figure 4.20).

the parameter value where the "block" takes place, and goes back to the $t_2 \times \theta_2$ space for link 2. It deletes the edge segment that correspond to the blocked parameter, in this case, the edge segment **IF**. In the pruned V-graph, the planner searches for another path for link 2. Along this new path for link 2, the planner builds and searches the $t_3 \times \theta_3$ space for a path for link 3 again. This process repeats until a path is found for link 3. In this particular example, a collision-free path for link 3 is generated after 15 backtracking searches. Figure 4.21 shows that edge segments **IF**, $\mathbf{I}N_1$, $\mathbf{I}N_2$, $N_3\mathbf{F}$ and $N_4\mathbf{F}$ have been removed before the planner finds a path along which a collision-free path exists for link 3.

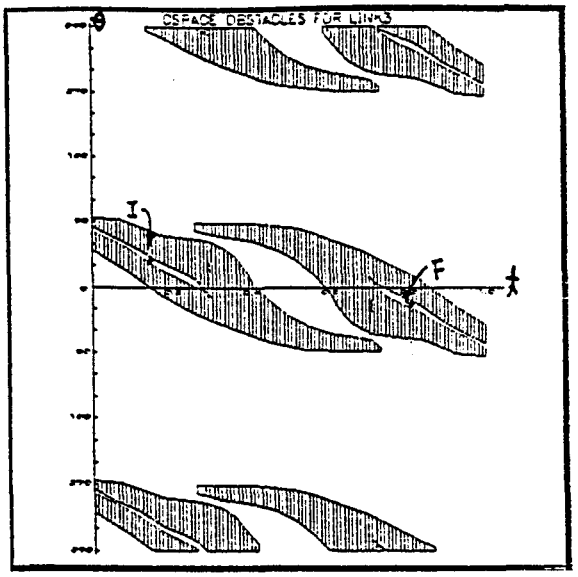


Figure 4.20: The V-graph for link 3. The initial position, I, and final position, F, are trapped in the forbidden region since link 2 moves along the path IF shown in Figure 4.19.

4.8 Complexity Analysis

A full complexity analysis is beyond the scope of this thesis. We briefly present results from Gupta and Guo [5] where a more detailed analysis is given. Since we compute discretized approximations in $t_i \times \theta_i$ space, there is no straightforward correspondence between the polygonal forbidden regions in $t_i \times \theta_i$ space, and the obstacle and the robot geometry. Hence, we present a fairly abstract analysis of the planner. Let f_i be the complexity of building the 2-dimensional sub-space $t_i \times \theta_i$, g_i is the complexity of searching this 2-dimensional sub-space and p_i is the number of paths explored in this sub-space. f_i, g_i and p_i are all functions of the number of vertices, faces and edges in each of obstacles and robot links. Assume that the upper bounding values of f_i, g_i and p_i are given as f, g and p, respectively. The

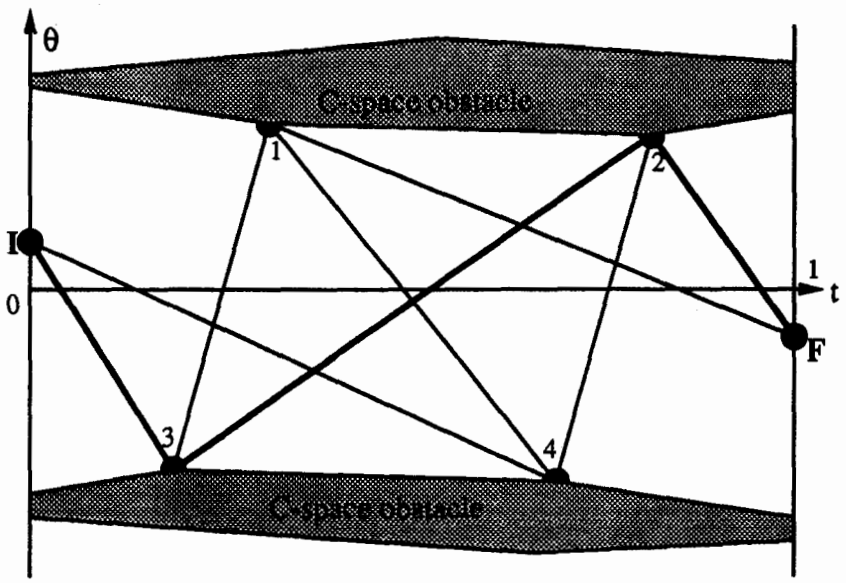


Figure 4.21: The pruned V-graph for link 2 after 15 backtracking searches is shown. Along the new path $\mathbf{I}N_3N_2\mathbf{F}$, the planner successfully finds a free path for link 3 (See column (d) in Figure 5.6).

overall complexity of the planner with level k backtracking is [5]:

$$O(np^k(2g+f)).$$

Furthermore, our representation is based on visibility graph in the $t_i \times \theta_i$ space. Suppose there are a maximum of m vertices in the $t_i \times \theta_i$ space. Since an edge is deleted every time the planner backtracks, at most $O(m^2)$ paths are possible in $t_i \times \theta_i$ space for any i, i.e., p is $O(m^2)$ [31]. Building and searching a visibility graph is then in time $O(m^2)$. Note that the major part of the time complexity in $O(m^2)$ is the time spent on computing the $t_i \times \theta_i$ space obstacles (this can be seen from our experimental result in chapter 5). Hence, the time complexity of our motion planner with the backtracking mechanism is

$$O(nm^{2k+2})$$

for a backtracking level k. Thus, the planner complexity is exponential in k, however, for a given k, the planner complexity will be polynomial in m. In practice, k will be a small integer (1, 2, or 3).

CHAPTER 5

Experimental Results

We have applied our motion planner to several examples. In this chapter, we report our results on six examples. In Figure 5.1, (a) shows the motion of a six-link manipulator arm and (b) shows the corresponding $t_i \times \theta_i$ spaces for each link. In Figure 5.2, (a) shows motions of a PUMA 560 arm and (b) shows the $t_i \times \theta_i$ space for each of robot links. Figure 5.3 shows the motions of another six-link manipulator arm (this example is similar to the one used by Barraquand and Latombe and has been chosen for comparison purposes), and Figure 5.4 shows the $t_i \times \theta_i$ space for this arm. Figure 5.5 show the motions of a three-link manipulator arm along the collision-free path found by our motion planner with the backtracking mechanism, and Figure 5.6 show the $t_i \times \theta_i$ space for Figure 5.5. In Figure 5.7 and 5.8, the collision-free motion was generated for a space arm. Figure 5.9 shows the motion planned for the same manipulator, but in a more complex environment ("H" shaped obstacle) for the space arm. Figure 5.10 shows the $t_i \times \theta_i$ space for it. The run times given in the Tables are with the planner running on a SUN Sparc station I.

The first example shown in Figure 5.1 has been chosen to show how our sequential planner works for manipulators with many degrees of freedom and to illustrate the role of backup movements in our planner. The planned collision-free motion is shown in Figure 5.1 (a). Note that the initial (top left corner) and the final positions (bottom right corner) are the same for the first five links. The corresponding $t_i \times \theta_i$ spaces are shown in Figure 5.1 (b). For link 1, there are four intervals that are forbidden. The horizontal axis represents the parameter $t_1 \in [0, 1]$ and the vertical axis represents the angle θ_1. Note that the whole free interval in which the initial and the final angle values θ_1^s and θ_1^g lie, is parametrized, and the parameter values t_1^s and t_1^g, which correspond to the start and the final angle values θ_1^s and θ_1^g, respectively, is determined. The line segment joining (t_1^s, θ_1^s) and (t_1^g, θ_1^g) determines the collision free path for link 1. This path is discretized and parameterized using the equal-length parametrization. This associates a discrete value θ_1 with each parameter value t_2. The forbidden ranges at each of these discrete parameter values are then determined, and the corresponding $t_2 \times \theta_2$ space is shown in Figure 5.1 (b). Since the initial and the final positions are the same for link 2, the path is trivial. However, note the additional extension segments from the start node to a node with minimum parameter value (in this case, 0.0) and from the final node to a node with maximum parameter value (in this case, 1.0). The $t_3 \times \theta_3$ space is then built along this extended path. This process repeats for links 3, 4 and 5. For the final link, this need not be done. Were these extension parts not chosen, the path for link 6 would not have been found. With the extension parts, however, a path is found as shown in $t_6 \times \theta_6$ space in Figure 5.1 (b). In order to avoid obstacles, the path for link 6 goes around an obstacle (in the middle) in the $t_6 \times \theta_6$ space. In the first segments of this path, the parameter value decreases to less than the value corresponding to the

initial configuration, which implies that the previous links move beyond their initial position. The same applies to the final configuration. Thus the manipulator arm "shrinks" first and then "stretches" again to its final configuration.

The next example is for the first three links of a PUMA-560 arm. In this example, there are five polyhedral obstacles in PUMA's workspace (including the bottom table which is also treated as an obstacle). The initial position of the forearm (link 3) is in between two vertical polyhedra, and the final position of the forearm is under the horizontal polyhedron [see Figure 5.2 (a)]. There is no collision when it moves along the path generated by our motion planner. Figure 5.2 (a) shows the motions and (b) the $t_i \times \theta_i$ space for the first three links. The blank rectangular regions represent the joint limits for each of the joint. This example was also implemented on a real PUMA 560 manipulator. The PUMA 560 is connected to a SUN 3/50. The obstacle dimensions were measured and manually entered in the machine. The PUMA 560 dimensions were taken from the user's manual. The output of the planner was downloaded to the PUMA controller which moved the PUMA along the collision-free path.

The example shown in Figure 5.3 is similar to that used by Barraquand and Latombe. They have planned the motion for this example with their numerical potential field approach with random search. This example has been deliberately chosen for comparison purposes and shows that our motion planner can find collision-free paths even in fairly tight situations.

The next example has been chosen to illustrate the backtracking mechanism in our planner. This has made our motion planner more powerful, especially in more cluttered environments. Consider the example shown in Figure 5.5. The initial

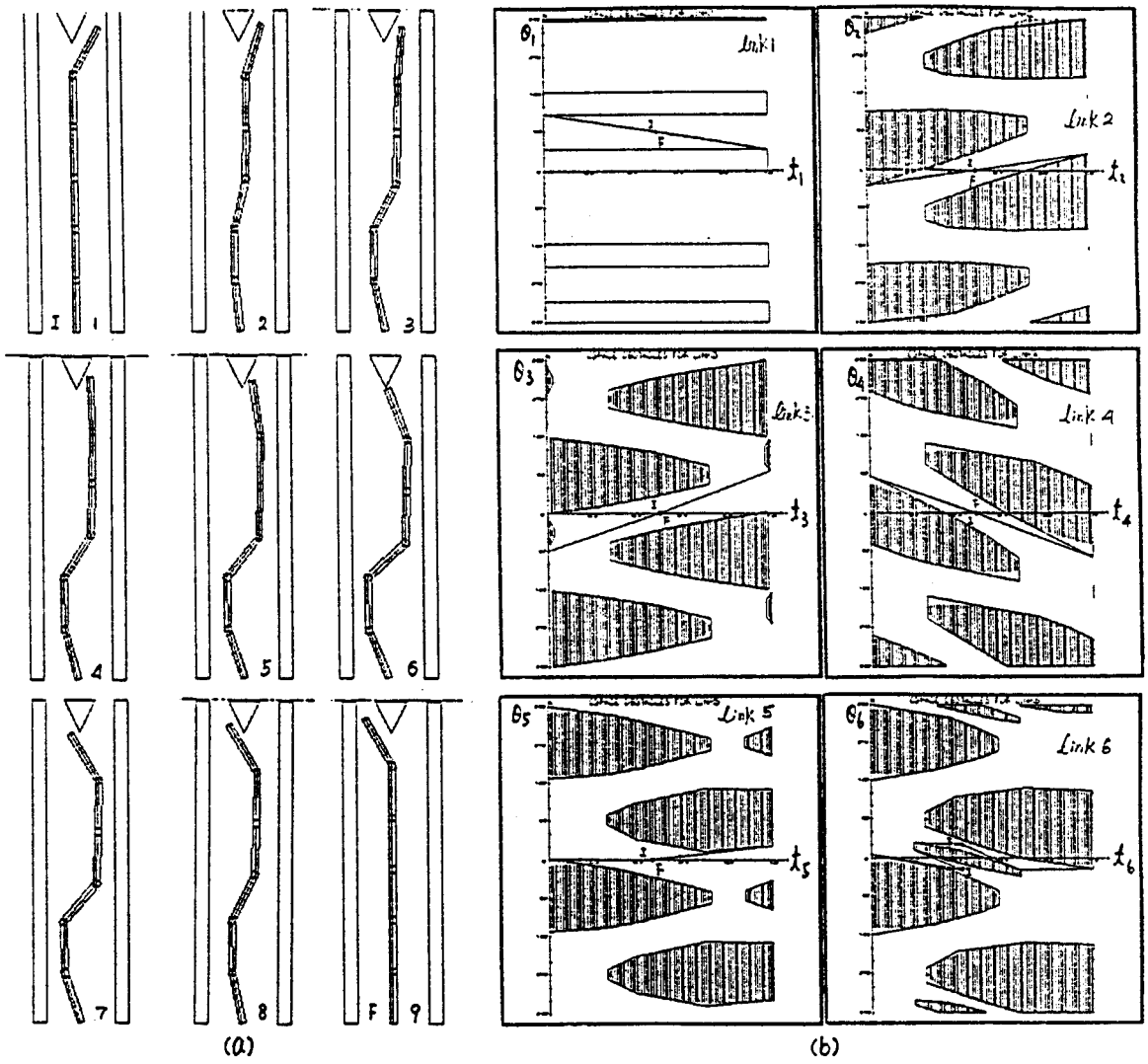


Figure 5.1: An example of a six-link arm with three obstacles. The path was found by our motion planner with the backup movement consideration. (a) the planned motions. The initial and final configurations are located at the top left corner and at the bottom right corner respectively. (b) the $t_i \times \theta_i$ space for each link.

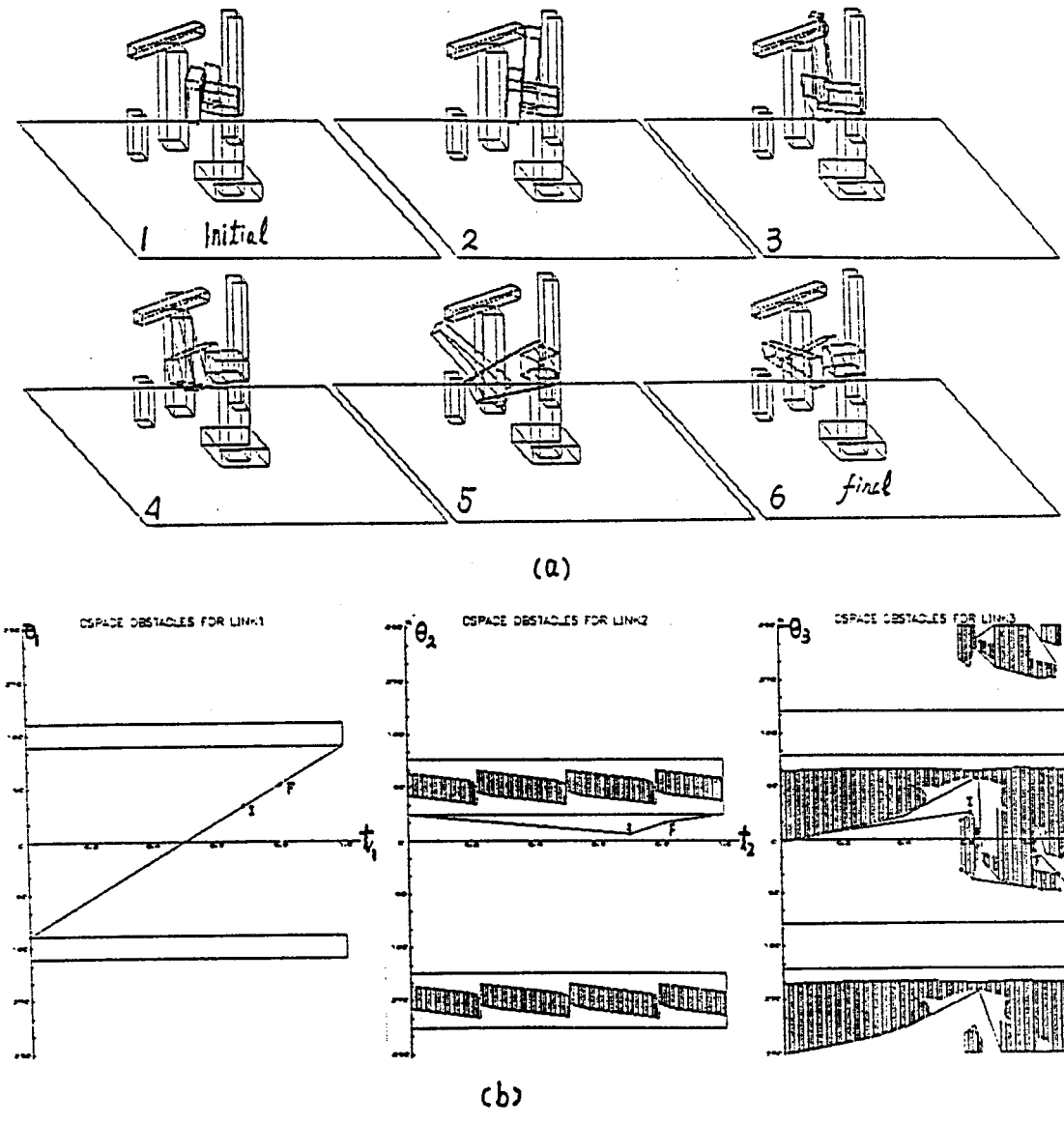


Figure 5.2: A planned motion for a PUMA-560 arm when several polyhedra exist. Along this path, the PUMA-560 arm moves without any collision. (a) illustrates the motion and (b) shows the $t_i \times \theta_i$ space for each link.

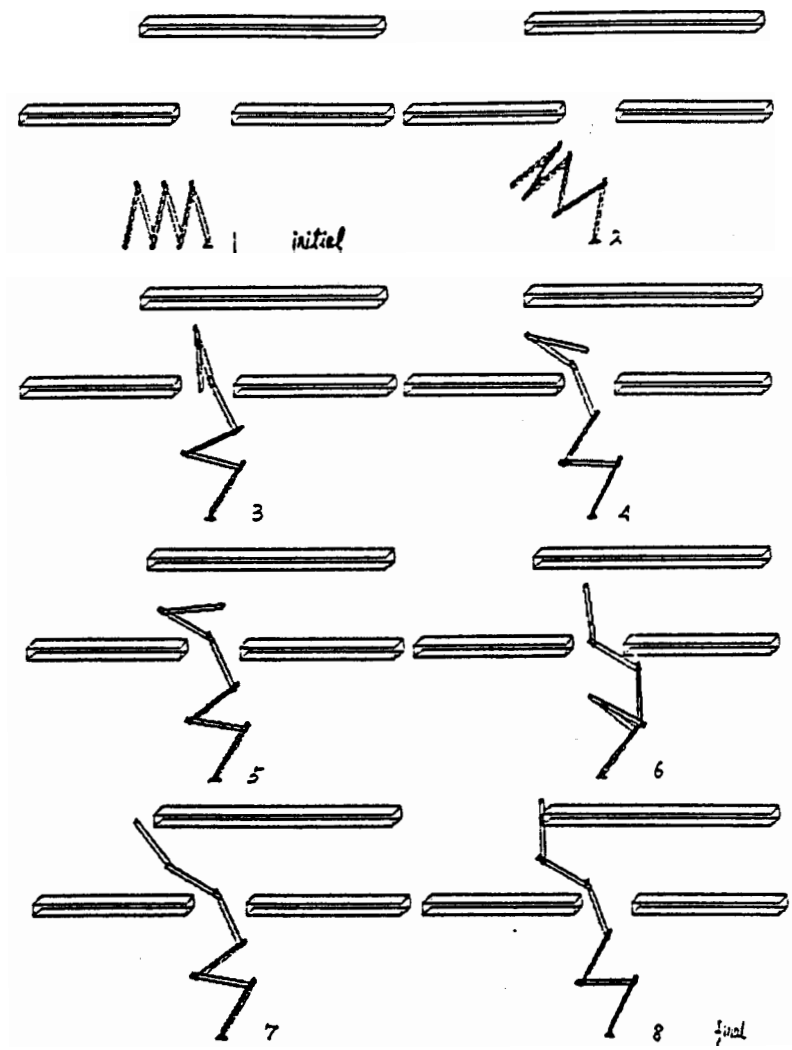


Figure 5.3: The planned motion for another six-link manipulator arm. This example is similar to that used by Barraquand and Latombe. The motion for this kind of manipulator arm has also been planned with their potential field approach.

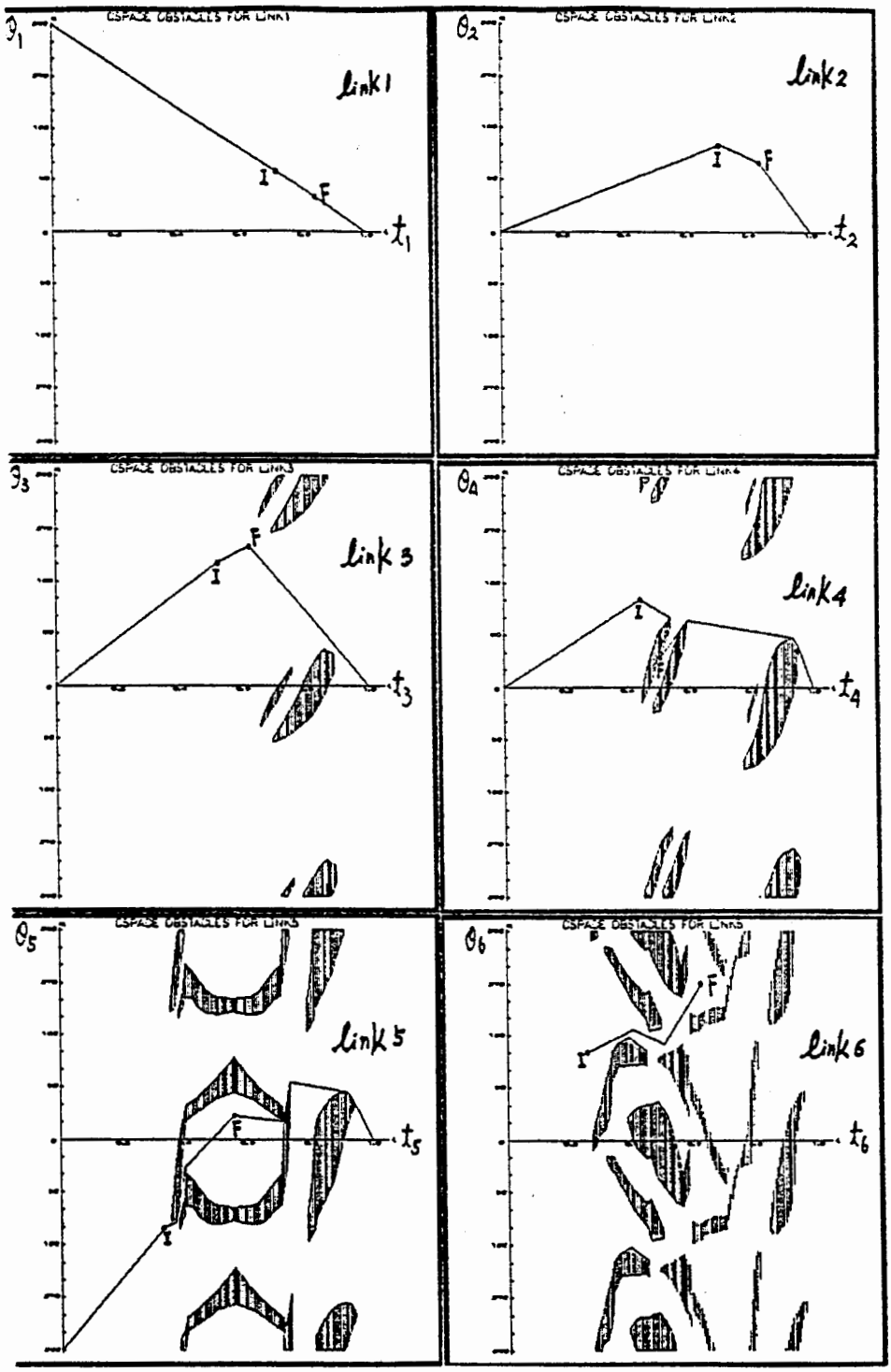


Figure 5.4: The $t_i \times \theta_i$ space for each link of the arm shown in Figure 5.3.

position of link 3 is illustrated at the top left corner. In this case, link 3 is blocked by two obstacles quite tightly. It can not move out from between obstacles 3 and 4 when the first two links move counterclockwise. A similar situation exists at the final position of link 3. Figure 5.6 shows the procedure of backtracking searches. In (a), both initial and final nodes are trapped in the forbidden regions. It implies that when link 1 rotates counterclockwise and link 2 rotates clockwise but not far enough, the initial node for link 3 is trapped in the forbidden area. A similar explanation holds for the final link. In (b), the initial node is in the forbidden region, and in (c) the final node is in the forbidden region. Our motion planner with backtracking mechanism has planned a free path after 15 backtracking searches, and (c) shows the final result after these 15 backtracking searches.

We have also applied our motion planner to the first four links of the proposed special purpose dextrous manipulator (SPDM) to be used in the space station project. For brevity, we will call this the space arm. Figure 5.7 shows the planned motion of the space arm, and Figure 5.8 shows the $t_i \times \theta_i$ spaces for each link. Since the path for link 2 is a straight line (generated at the first search, but not shown in the figure), no path exists for link 3 along this path. This brings out one shortcoming in the backtracking mechanism based on edge deletion, i.e., the set of paths considered is a proper subset of the paths in the V-graph. However, if the V-graph is very sparse, i.e., the number of nodes is very small in the V-graph, the number of paths explored is very small. For an extreme case (such as link 2 of this example), there is no obstacle in the corresponding $t_2 \times \theta_2$ space, and the only path is the straight line connecting the source and target nodes. Our approach is to add a predetermined small number of nodes in the V-graph, for instance, we have inserted two nodes in the middle arbitrarily. Then the planner backtracks and searches for

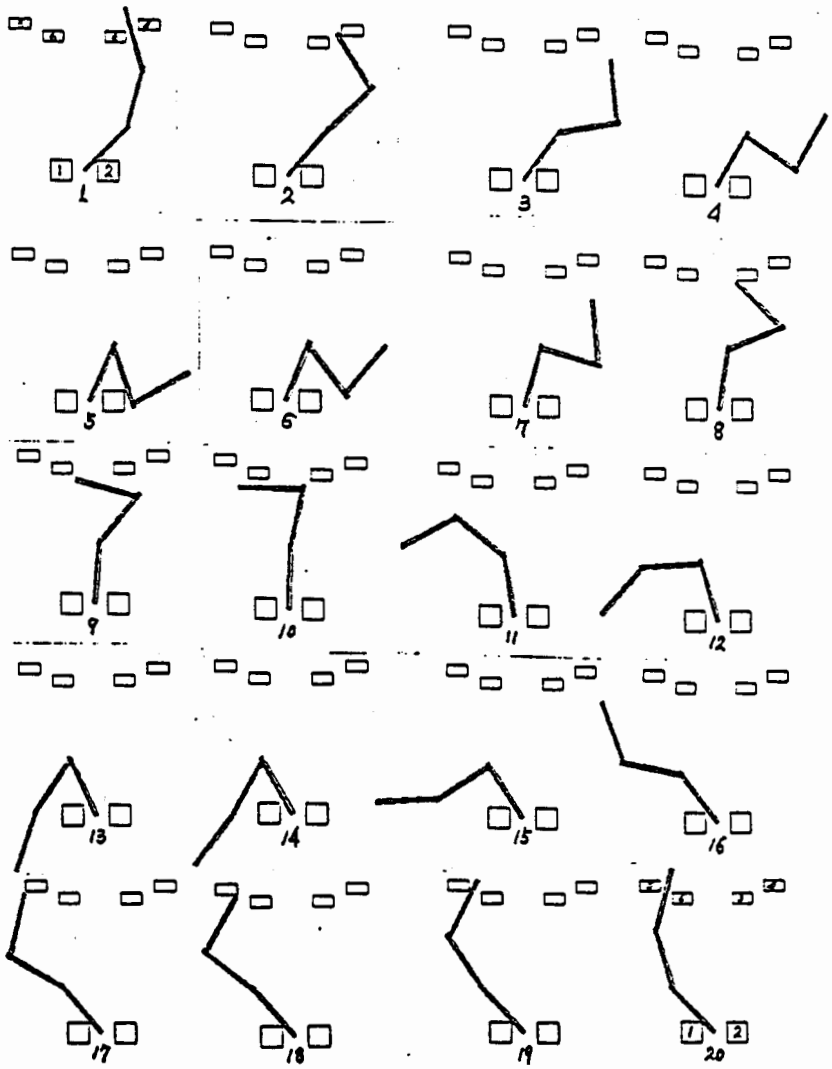


Figure 5.5: Another planned motion for a three-link arm. In this case, the environment is filled with six obstacles. The third link is blocked by obstacle 3 and 4 at its initial, and by obstacles 5 and 6 at its final position, respectively.

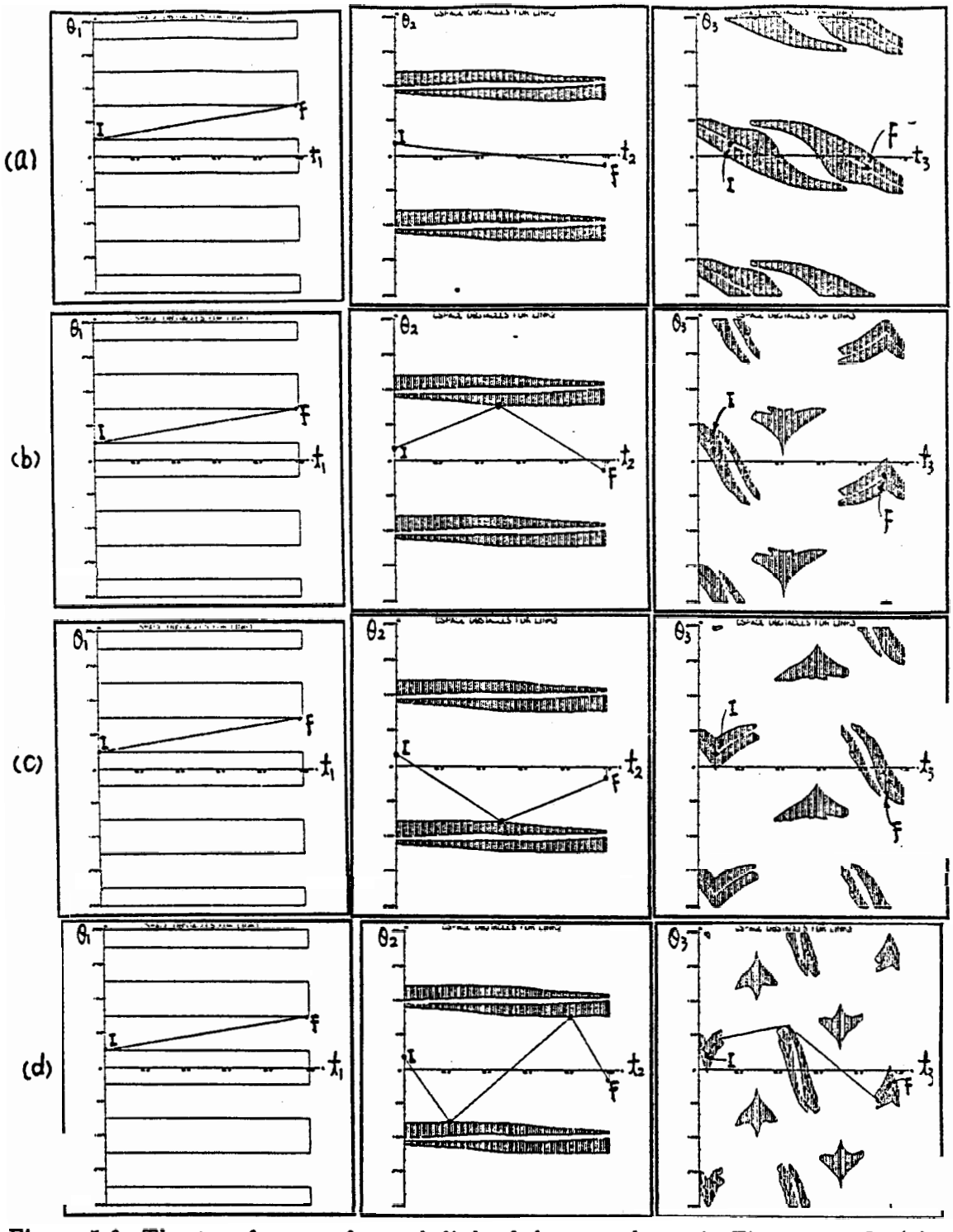


Figure 5.6: The $t_i \times \theta_i$ space for each link of the arm shown in Figure 5.5. In (a), the initial and final nodes are trapped in the forbidden area for link 3. In (b) and (c), the initial and final nodes are surrounded respectively. After 15 backtracking searches, a free path is found, as shown in (d).

another path in $t_2 \times \theta_2$ space. Along this new path, it plans the motion for link 3 again. Note that the planned motion was finally generated after one backtracking search to the second link (the $t_i \times \theta_i$ space for the backtracking search procedure is ignored here).

Finally, another example for the space arm with an "H" shaped obstacle is shown: the planned motion in Figure 5.9 and the $t_i \times \theta_i$ spaces in Figure 5.10.

Next we present and discuss the run times for a few examples. In general, there are four major components of our planner for each manipulator link: (i) computing forbidden angles, (ii) building V-graph, (iii) searching for a collision-free path, and (iv) parameterizing the generated path. Recall that the worst case complexity of the planner is exponential in the backtracking level, k, but polynomial in n, the degrees of freedom, for a given k. The approximate run time for the planner can be obtained by multiplying the average run time for a single link by the number of single link problems solved (including the backtracking searches). For the example shown in Figure 5.5, the planner should backtrack one level (only one level for this particular example) to link 2 because of no path found for link 3. In this case, the backtracking level, $k = 1$. For this example, the planner backtracks to link 2 fifteen times, and the total run time is roughly $15 \times (62.941 + 50.242 - 48.987) = 962.940$ seconds (the run time shown in the second table is for solving one single link problem).

The run times (in seconds) given below are obtained by executing the planner on Sparc station I in a time-shared environment. Note that the run times for a dedicated machine will be faster.

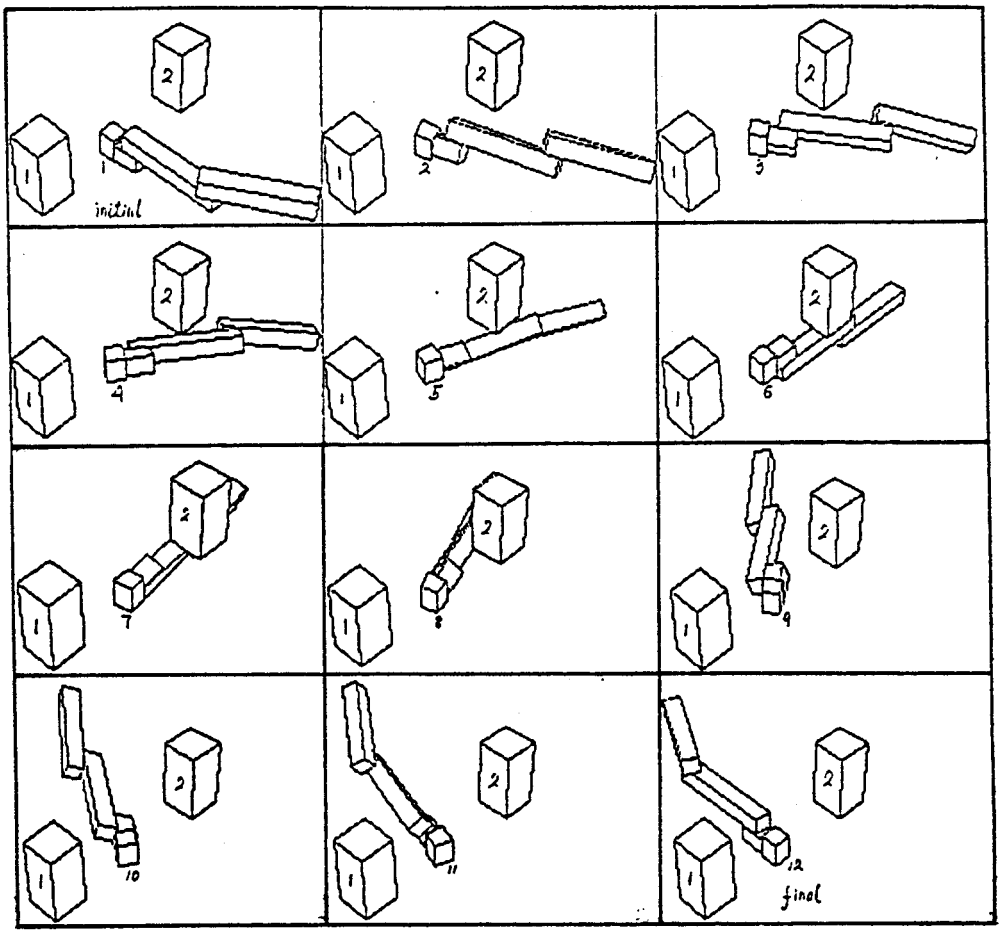


Figure 5.7: The planned motions for a four-link space arm. In this case, the environment is filled with two simple polyhedral obstacles, and the sequence of intermediate motions is shown from the top left corner to the bottom right corner.

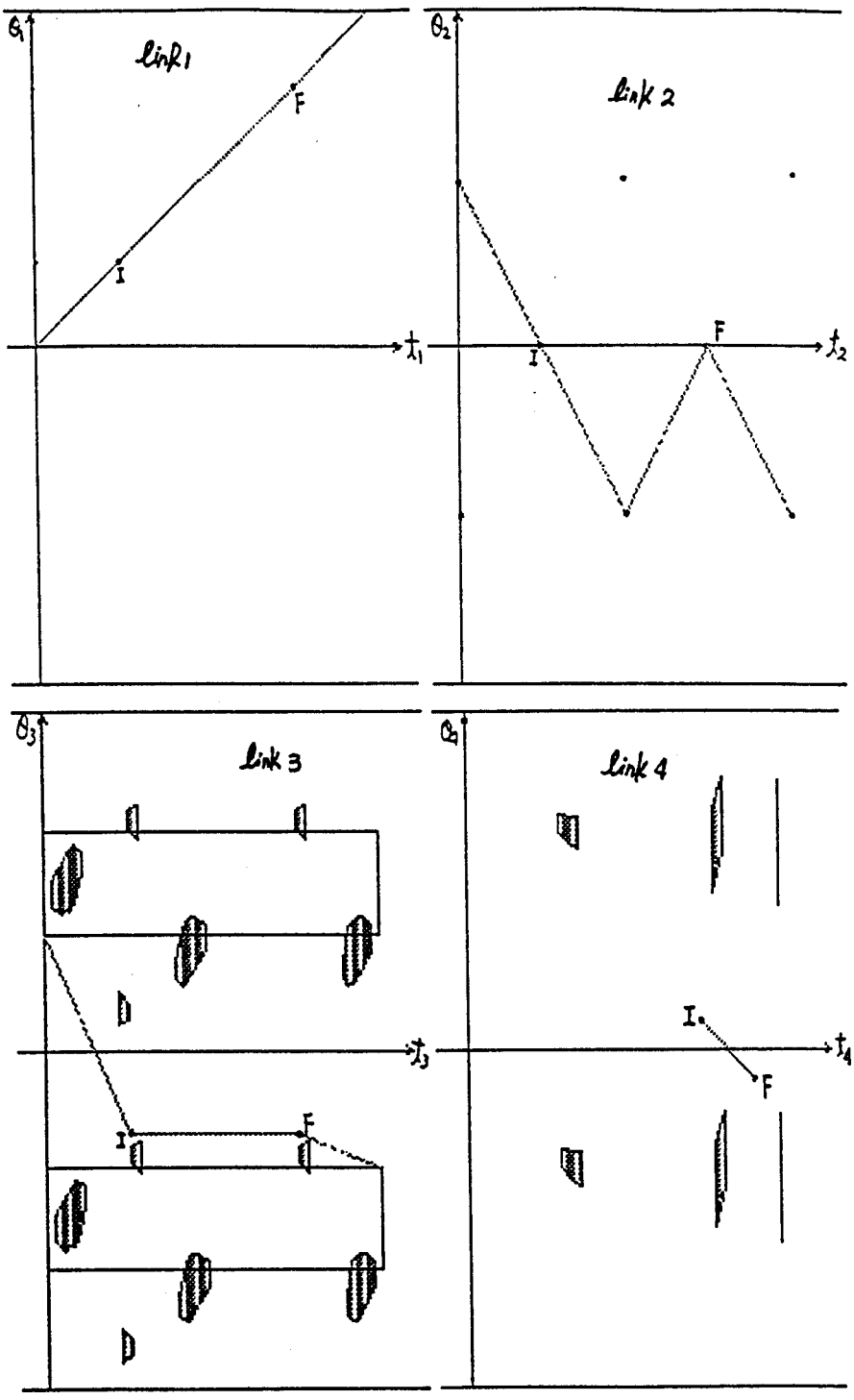


Figure 5.8: The $t_i \times \theta_i$ space for each link of the space arm shown in Figure 5.7.

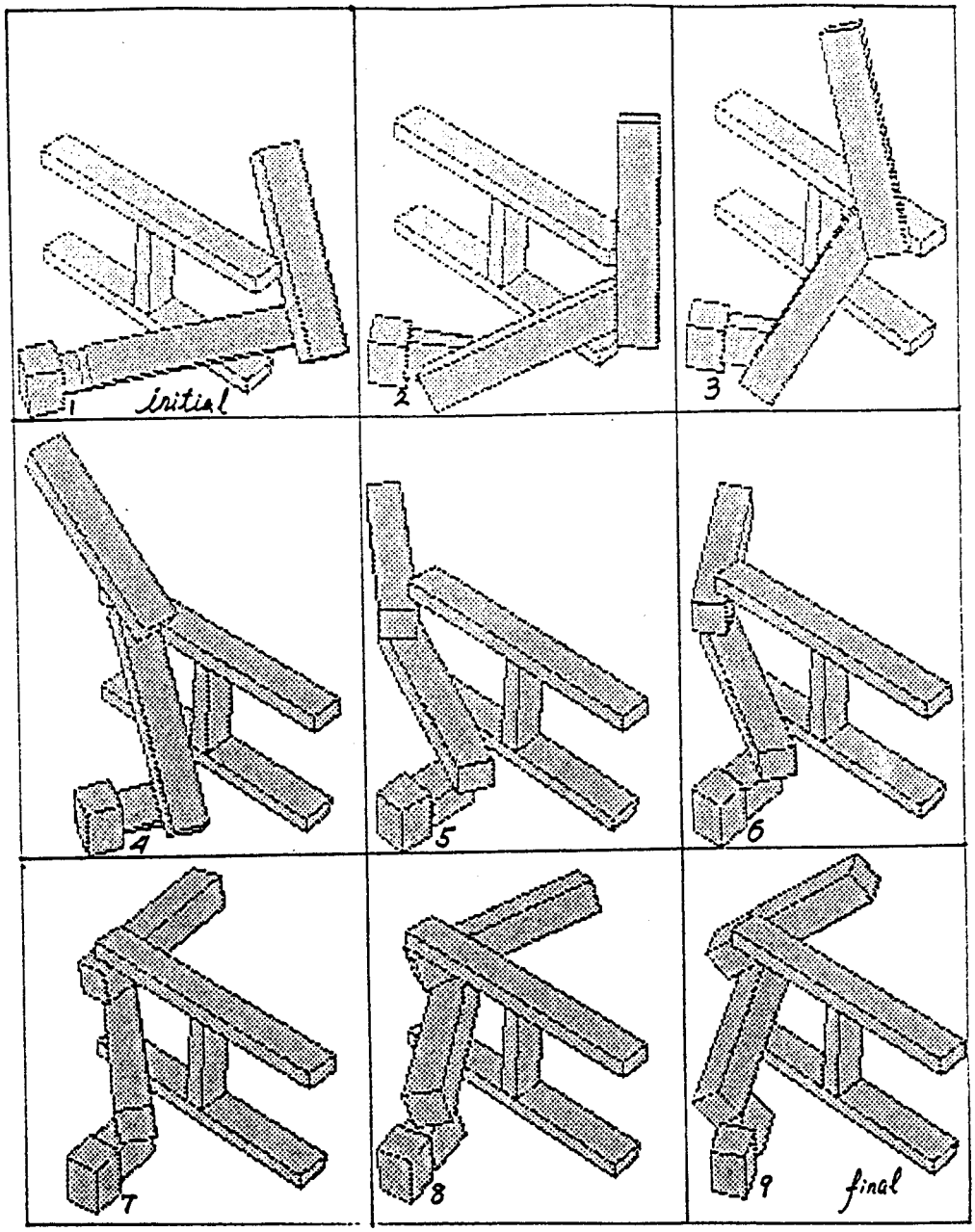


Figure 5.9: The planned motions for a four-link space arm in the environment filled with a "H" shaped obstacle. The sequence of intermediate motions is shown from the top left corner to the bottom right corner.

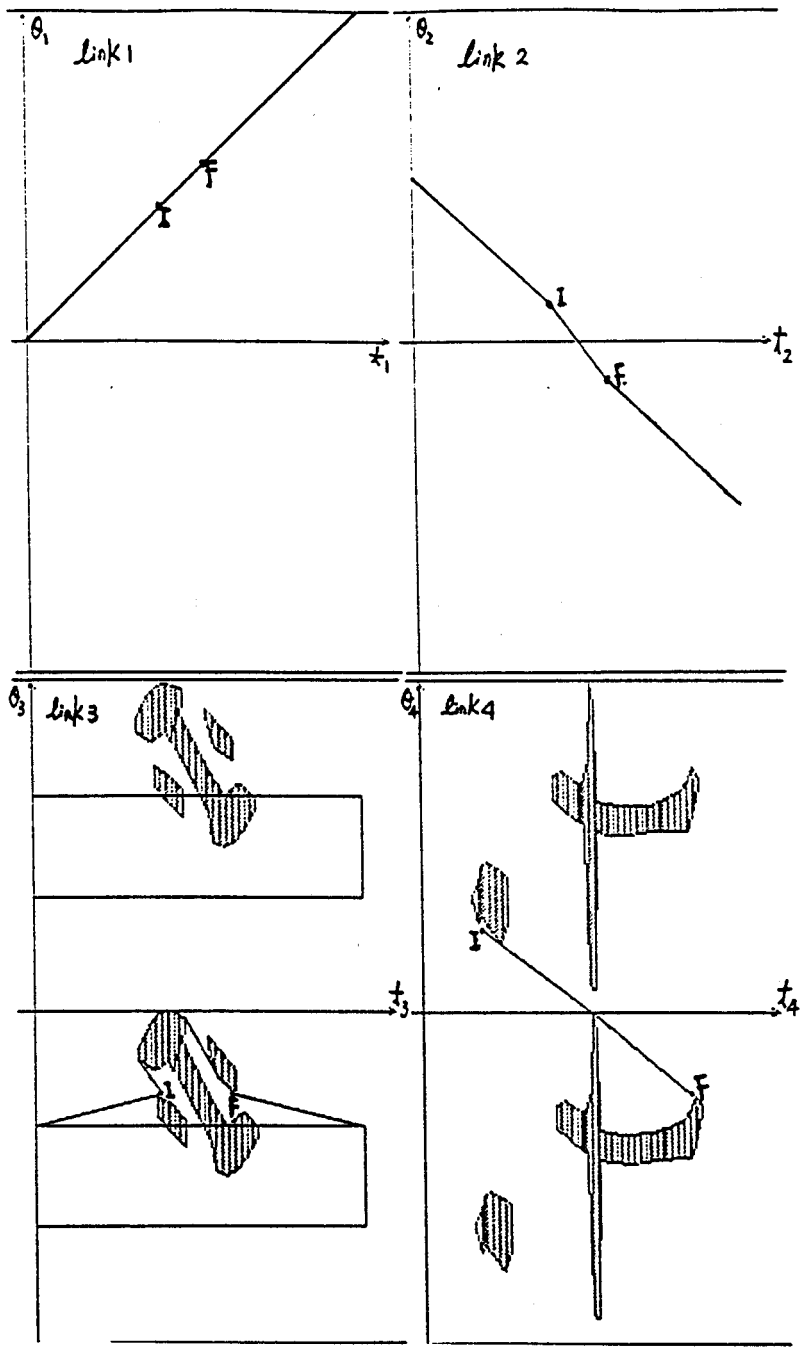


Figure 5.10: The $t_i \times \theta_i$ space for each link of the space arm in the case of "H" shaped obstacle.

Tables of Run Time

(in second)

(1) A six-link arm with three obstacles (Figure 5.1)

	forbidden angles	V-graph	searching	sampling	subtotal
Link 2	38.047	5.877	0.879	0.043	44.846
Link 3	24.667	2.260	0.716	0.024	27.667
Link 4	25.656	3.005	0.778	0.018	29.457
Link 5	31.171	2.643	0.804	0.024	34.642
Link 6	25.492	6.659	0.399	0.030	32.580
Total	145.033	20.444	3.576	0.139	169.192

(2) A three-link arm with six obstacles (Figure 5.5)

	forbidden angles	V-graph	searching	sampling	subtotal
Link 2	48.987	0.757	0.495	0.003	50.242
Link 3	53.524	8.333	1.078	0.006	62.941

The total run time is $962.940 = 15 \times (62.941 + 50.242 - 48.987)$, because of 15 backtracking searches to link 2.

(3) A four-link space arm with two obstacles (Figure 5.7)

	forbidden angles	V-graph	searching	sampling	subtotal
Link 2	16.007	0.151	0.509	0.004	16.671
Link 3	17.586	7.729	0.712	0.006	26.033
Link 4	16.895	1.530	0.208	0.008	18.653

The total run time is $72.047 = 2 \times (26.033 + 16.671 - 16.007) + 18.653$, because of one backtracking search to link 2.

(4) A four-link space arm with "H" obstacle (Figure 5.9)

	forbidden angles	V-graph	searching	sampling	subtotal
Link 2	49.034	0.248	1.923	0.003	51.208
Link 3	48.554	5.292	2.458	0.005	56.309
Link 4	55.170	7.780	1.059	0.011	64.020
Total	152.758	13.320	5.440	0.019	171.537

CHAPTER 6

Conclusions

6.1 Summary

In this thesis, a new motion planner for planning collision-free paths for a manipulator arm with many degrees of freedom among known stationary obstacles has been developed and implemented. This planner, which is based on a sequential search strategy, is efficient and applicable for many-degree-of-freedom manipulators simply because the n-dimensional motion planning problem is decomposed and simplified into simpler single link problems which are solved sequentially. A backtracking strategy is incorporated into the planner. The complexity of the planner is polynomial in n for a given level of backtracking, k. The core of the planner is the single link problem - plan a motion for one degree of freedom link (angle for a revolute joint) to avoid collisions as one end of the link moves along a given path. The path generated after solving the single link problem is then parameterized. Along this path, the planner solves the single link problem for next link until the last single link problem

(for the last link of the manipulator arm) is solved.

Although the planner is not complete, in most situations, it does find a collision-free path. However, it is well known that the complexity of the motion planning problem is exponential in n. Our planner, therefore, represents one way of developing practical motion planners for many-degree-of-freedom manipulator arms. We have demonstrated the effectiveness of the planner for a variety of manipulators up to six degrees of freedom in the order of a few minutes on a SUN Sparc station I. We can easily envisage the planner solving problems for manipulators with much larger number of degrees of freedom. In summary, our planner has a number of advantages: (i) it completely avoids the difficulty of representing n-dimensional C-space obstacles, (ii) it is efficient and practical – runs at the most in the order of a few minutes, even for manipulators with high number of degrees of freedom, (iii) it is deterministic (in the sense that it does not use stochastic search techniques), and (iv) it provides a mechanism for trading off execution speed of the planner with its completeness.

6.2 Some Open Issues

Developing and implementing a motion planner involves dealing with many areas – object modeling and representation, computational geometry, and search techniques. However, the treatment of these topics was far from exhaustive, even though some approaches have been suggested and experimented with. Next we discuss the author's selection of some possible extensions and some interesting and unsolved issues.

1. *Motion planning problems with multi-degree of freedom at one joint*

   Our implementation assumes that each joint has one degree of freedom. This is a very basic structure and most industrial manipulators fall under this category. There exist manipulators in which there is more than one degree of freedom at one joint, say three degrees of freedom at a spherical joint, such that a robot link can rotate around three Cartesian axes called "yaw", "pitch" and "roll". How do we extend our approach to such manipulators? There are two possibilities. First is that we could solve a higher dimensional problem, in this case, a four degree of freedom problem, i.e., as the base of the joint moves along a certain path, plan for all three degrees of freedom simultaneously. Another possibility would be to arbitrarily prioritize the degrees of freedom and plan the motion sequentially. However, there is no natural way to prioritize. In fact, the prioritizing may depend on the workspace and the manipulator structure.

2. *Motion planning problems with moving obstacles*

   Another problem is how to plan the motion of a manipulator when some of the obstacles in the robot's workspace are assumed to be moving along known trajectories with known velocity and acceleration. There are some approaches to solve some simple problems for moving obstacles [2, 3]. We believe that our approach can be combined with these approaches to plan the motions of a manipulator among moving obstacles.

3. *Motion planning problems with constraints on the robot hand*

   There are other versions of motion planning problem, for instance, the robot hand may be constrained to move on a planar surface. Extension of our planner

to solve such constrained motion planning problems is an interesting problem.

4. *Efficient motions*

   The consideration comes from the unnecessary movements of some robot links. See Figure 5.5. Link 2 rotates too far away from a "reasonable path". In fact, it is not necessary for link 2 to rotate further down when link 3 moves out from the "gap" between obstacles 3 and 4. The reason is that the motion planner only searches several nodes in the free $t_i \times \theta_i$ space. Definitely, a free-space based approach will give better and safer paths. We would recommend using a free-space based approach instead of the visibility graph based approach that we have used to solve the two-dimensional planning problem.

5. *Geometric reasoning and dynamic prioritizing*

   The success of our sequential planner lies in that it decomposes motion planning for an n-link into a sequence of simpler single-link problems, starting from the base link and finishing at the last link. Currently, we rely on the backtracking strategy to find another path for the previous link, if no path exists for the current link. However, the backtracking strategy is completely based on selecting paths in the $t_i \times \theta_i$ spaces. One way to improve it would be to use geometric reasoning in the Cartesian space to guide which path to select in the $t_i \times \theta_i$ spaces. In fact, in many situations, it may be better to use some other ordering (than the base link to last link ordering we use in our planner) or prioritizing of the links, i.e., solve the single link problems in a different order. In fact, this prioritizing of links will, in general, depend on the manipulator and obstacle configuration. We could, for example, use a geometric reasoning system to dynamically assign the link priorities, and then sequential planner

would then plan a path based on the link priorities.

6. *Completeness*

   Our planner is not complete, i.e., it may not find a collision-free path even if one exists. With the backtracking strategy, we have provided one way of trading off efficiency of the planner with completeness. However, even with full backtracking, the planner is not complete. An open issue is to consider variations of paths different than the one suggested in our backtracking methodology.

7. *Integrating the planner with a vision module*

   The planner requires a full geometric model of the obstacles in its environment. Currently, we manually enter this. Often it is assumed that these models will be derived from a CAD database. However, these are cumbersome processes. A better way would be to automatically acquire these models from sensed data, e.g., intensity or range images [6]. Such motion planners integrated with sensing modules would go a long way toward making future robots autonomous.

Appendix A

Coordinate Systems and Geometric Representations

A.1 Link Coordinate Systems

To describe each link of a manipulator arm, we use the standard notations as in [44]. The base frame of the robot coincides with its workspace coordinate system, which is usually called a *world frame*, or *universal frame*, and is denoted by **W**. The obstacle frame, **O**, is related to the world frame by a position vector $\mathbf{p_o}$. The link frames are named by number according to the link to which they are attached. That is, frame $\mathbf{L_i}$ is attached rigidly to link i. The convention we use to locate frames on the links is as follows: the Z axis of frame $\mathbf{L_i}$, called $\mathbf{Z_i}$, is coincident with joint i. The origin of frame $\mathbf{L_i}$ is located where the a_i perpendicular intersects the joint i axis. $\mathbf{X_i}$ points along a_i in the direction from joint i to joint $i + 1$. $\mathbf{Y_i}$ is formed by

the right-hand rule to complete frame $\mathbf{L_i}$. Figure 4.1 shows the whole coordinate systems.

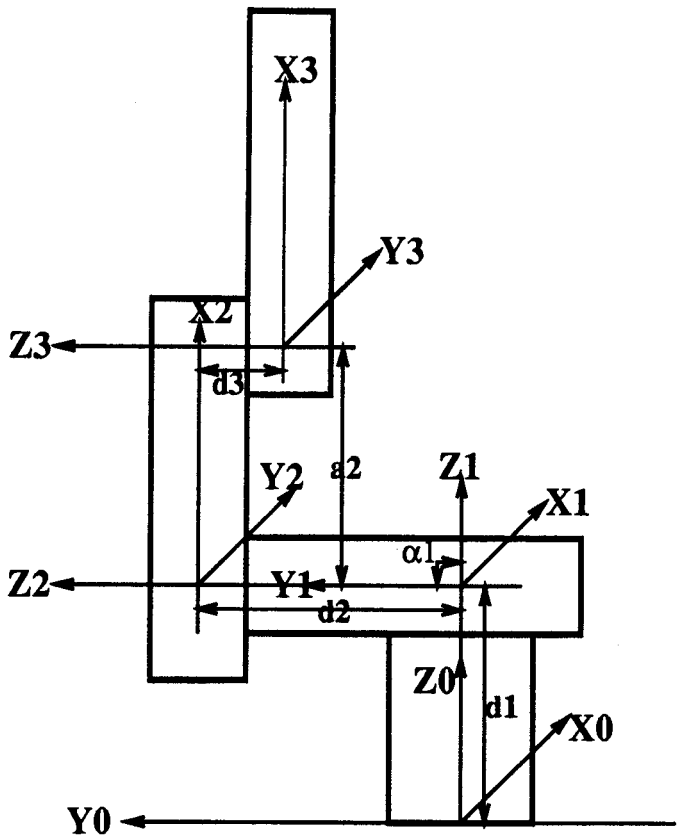
If the link frames have been attached to each link according to our convention, the following definitions of the link parameters are valid:

- a_i: the distance from axis z_i to axis z_{i+1} measured along axis x_i;
- d_i: the distance from axis x_{i-1} to axis x_i measured along axis z_i;
- α_i: the angle between axis z_i and axis z_{i+1} measured about axis x_i;
- θ_i: the angle between axis x_{i-1} and axis x_i measured about axis z_i;

Hence a robot manipulator arm can be represented kinematically by giving the values of four quantities for each link. In Figure A.1, we give an example of the definitions of link parameters (the example shows the first three links of PUMA 560, and the values of quantities given in the figure have been used in our implementation). We would like to mention again that there is only one degree of freedom on each joint. The purpose of our motion planner is to generate the value of θ_i at which no collision happens between a robot link and obstacles.

A.2 Geometric Models And Representations

In our implementation, robot links and obstacles are both represented by convex polyhedra which may have an arbitrary number of surfaces and vertices. We limit our computer representation to convex polyhedra because they greatly simplify calculations of configuration space for each robot link. For a nonconvex polyhedron,



Link Parameters

Link 1: **a0** = 0.0
$\alpha 0 = 0.0$
d1 = 26.45
$\theta 1$ (variable)

Link 2: **a1** = 0.0
$\alpha 1 = -90.0$
d2 = 9.2
$\theta 2$ (variable)

Link 3: **a2** = 17.0
$\alpha 2 = 0.0$
d3 = −4.0
$\theta 3$ (variable)

Figure A.1: An example of definitions of link parameters for PUMA 560

it can be decomposed into several convex polyhedra. For other kinds of models, like cylinders and spheres, we can use bounding convex polyhedra. The following sections give the data structures to describe the geometric models.

A.2.1 Data Structures For A Robot Arm

Usually, a robot arm is composed of several links, and one link is modeled as a cylinder or square bar. We described any robot link as a square bar which has six faces and eight vertices. Therefore, in our implementation, it is the basic structure of the robot link. The following are defined for implementation of our motion planner.

(1) The Data Structure: Arm

```
typedef struct arm {
        int          num_links;
        Link-state link-state[MAXLINKS];
        Link-style link-style[MAXLINKS];
        } Arm;
```

In this structure,

- num_links: the number of links of the given robot arm.
- link-state: a data structure which describes the state of the specified robot link.
- link-style: a data structure which describes the solid of the given robot link.

(2) The Data Structure: Link-state

The Link-state structure is defined as below,

```
typedef struct link-state{
        float       a;
        float       d;
        float       α;
        float       θ;
        } Link-state;
```

(3) The Data Structure: Link-style

The Link-style structure is defined as below,

```
typedef struct link-style{
        int         num_faces;
        Face        face[MAXFACES];
        } Link-style;
```

In this structure,

- num_faces: the number of faces os a robot link. The limit is MAXFACES which is defined as SIX because a robot link is described as a square bar which has at most six faces.
- face: a data structure which represents one planar surface of the robot link.

(4) The Data Structure: Face

The definition of Face is

```
typedef struct face{
        int             num_vertices;
        Normal          normal;
        Vertex          vertex[MAXVERTICES];
        } Face;
```

where

- num_vertices: the number of vertices of the surface. the maximum number is four according to *square bar* description of the robot link.
- normal: a data structure which describes the unit normal vector of the given surface.
- vertex: a data structure which represents the position of the vertex and the positions of its neighbors.

(5) The Data Structure: Vertex

The definition of Vertex is

```
typedef struct vertex{
        int             num_neighbors;
        Vector          point;
        Vector          neighbor[MAXNEIGHBOURS];
        } Vertex;
```

where

- num_neighbors: how many neighbors the specified vertex has. In our case, the maximum number is three(*square bar* definition).
- point: a data structure which is the position vector of the given vertex.
- neighbor: a data structure describing the positions of vertex neighbors.

A.2.2 Data Structures For Obstacles

Obstacles in a robot workspace are represented as convex polyhedra. The simplest model is a cube, a square bar, or a pyramid. We describe a group of obstacles by a obstacle list data structure, which definition is

```
typedef struct obstacle-list{
        int             num_obstacles;
        Obstacle        obstacle[MAXOBSTACLES];
        } Obstacle-list;
```

where

- num_obstacles: how many obstacles there are in the robot workspace. The maximum number depends on how many obstacles you want to simulate a workspace.
- obstacle: a data structure to describe an obstacle.

Here is the obstacle data structure.

```
typedef struct obstacle{
        int         num_faces;
        float       θx;
        float       θy;
        float       θz;
        Vector      position;
        Face        face[MAXFACES];
        } Obstacle;
```

where

- num_faces: same as in Link-style structure;
- θ_x: the rotation angle around the x-axis;
- θ_y: the rotation angle around the y-axis;
- θ_z: the rotation angle around the z-axis;
- position: the position vector of the obstacle in the robot workspace.
- face: same as in Link-style structure.

A.3 Transformation Between Coordinate Systems

As we know, the robot coordinate system is a multi-coordinate system. The robot workspace is defined with respect to a reference coordinate system which is called a *world system* in our approach. Therefore, a coordinate transformation is needed when a potential contact between a manipulator link and an obstacle is calculated, because manipulator links and obstacles are described in their own coordinate systems, respectively. There are two ways to do the coordinate transformation. One is to transform manipulator links and obstacles from their own systems to the reference system(*world coordinate system*). Another is to transform obstacles from their own systems to the current link coordinate system which is called a *link coordinate system.* In our implementation, the second has been selected since the final results are the joint angles in our motion planner. The transformation between *link coordinate system* has been given in [45].

Appendix B

Computing Potential Contact Angles In 3D

A potential contact angle is a value of any possible contact between a vertex and a surface or between two edges (we only discuss the case where the joints are rotary ones so that the configuration parameters are the joint angles). Suppose link k will be planned in the three-dimensional space, and also the models of link k and obstacles are convex polyhedra. We have defined the coordinate frame so that the origin corresponds to the position of revolute joint k and the **z** axis is aligned with the joint axis. The coordinate representation of all vectors (i.e. vertex vectors and surface normal vectors) is relative to this coordinate frame. When link k rotates around the **z** axis (the joint k axis), each of its vectors rotates the same value of the joint angle, and each point on the vector moves along a circle whose center is at a different level (different z value) along the **z** axis. We assume that the initial position of the link k corresponds to $\theta_k = 0$. The derivation in this chapter is based

on [9], but it is more detailed. Now let's compute the potential contact angle for three different types of contact.

(1) Type A contact: Vertex of a link with face of an obstacle

We are given a vertex of link k whose position vector is $\mathbf{v}$ and an obstacle surface whose plane equation is $\mathbf{n} \cdot \mathbf{p} + d = 0$ (here $\mathbf{n}$ is the outward pointing unit normal of the plane, $\mathbf{p}$ is a vector of any point on the surface and d is the distance between the origin and the surface). In order to determine the value of θ_k, we obtain the equation for θ_k by substituting the vertex position vector, rotated by θ_k, into the plane equation and then solving for θ_k. Let $\mathbf{v}'$ denote the rotated vertex vector, then the coordinates of $\mathbf{v}'$ are

$$\begin{aligned} v_x' &= v_x \cos\theta_k - v_y \sin\theta_k \\ v_y' &= v_x \sin\theta_k + v_y \cos\theta_k \\ v_z' &= v_z \end{aligned}$$

Substituting $\mathbf{v}'$ into the plane equation yields a simple trigonometric equation

$$(n_x v_x + n_y v_y)\cos\theta_k + (n_y v_x - n_x v_y)\sin\theta_k = -d - n_z v_z \qquad \text{(B.1)}$$

whose solution is

$$\theta_k = cos^{-1}(x/y) + \phi \qquad \text{(B.2)}$$

where

$$\begin{aligned} x &= -d - n_z v_z \\ y &= \sqrt{(v_x^2 + v_y^2)(1 - n_z^2)} \\ \phi &= \arctan(n_y v_x - n_x v_y, n_x v_x + n_y v_y) \end{aligned}$$

Now let's determine whether increasing or decreasing θ_k causes a collision, i.e., whether a contact angle is a *start* or an *end* of the forbidden range. See equation B.1. The left side represents the perpendicular distance of the rotated vertex from the obstacle surface. Therefore, the derivative of the left side of B.1 will determine whether the contact angle is a *start* of the forbidden range or an *end* of it. For type A contact, if $sign(dir) = -1$, the increasing of θ_k will cause a collision, i.e., this contact angle is the *start*, or if $sign(dir) = 1$, the decreasing of θ_k will cause a collision, i.e., it is the *end*. Here *dir* equals

$$dir = (n_y v_x - n_x v_y)\cos\theta_k - (n_x v_x + n_y v_y)\sin\theta_k \tag{B.3}$$

The orientation constraint simply tests whether the other endpoints of all the edges meeting at the contact vertex are outside of the obstacle surface. This is done by substituting the edge vectors into the left side of the plane equation and then if all the values are positive, the orientation constraint is satisfied.

(2) Type B contact: vertex of obstacle with surface of link

Similarly, we are given an obstacle vertex and a link surface. The procedure of computing θ_k is almost identical to the type A case. Here we mention that the only

difference is the sign of the first argument to the arctangent, and the *dir* will also be different from equation B.3. They are shown in following equations,

$$\begin{aligned} \phi &= \arctan(-n_y v_x + n_x v_y, n_x v_x + n_y v_y) \\ dir &= (-n_y v_x + n_x v_y)\cos\theta_k - (n_x v_x + n_y v_y)\sin\theta_k \end{aligned}$$

(3) Type C contact: edge of link with edge of obstacle

There is no doubt that this case is substantially more difficult. We make use of the results derived in [9], but we discuss it briefly. The basic idea is that the points on the edge can be represented by a parametric equation. Let $\mathbf{l}$ be the position vector of the intersection point on the link edge and $\mathbf{o}$ the position vector of the intersection point on the obstacle edge. Then the parametric equations for l and o are as follows:

$$\begin{aligned} \mathbf{l} &= t_l\mathbf{m} + \mathbf{v} \\ \mathbf{o} &= t_o\mathbf{n} + \mathbf{w} \end{aligned}$$

where $\mathbf{v}$ is the position vector of one of the endpoints of the link edge and $\mathbf{m}$ is the difference vector between the endpoints of the link edge, $\mathbf{w}$ is the position vector of one of the endpoints of the obstacle edge and $\mathbf{n}$ is the difference vector of the endpoints of the obstacle edge. The parameter $t_l \in [0,1]$ parameterizes along the link edge, and and $t_o \in [0,1]$ along the obstacle edge.

As the edge rotates around the $\mathbf{Z}$ axis, points on the edge trace out circles. The

equation for points on those circles are

$$x^2 + y^2 = (m_x t_l + v_x)^2 + (m_y t_l + v_y)^2 \tag{B.4}$$

$$z = m_z t_l + v_z \tag{B.5}$$

We assume $m_z \neq 0$, then combine these two equations above by solving the second equation for $t_l = (z - v_z)/m_z$ and then substituting into the first equation to obtain

$$x^2 + y^2 = \left(\frac{m_x}{m_z}(z - v_z) + v_x\right)^2 + \left(\frac{m_y}{m_z}(z - v_z) + v_y\right)^2 \tag{B.6}$$

Equation (B.6) is an implicit equation for points on the rotation surface.

Remember that when intersection happens, the intersection point on the obstacle edge must also satisfy equation (B.6). Consider the parametric form of this point as

$$x = n_x t_o + w_x \quad y = n_y t_o + w_y \quad z = n_z t_o + w_z.$$

Then substituting into (B.6) gives a quadratic equation in t_o, which is

$$p t_o^2 + q t_o + r = 0. \tag{B.7}$$

where

$$p = (n_x^2 + n_y^2)m_z^2 - (m_x^2 + m_y^2)n_z^2$$

$$
\begin{aligned}
q &= 2[(n_x w_x + n_y w_y)m_z^2 - (m_x^2 + m_y^2)(w_z - v_z)n_z - (m_x v_x + m_y v_y)m_z n_z] \\
r &= (w_x^2 + w_y^2)m_z^2 - [(m_x(w_z - v_z) + v_x m_z)^2 + (m_y(w_z - v_z) + v_y m_z)^2]
\end{aligned}
$$

Having two roots of t_o we can solve for two values of t_l since we know that the z values at contact must be equal. Therefore, the two values of t_l can be obtained by the following equation:

$$
t_l = \frac{n_z t_o + w_z - v_z}{m_z} \tag{B.8}
$$

For those values of t_o and t_l, only $t_o \in [0,1]$ and $t_l \in [0,1]$ are acceptable (satisfying the *in-edge* constraint). Having obtained the position vectors $\mathbf{l}$ and $\mathbf{o}$, the potential contact angle can be easily calculated, i.e.,

$$
\theta_k = \arctan(l_x o_y - l_y o_x, l_x o_x + l_y o_y). \tag{B.9}
$$

For the case where $m_z = 0$, we solve for t_o simply by $t_o = (v_z - w_z)/n_z$ in terms of equal z values. Then we solve quadratic equation (B.7) for t_l. We must note that the coefficients p, q and r are different from those used in (B.7). These coefficients are given by

$$
\begin{aligned}
p &= (m_x^2 + m_y^2) \\
q &= 2(m_x v_x + m_y v_y) \\
r &= v_x^2 + v_y^2 - (n_x t_o + w_x)^2 + (n_y t_o + w_y)^2
\end{aligned}
$$

In what follows we need to determine whether increasing or decreasing θ_k will cause a collision if the θ_k satisfies the orientation constraint. We have to make sure that the vector $\mathbf{c}(\theta_k)$ is the outward-pointing by a factor k, where $k = sign(\mathbf{c} \cdot (\mathbf{e_1} + \mathbf{e_2}))$. Then we obtain the type-C C-surface equation.

$$k\mathbf{c}(\theta_k) \cdot (\mathbf{w} - \mathbf{v}(\theta_k)) = 0 \tag{B.10}$$

If the derivative of the left side of the equation is positive, increasing θ_k will cause a collision.

Note that when we have already obtained a *start* value and an *end* value of the forbidden range, it is not necessary to compute the potential contact angle again since we consider that the robot links and obstacles are both convex polyhedra.

Appendix C

Geometric Constraints

We consider only the case where robot links and obstacles are convex polyhedra. Lozano-Pérez first defined three basic contacts in space, called Type A, B and C (see Appendix B for more details). The definitions of three types of contacts are shown in Figure 4.3. Lozano-Pérez then defined constraints in terms of those three types of contact [9]. In our implementation, we make use of his ideas and techniques.

There are different constraints for different types of contacts. For the contact between vertex and surface, there are two constraints: (1) *in-surface* constraint and (2) *orientation* constraint. For the contact between two edges, there is an *orientation* constraint. We will discuss them respectively.

(1) *in-surface* constraint

When we obtained a potential contact angle in between the link and an obstacle, we need to prove if it is the *real* one or not by checking the two constraints. The *in-surface* constraint means that the vertex must be within the surface of the obstacle

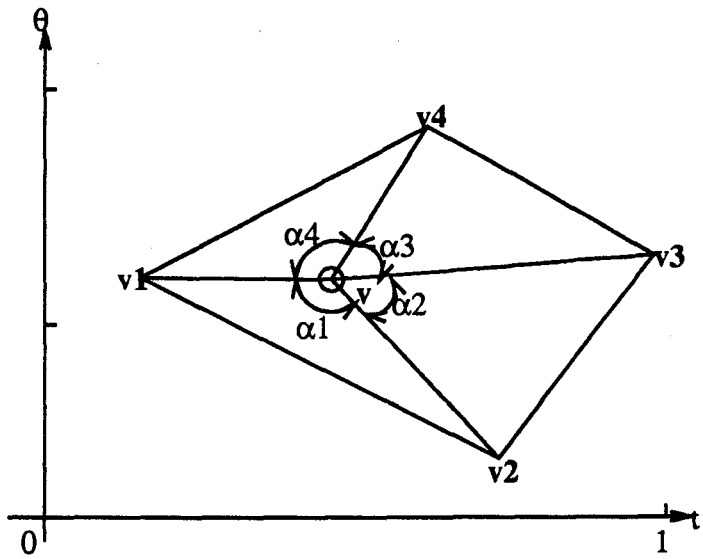


Figure C.1: If $\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 = 2\pi$, the vertex v is within the surface.

after rotating it by the value of the potential contact angle. In some cases, the rotated vertex may not be within the surface even though it satisfies the surface function, because a surface function can describe an infinite surface in the three-dimensional space. Since we know that the contact vertex is on the surface, all that remains to be determined is whether it lies within the surface. This can be done by computing the sum of angles which are formed by the vertex with each two vertices of the surface as shown in Figure C.1. The constraint is satisfied if the sum of the angles is equal to 2π. Note that for type A contact, the vertex has to be rotated by the value of the potential angle in the positive direction, and for type B, the rotation direction must be opposite to that for type A.

(2) *orientation* constraint

For *orientation* constraint, the orientation of the edges at the potential contact

point must be compatible, that is, the edges that define the contact vertex must be outside of the touched surface of the polyhedron. Therefore, the detection of the *orientation* constraint can be simple. All that is required is that the edges forming the contact vertex be outside of the contact surface. Practically, the way of testing this constraint is by ensuring that the polyhedron edges that intersect at the contact vertex point outward from the contact surface. This can be done by first rotating the contact vertex by the potential angle value, then checking the direction which the edges of the polyhedron point to. For the example shown in Figure C.2, if $\mathbf{e_1}, \mathbf{e_2}$ and $\mathbf{e_3}$ are the edge vectors pointing away from the vertex, and $\mathbf{n}$ is the unit normal vector of the contact surface, then the orientation constraint boils down to

$$sign(\mathbf{n} \cdot \mathbf{e_1}) \geq 0 \quad sign(\mathbf{n} \cdot \mathbf{e_2}) \geq 0 \quad sign(\mathbf{n} \cdot \mathbf{e_3}) \geq 0$$

where $sign(x) = x/|x|$ for $x \neq 0$ and 0 otherwise.

It is more difficult to detect the orientation constraint for type C contact. Theoretically, when the two edges are in contact (Figure C.3), any motion component perpendicular to both of them will cause a collision while a component of motion along either edge will not cause a collision. The direction perpendicular to both edges is simply the cross product of the two edge vectors (don't forget that the link edge must be rotated to the contact angle). The cross product generates a new vector, called vector $\mathbf{c}$, which makes the orientation constraint checking more convenient. From Figure C.3, $\mathbf{m}$ is an edge vector on the link, $\mathbf{e_1}$ and $\mathbf{e_2}$ are the other two edge vectors meeting at one end of edge $\mathbf{m}$; $\mathbf{n}$ corresponds to an edge vector on the obstacle; also $\mathbf{e_3}$ and $\mathbf{e_4}$ are the other two edge vectors meeting at one

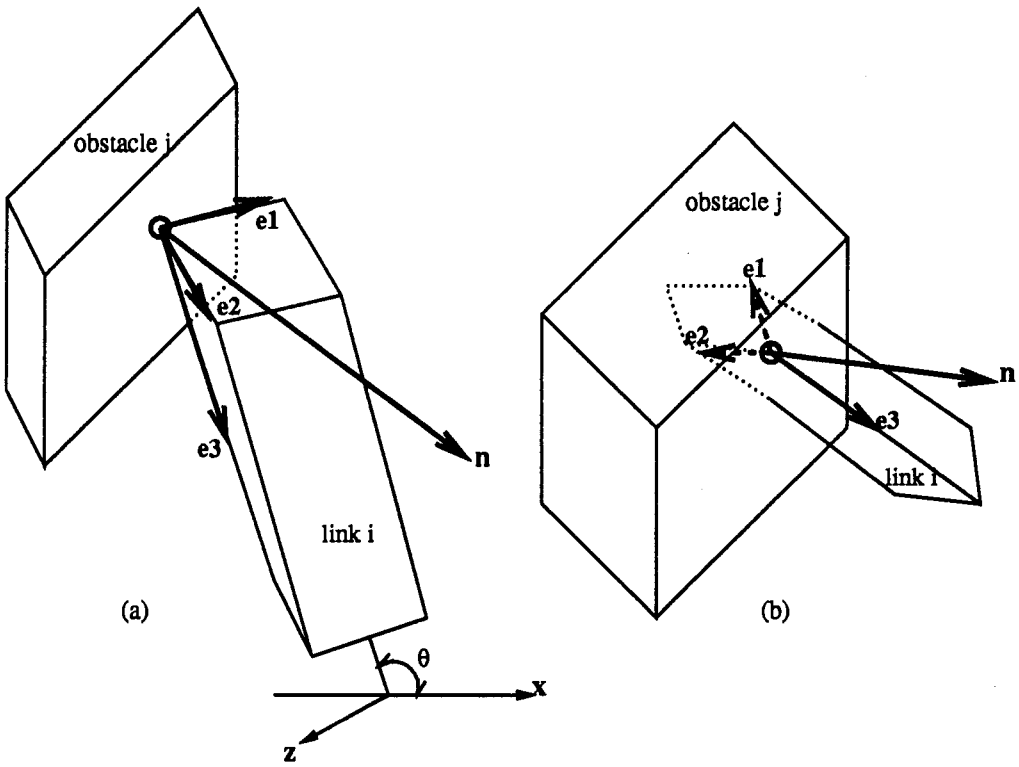


Figure C.2: Testing *orientation* constraint for polyhedral contact between vertex and surface. (a) the orientation constraint has been satisfied so that θ is a "real" contact angle; (b) the constraint is not satisfied.

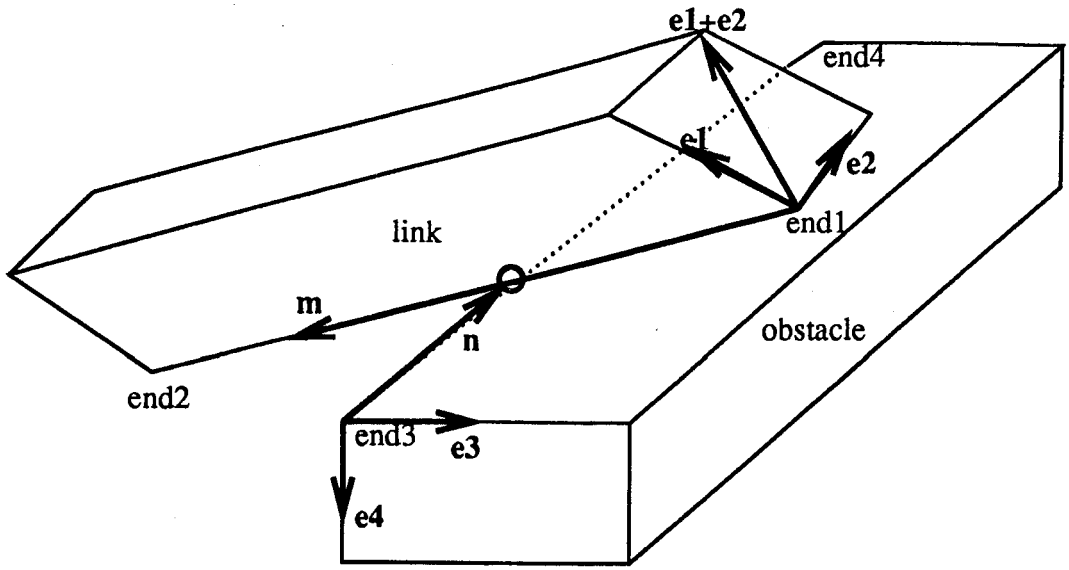


Figure C.3: Testing *orientation* constraint for polyhedral contact between two edges.

end of edge **n**. Then the followings are necessary and sufficient conditions for the orientation constraint for type C contact:

$$\mathbf{c}(\theta_k) = \mathbf{m}(\theta_k) \times \mathbf{n}$$

$$s = sign(\mathbf{c} \cdot \mathbf{e_1}) = sign(\mathbf{c} \cdot \mathbf{e_2})$$

$$s' = sign(\mathbf{c} \cdot \mathbf{e_3}) = sign(\mathbf{c} \cdot \mathbf{e_4})$$

whenever $s \neq s'$ is satisfied.

If the obstacle is a nonconvex polyhedron, there may be another constraint, called *reachability* constraint. This constraint, however, requires examining all the contacts

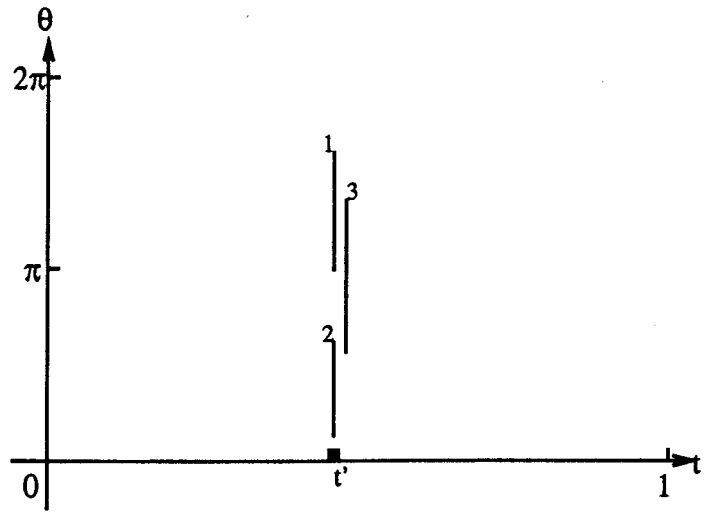


Figure C.4: At parameter t', three forbidden ranges are merged into one.

of the link with a given obstacle that satisfy the first two constraints. The idea is that for each contact angle θ we determine whether values of θ_k greater than θ cause collision or whether values of less than θ cause collision. According to the collision directions, the contact angles can be merged to form the ranges of forbidden values for θ_k. We restrict the model of obstacles to be convex polyhedra so that we don't need reachability checking in our motion planner. However, a nonconvex polyhedron can be decomposed into several convex polyhedra, but we still need to merge the overlapped values of θ_k just because the two obstacles are too close to each other. After the "merging" procedure, the *reachability* constraint is satisfied automatically. Figure C.4 shows the procedure of merging the overlapped ranges.

REFERENCES

[1] Gupta, Kamal Kant, 1990, "Fast collision avoidance for manipulator arms: a sequential search strategy", *IEEE Transactions on Robotics and Automation.* 6(5).

[2] Kant (Gupta), Kamal, and Zucker, S.W., 1986, "Toward efficient trajectory planning: the path-velocity decomposition", *International Journal of Robotics Research.* 5(3).

[3] Kant (Gupta), Kamal, and Zucker, S.W., 1988, "Planning smooth collision-free trajectories: path, velocity, and splines in free-space", *International Journal of Robotics and Automation.* 2(3).

[4] Gupta, Kamal Kant and Guo, Zhenping, June, 1991, "Toward practical motion planners: experiments with a sequential search strategy", *'91 ICAR Fifth Inter. Conf. on Advanced Robotics.* Pisa, Italy, pp. 1006-1011.

[5] Gupta, Kamal Kant and Guo, Zhenping, Oct., 1991, "Motion Planning for Many Degrees of Freedom: Sequential Search With Backtracking", CSS-IS TR 91-11.

[6] Gupta, Kamal Kant and Zhu, Xiao Ming, Oct., 1991, " Extracting Polyhedral Models From A Range Image: A Hybrid Approach", CSS-IS TR 91-10.

[7] Lozano-Pérez, Tomas and O'Donnell, P. A., April, 1991, "Parallel Robot Motion Planning", *IEEE Conference on Robotics and Automation*, Sacramento, CA., pp. 1000-1007.

[8] Lozano-Pérez, Tomas, March, 1989, "Task-Level Planning of Pick-and-Place Robot Motion," *IEEE Journal of Computer.*

[9] Lozano-Pérez, Tomas, June 1987, "A Simple Motion-Planning Algorithm for General Robot Manipulators," *IEEE Journal of Robotics Automation*, vol. RA-3, No.3.

[10] Lozano-Pérez, Tomas, October, 1981, "Automatic Planning of Manipulator Transfer Movements," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-11, No.10.

[11] Lozano-Pérez, Tomas, Feb., 1983, "Spacial planning: A configuration space approach," *IEEE Trans. on Comput.*, vol. C-32, pp. 108-120.

[12] Lozano-Pérez, Tomas and Wesley, M. A., October, 1979, "An Algorithm for Planning Collision_Free Paths Among Polyhedral Obstacles" *Communications of the ACM*, vol. 22, No. 10.

[13] Lozano-Pérez, Tomas and O'Donnell, P. A., April, 1991, "Parallel Robot Motion Planning", *IEEE Conference on Robotics and Automation*, Sacramento, CA., pp. 1000-1007.

[14] Faverjon, B., 1984, "Obstacle avoidance using an octree in the configuration space of a manipulator", *IEEE Conference on Robotics and Automation*, pp. 504-512.

[15] Faverjon, B. and Tounassoud, P., 1987, "A local based approach for path planning of manipulators with a high number of degrees of freedom", *IEEE Conference on Robotics and Automation*, pp. 1152-1159.

[16] Kodno, Koichi, June 1991, "Motion Planning with Six Degrees of Freedom by Multistrategic Bidirectional Heuristic Free-Space Enumeration", *IEEE Transactions on Robotics and Automation.* vol. 7, no. 3.

[17] Barraquand, J., Langlois, B. and Latombe, J.C., 1989. "Robot Motion Planning with Many Degrees of Freedom and Dynamic Constraints", *International Symposium of Robotics Research*, Tokyo, Japan.

[18] Barraquand, J., Langlois, B. and Latombe, J.C., 1989. "Numerical Potential Field Techniques for Robot Path Planning", Report No. STAN-CS-89, Department of Computer Science, Stanford University, CA.

[19] Barraquand, J., and Latombe, J.C., 1990. "A Monte-Carlo Algorithm for Path Planning With Many Degrees of Freedom", *IEEE Int. Conf. Proc. on Robotics and Automation*, Cincinnati, Ohio, pp. 1712-1717.

[20] Khatib, Oussama, 1986, "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots", *The International Journal of Robotics Research*, vol. 5, no. 1.

[21] Boyse, John W., January, 1979, "Interference Detection Among Solids and Surfaces" *Communications of the ACM*, vol. 22, No. 1.

[22] Donald, B., 1984. "Motion Planning with Six Degrees of Freedom", *MIT AI Tech. Rep.*, 791, pp. 504-512.

[23] Donald, B., 1987. "A Search Algorithm for Motion Planning with Six Degrees of Freedom", *MIT AI Tech. Rep.*, 791, pp. 504-512.

[24] Gouzenes, Laurent, 1984, "Strategies for solving collision-free trajectories for mobile and manipulator robots", *Int. Journal of Robotics Research*, 3(4). MIT Press.

[25] Hopcroft, Joseph, and Whitesides, S., 1985, "On the movement of robot arms in 2-dimensional bounded regions", *SIAM Journal of Computing*, 14(2).

[26] Korein, J., 1985, "A geometric investigation of reach", *MIT Press.*

[27] Schwartz, J. T. and Sharir, M., 1981, "On the 'Piano Movers' problem I. The case of a two-dimensional rigid polygonal body moving amidst polygonal barriers", Report No. 41. Dept. of Comp. Sc., Courant Institute of Math. Sc., New York.

[28] Schwartz, J. T. and Sharir, M., Oct. 1982, "On the 'Piano Movers' problem II. General techniques for computing topological properties of real algebraic manifolds", Report No. 41. Dept. of Comp. Sc., Courant Institute of Math. Sc., New York.

[29] Schwartz, J. T. and Sharir, M., 1988, "A Survey of Motion Planning and Related Geometric Algorithms", *Artificial Intelligence*, North Holland, vol. 37, pp. 157-169.

[30] Sharir, Micha, March 1989, "Algorithmic Motion Planning in Robotics", *Computer, IEEE Trans.*

[31] Udupa, S., 1977, "Collision detection and avoidance in computer controlled manipulators", *Proc. 5th Int. Joint Conf. Artificial Intell*, Cambridge, MA.

[32] Whitesides, S., 1985, "Computational geometry and motion planning", *Computational Geometry, Machine Vision and Pattern Recognition*, G. Toussaint(Ed.). Amsterdam, North Holland.

[33] Xing, D. M., 1987, "Collision-free trajectory generation for a general robot manipulator", Tech. Rep. CRIIF, Laboratoire de Robtique. Paris.

[34] Brooks, R. A., 1983, "Planning collision-free motions for pick and place operations", *Int. J. Robotics Res.*, vol. 2, no. 4.

[35] Brooks, R. A. and Lozano-Pérez T., Aug. 1983, "A subdivision algorithm in configuration space for findpath with rotation", *Proc. 8th Int. Joint Conf. on AI*, MIT AI Memo 684; (also *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-15, pp. 224-233, Mar./Apr. 1985).

[36] Hiroshi, Imai and Masao, Iri, 1986, "An Optimal Algorithm for Approximating a Piecewise Linear Function", *Journal of Information Processing*, vol. 9, no. 3, pp. 159-162.

[37] Canny, J.F., 1987. "The Complexity of Robot Motion Planning", Ph.D. Dissertation, Department of Electrical Engineering and Computer Science, MIT.

[38] Jun, Sungtaeg and Shin, Kang G., June 1991, "Shortest Path Planning in Discretized Workspaces Using Dominance Relation", *IEEE Transactions on Robotics and Automation.* vol. 7, no. 3.

[39] O'Dunlaing, C. and Yap, C. K., 1985, "The Voronoi diagram method motion-planning: I. The case of a disc", *J. Algorithm.* vol. 6, pp. 104-111.

[40] Suh, S. H. and Shin, K. G., 1987, "Robot path planning with a weighed distance-safety", in *Proc. 26th Conf. Decision and Control*, pp. 634-641.

[41] Glavina, Bernhard, June 1991, "A Fast Motion Planner for 6-DOF Manipulators in 3-D Environments", *'91 ICAR Fifth Inter. Conf. on Advanced Robotics*, Pisa, Italy, pp. 1176-1181.

[42] Hart, P. E., Nilsson, N. J., Raphael, B., 1968, "A formal basis for the heuristic determination of minimum cost paths", *Trans. Syst. Sci. Cybern.*, vol. SSC-4, No. 2.

[43] Brady, Michael 1989, "Robotics Science," chapter 2, "Planning" The MIT Press, Cambridge, Massachusetts, London, England.

[44] Horowitz, Ellis and Sahni, Sartaj, 1976, "Fundamentals of Data Structures," chapter 6, "Graphs," *Computer Science Press,* Potomac, Maryland.

[45] Craig, John J., 1986 "Introduction to Robotics," chapter 3, *Stanford University,* California.