National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service

Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

NL-339 (r. 88/04) c

Canada

UNIVERSITY OF ALBERTA

COLLISION-FREE MOTION PLANNING AND APPLICATION TO THE PUMA 560 MANIPULATOR

by

YUGUANG (EUGENE) CAO ©

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

DEPARTMENT OF ELECTRICAL ENGINEERING

EDMONTON, ALBERTA

FALL, 1990

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service

Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-65009-5

UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR: **YUGUANG (EUGENE) CAO**

TITLE OF THESIS: **COLLISION-FREE MOTION PLANNING AND APPLICATION TO THE PUMA 560 MANIPULATOR**

DEGREE: **MASTER OF SCIENCE**

YEAR THIS DEGREE GRANTED: **FALL, 1990**

PERMISSION IS HEREBY GRANTED TO THE UNIVERSITY OF ALBERTA LIBRARY TO REPRODUCE SINGLE COPIES OF THIS THESIS AND TO LEND OR SELL SUCH COPIES FOR PRIVATE, SCHOLARLY OR SCIENTIFIC RESEARCH PURPOSES ONLY.

THE AUTHOR RESERVES OTHER PUBLICATION RIGHTS, AND NEITHER THE THESIS NOR EXTENSIVE EXTRACTS FROM IT MAY BE PRINTED OR OTHERWISE REPRODUCED WITHOUT THE AUTHOR'S WRITTEN PERMISSION.

(SIGNED) CAO. Yuguang

PERMANENT ADDRESS:

14713-46TH AVENUE
EDMONTON, ALBERTA, T6H 5M6
CANADA

DATE: Oct. 3, 1990

UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

THE UNDERSIGNED CERTIFY THAT THEY HAVE READ, AND RECOMMEND TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH FOR ACCEPTANCE, A THESIS

ENTITLED **COLLISION-FREE MOTION PLANNING AND APPLICATION TO THE PUMA 560 MANIPULATOR**

SUBMITTED BY **YUGUANG (EUGENE) CAO**

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF **MASTER OF SCIENCE.**

V. G. Gourishankar

SUPERVISOR: DR. V.G. GOURISHANKAR

Roger W Toogood

COMMITTEE MEMBER: DR. R.W. TOOGOOD

R. E. Rink

COMMITTEE MEMBER: DR. R.E. RINK

DATE: October 3, 1990

This thesis is dedicated

to my parents Prof. Jinzhou Cao and Prof. Jiajin Zhu,

to my wife Ziping (Jill),

to my brother Dr. Yuming Cao and my sister Yuhong,

to Zhenmin and little Alex, and

to Mr. and Mrs. H. McCombs,

for reasons that they know best.

ABSTRACT

Automatic motion planning of industrial robots in the presence of obstacles is considered to be a difficult task. Many approaches have been proposed in the literature while the basic algorithms may be independent of any particular manipulator, their application however is dependent on the particular kind of manipulator. For this reason, there is still no general purpose path planning technique available today to cover a variety of manipulators. Most of the motion planning algorithms are very complicated and difficult to implement in practical applications, and the process of searching for optimal collision-free paths is often time-consuming.

In this thesis, a new path planning algorithm for collision avoidance in a two-dimensional workspace is proposed. It uses the configuration-space (C-space) concept to shrink the moving object (the robot gripper) to a reference point on the object and enlarge all workspace obstacles to the shape of the object to compensate for this. These C-space obstacles are then enclosed by rectangular hulls, which simplifies the descriptions of both the obstacles and the free space and reduces computational effort required. Then a local search is performed, using Bellman's principle of optimality, in a reduced space defined by the local start and goal points. Such a search requires very little time to find a local minimum-distance path.

This algorithm has been implemented on a PUMA 560 manipulator. Collision-free paths are generated off-line on a supervisory computer and uploaded to the PUMA's Unimate controller.

ACKNOWLEDGEMENTS

The author would like to express his deep and sincere gratitude to his supervisor, Professor V. G. Gourishankar, for his guidance in formatting the objective of this work, and for his perspective and generous support throughout this research project. His enormous patience, understanding and personal advice have most greatly encouraged the author in this and other endeavors.

The author also wishes to thank Dr. R. W. Toogood who was the author's previous supervisor of another project, and Professor R. E. Rink, for their invaluable assistance and direction as part of the author's supervisory committee.

In addition to the above individuals, the author is heavily indebted to Dr. Joel King for his genuine interest and suggestions, for the countless hours he spent in helping the author in so many ways. Dr. King's help is gratefully acknowledged. Thanks to other researchers and graduate students, in particular, Frank Niscak, Philip Pang, Louis Brassard and Lee Kruszewski for their suggestions and help.

The financial support the author received from the Department of Electrical Engineering, University of Alberta, and from the author's friends Mr. & Mrs. H. McCombs is also most gratefully acknowledged. This research would have never been possible without such support.

Finally, the encouragement from the author's parents in China, the forbearance and the sacrifices of the author's wife during the entire period of this research cannot go without mention. The author owes them a debt of gratitude that would never be repaid in full.

TABLE OF CONTENTS

CHAPTER PAGE

1. INTRODUCTION.. 1
    1.1. BACKGROUND AND MOTIVATION.......................... 1
    1.2. OBJECTIVE OF THIS PROJECT AND ORGANIZATION OF THE THESIS.......................... 3
2. COLLISION-FREE MOTION PLANNING.............................. 6
    2.1. DEFINITION OF COLLISION-FREE MOTION PLANNING....... 6
    2.2. CONFIGURATION SPACE (C-space) APPROACH............. 8
    2.3. FREE SPACE DECOMPOSITION (FSD) APPROACH............ 13
    2.4. HIERARCHICAL ORTHOGONAL SPACE (HOS) APPROACH....... 14
    2.5. OCTREE FREE SPACE (OFS) APPROACH................... 19
    2.6. GENERALIZED VORONOI DIAGRAMS (GVD)................. 23
    2.7. ALTERNATE PATH METHOD (APM)........................ 27
    2.8. DISCUSSION AND SUMMARY............................. 30
3. COLLISION-FREE MOTION PLANNING CRITERIA..................... 33
    3.1. INTRODUCTION....................................... 33
    3.2. OVERVIEW OF THE OPTIMAL PATH PLANNING PROBLEM...... 33
        3.2.1. NEAR MINIMUM-TIME MOTION PLANNING............ 35
            3.2.1.1. AVERAGED DYNAMICS METHOD FOR DYNAMICS LINEARIZATION............... 35
            3.2.1.2. MINIMUM-TIME CONTROL................. 37
            3.2.1.3. NEAR MINIMUM TIME CONTROL............ 39
        3.2.2. SHORTEST DISTANCE MOTION PLANNING........... 41

3.3. SUMMARY 45

4. SEARCHING FOR MINIMUM DISTANCE PATHS-A NEW ALGORITHM 46

4.1. INTRODUCTION 46

4.2. SEARCHING FOR AN OPTIMAL PATH IN CARTESIAN SPACE USING BELLMAN'S PRINCIPLE OF OPTIMALITY 48

4.2.1. MODELING OF OBJECTS USING PRIMITIVE SHAPES .. 48

4.2.2. BUILDING A SEARCH GRAPH 51

4.2.3. SEARCHING FOR A MINIMUM DISTANCE PATH WITH HEURISTICS 57

4.2.3.1. THE REDUCED SEARCH SPACE $\Omega(CO_v)$ AND VALID OBSTACLES CO_v AND THEIR VALID VERTICES $VERT_v$ 57

4.2.3.2. THE DEFINITIONS OF mid_s, mid_g, mid_1 AND mid_2 61

4.2.3.3. DESCRIPTION OF THE SEARCH ALGORITHM .. 63

4.3. DISCUSSION AND SUMMARY 67

5. COLLISION-FREE PATH: APPLICATION OF THE ALGORITHM TO THE PUMA 560 MANIPULATOR 73

5.1. INTRODUCTION 73

5.2. BRIEF DESCRIPTION OF THE PUMA 560 ROBOT, ITS WORKSPACE AND VAL II 75

5.3. COMMUNICATION BETWEEN THE SUPERVISOR COMPUTER AND VAL II 78

5.3.1. DIRECT CONTROL OF THE PUMA FROM THE SUN 83

5.3.2. COMMUNICATING FROM A C-APPLICATION PROGRAM USING C-VAL II 84

5.4 FINDING A SAFE PATH FOR THE PUMA 560 AMONG OBSTACLES.. 85

5.4.1. AVOIDING THE LOWER AND UPPER BOUNDS OF THE PUMA WORKSPACE.......................... 85

5.4.2. DESCRIPTION OF THE FIND-PATH ALGORITHM FOR THE PUMA 560............................ 89

5.5. EXAMPLES AND RESULTS................................. 97

5.6. EVALUATION AND DISCUSSION.......................... 99

6. CONCLUSIONS AND SUGGESTIONS...........................116

6.1. SPEEDING UP THE COMMUNICATION BETWEEN THE PUMA AND THE SUN..118

REFERENCES...121

APPENDICES...131

1. DIRECT CONTROL OF THE PUMA FROM THE SUN..............131

2. C LIKE FUNCTIONS OF VAL II (C-VAL II)................132

3. LIST OF ALL PATH PLANNING ROUTINES FOR THE PUMA 560 AND THE "MAKEFILE"...................................136

4. THE FORMAT OF INPUT AND OUTPUT DATA FILES............139

5. A SAMPLE AUTO-START PROGRAM..........................140

LIST OF TABLES

TABLE 4.1. THE MINIMUM-DISTANCE PATH SEARCHING ALGORITHM..... 69

TABLE 5.1. THE MOTION PLANNING ALGORITHM FOR THE PUMA 560.... 91

TABLE 5.2. THE SUBROUTINE FOR GENERATING THE OPTIMAL PATH.... 92

TABLE 5.3. THE SUBROUTINE FOR EXECUTING THE PUMA MOTIONS..... 96

TABLE A4.1. EXAMPLE OF THE INPUT DATA FILE..................139

TABLE A5.1. A SAMPLE AUTO-START PROGRAM STORED IN THE VAL II MEMORY....................................141

LIST OF FIGURES

FIGURE 2.1. THE WORKSPACE Ω AND THE FREESPACE Ω_{free}.......... 8

FIGURE 2.2. THE VISIBILITY GRAPH OF THREE OBSTACLES.......... 12

FIGURE 2.3. A GENERALIZED CONE DEFINED BY SEVEN PARAMETERS L, b_l, b_r, s_l, s_r, m AND c, $m<0$................. 15

FIGURE 2.4. TWO-DIMENSIONAL GRID WORKSPACE.................. 17

FIGURE 2.5. OCTREE CELLS AND THEIR VALUES.................... 21

FIGURE 2.6. A GVD GRAPH OF THREE OBSTACLES.................. 25

FIGURE 2.7. DIFFERENCES BETWEEN A V-GRAPH AND A GVD GRAPH OF THE SAME SET OF THREE OBSTACLES IN THE SAME WORKSPACE........................... 26

FIGURE 2.8. THE ALTERNATE PATH PROPOSED IN [21]............. 29

FIGURE 2.9. THE ALTERNATE PATH TO AVOID THE PUMA TRUNK....... 31

FIGURE 4.1. ENCLOSING OBSTACLES BY RECTANGULAR HULLS......... 47

FIGURE 4.2. (A) ORIGINAL WORKSPACE OBSTACLES................. 52

(B) TRANSFORMED WORKSPACE WITH C-SPACE OBSTACLES. 53

FIGURE 4.3. A SEARCH GRAPH WITH CONNECTION LINES MARKED WITH "*" THAT ARE TO BE ELIMINATED FROM THE GRAPH..... 54

FIGURE 4.4. OBSTACLE SIDE C-D IS A CONNECTION LINE........... 56

FIGURE 4.5. A SEARCH GRAPH WITH COST VALUES ON ALL VALID VERTICES.................................. 58

FIGURE 4.6. ROBOT WORKSPACE AND THE REDUCED SPACE............ 59

FIGURE 4.7. DEFINITIONS OF mid_s, mid_g, mid_1 and mid_2......... 62

FIGURE 4.8. A CASE WHEN ONLY mid_1 and mid_2 EXIST............. 62

FIGURE 4.9. $\mathit{path}\text{-}l^{i}$ WITH MODIFIED MID-SECTION BY $\mathit{path}\text{-}l^{i+1}$... 65

FIGURE 5.1. THE PUMA 560 MANIPULATOR AND ITS WORLD COORDINATE SYSTEM X-Y-Z.......................... 74

FIGURE 5.2. THE WORKSPACE OF THE PUMA 560 DEFINED BY THE UPPER BOUND ENVELOPE R_{max} AND LOWER BOUND ENVELOPE R_{min}.................................. 79

FIGURE 5.3. THE PUMA 560 CONTROLLER AND ITS SUPERVISOR COMPUTER... 81

FIGURE 5.4. FINDING AN ALTERNATE PATH FOR AVOIDING THE PUMA'S LOWER BOUND ENVELOPE R_{min}................. 87

FIGURE 5.5. COLLISION-FREE PATHS GENERATED IN DIFFERENT SITUATIONS......................................103

FIGURE 5.6. DIFFERENT COLLISION-FREE PATHS GENERATED IN THE SAME ENVIRONMENT BUT DIFFERENT START AND GOAL POINTS.....................................106

FIGURE 5.7. A COLLISION-FREE PATH GENERATED IN A VERY CLUTTERED ENVIRONMENT............................108

FIGURE 5.8. A COLLISION-FREE PATH GENERATED IN A LESS CLUTTERED ENVIRONMENT WHEN THE ONLY THING TO AVOID IS THE PUMA TRUNK.......................109

FIGURE 5.9. EXAMPLES OF AVOIDING THE LOWER BOUND ENVELOPE R_{min} 110

FIGURE 5.10. EXAMPLES OF CASES WHEN mid_s AND mid_g ARE REQUIRED TO HELP THE ROBOT FIND A SAFE PATH.....112

FIGURE 5.11. A COMPARISON WITH THE OFS APPROACH..............114

CHAPTER 1. INTRODUCTION

1.1. Background And Motivation

Planning collision-free motions for robotic manipulators in an environment filled with obstacles has been a difficult and interesting research subject since late 70s. The objective of motion planning is to construct algorithms that can transform a high-level goal, such as the specification of a mechanical assembly for a robot manipulator or a target location for an outdoor mobile robot, into sequences of low-level commands to the control computer to accomplish the goal. During the past ten years or so, many motion planning approaches have been proposed in numerous academic papers and research reports. The problem of collision-free motion planning was, probably, first discussed fully and generally in a series of five articles in 1983 and 1984 under the title of "*The Piano Movers' Problem*" (see [66]). The catchy title was chosen to suggest the difficulty of the underlying problem, namely, the navigation of some irregularly shaped three dimensional objects such as pianos and motion of robots among obstacles. These papers led to one approach to robot collision avoidance [12]. Since the work environment of the robot is precisely known, either exact algorithmic bounds can be found for computing a path in this environment, or it could be decided that such a path does not exist. However the algorithms arising out of this approach were all impractical to implement in terms of the amount of computing time required [16]. Nevertheless, there have been steady improvements and significant simplification of motion planning algorithms for special cases of the problem since this initial "Piano Movers'" series of papers. Yap has written a short history in [84] of the motion planning problem up to 1987.

A survey [43] has recently shown that collision-free motion planning is still and will continue to be one of the most active research areas in the field of robotics. This is due to the fact that automatic collision-free path planning of robot movement, operated in a manufacturing environment has become very important in order to avoid the tedious teaching-by-doing (or showing) procedure and to increase productivity. Citing arc welding as an example, the development of practical motion planning algorithms for robot motion for MIG or TIG welding along complicated three-dimensional paths could speed up the operation significantly.

General robotic systems can be divided into two categories: industrial robots (or robot arms) and mobile robots. The first category consists of stationary robots with articulated multilinks that are used for moving objects, handling materials and assembling parts. There are several types of collisions in these applications. For instance, the robot may have to come into physical contact with the object to be moved before the desired force and movement can be applied. When the robot makes the contact, a collision is deemed to have occurred. Therefore, the normal operation of a general-purpose robot may involve a series of such collisions. Another type of collision occurs when the end-effector of a robot travels from its initial location and orientation to the specified goal location and orientation, and encounters obstacles in the work space of the robot. Collisions between some part of the robot and the obstacles may occur before the desired task can be completed. A third type of collision occurs when two or more industrial robots work together in the same work environment to handle complicated and dexterous tasks that only one robot cannot do. In such a case, collision may occur between the robots and obstacles in the common workspace that they share. A severe collision may physically cause damage to a robotic system and its environment. Therefore such a collision should be avoided. It is these types of collisions that a motion planning technique is designed to avoid. Another category of robotic systems consists of autonomous or guided mobile vehicles, called mobile robots, which usually traverse in two dimensions. These robots often encounter obstacles and motion planning is also very important. In this thesis we consider collision free path planning for stationary robots.

Collision-free motion planning is also known as, *collision avoidance*, *find-path problem*, *collision-free path planning*, or simply, *path planning* or *motion planning* [84]. These names will be used interchangeably throughout this thesis.

1.2. Objective Of This Project And Organization Of The Thesis

As mentioned earlier, collision-free motion planning is one of the most important and active research topics in robotics. A great deal of research work has been done in the past decade and continues today. Hundreds of papers have been written, though most of them only deal with problems in two dimensional space, such as mobile robots [33] [39] [40] [41] [54] [73] [75] [85] [88], two or three link planar manipulators [13] [24] [62] [74] [76] [80], and Cartesian robots (the work space may be three dimensional but only translation is involved)

[21] [32] [49]. Some 3-D applications consider only non-cluttered situations, with only one or very few obstacles in the workspace. Many of the existing approaches give only simulation results and do not show how they can be implemented practically on a real industrial robot. Motion planning algorithms for manipulators made up of revolute joints are particularly scarce. Another difficulty is that the application of automatic collision-free path planning algorithms are manipulator-specific. A general-purpose task-level programming technique to cover a variety of robots does not exist now and may still be many years away. Fast collision detection techniques remain difficult to find. Other issues, such as modeling solid shapes and planning optimal collision free paths, are also under development by robotics researchers in many labs and institutions.

In this thesis, collision-free motion planning in Cartesian space of multilink manipulators, such as the PUMA 560 which has six degrees of freedom, will be investigated. This is different from the other more common approaches that deal with joint space path planning, as the problem is fairly easy to handle for a two or three link planar manipulator since the inverse kinematics transformation from Cartesian space configuration to joint space configuration is straight forward. The project will be limited to considering the two dimensional case, since 3-D path planning for multilink robots is a little bit too complex and lengthy for an M.Sc. level project. However, extension to 3-D path planning is also discussed in this thesis.

In Chapter 2, some of the most well-known collision-free motion planning techniques in the literature are reviewed and their advantages, shortcomings, and limitations in practical applications are mentioned. Chapter 3 reviews two major motion planning criteria considered in almost all approaches, (viz.) near-minimum traveling time and shortest traveling distance. Chapter 4 introduces a new collision-free motion planning and path searching algorithm. This algorithm will find the shortest collision-free path in a two-dimensional environment for a circular hand that is shrunk to a point among grown obstacles enclosed by rectangular hulls. This algorithm will reduce significantly the computation of the grown obstacles and the path searching time without wasting much free space. It may be extended directly to applications of three-dimensional motion planning for mobile robots using the articulated cylinder concept [73] to represent the free space, and Cartesian type of robots using an alternative strategy introduced in [49]. In such a case, the 3-D robot hand can be modeled as a sphere [21] [73] and the obstacles can be modeled as tetragonal prisms. Chapter 5 shows the application of this algorithm to a PUMA 560 manipulator in two-dimensional space. The vertical z coordinate is kept constant at a specific value. Chapter 5 also includes the communication between the supervisor computer (SUN 3/160) and VAL II controller. Next, comparisons of the new approach with others are made. The major method compared is the octree-space approach introduced in [32]. In the last chapter of this thesis are some conclusions and suggestions for further research based on the experience gained in the present research project.

CHAPTER 2. COLLISION-FREE MOTION PLANNING

2.1. Definition Of Collision-Free Motion Planning

The task of programming robots for collision-free motion planning is widely recognized as a difficult activity, even in the case of the simplest applications, such as a "pick-and-place" operation among obstacles. This is mainly because [8] that: (a) the find-path problem is not decomposable and (b) it is difficult to plan collision-free motions that involve three-dimensional space rotations. Canny has written a very comprehensive Ph.D thesis [11] to show the complexity of robot motion planning, for which he received the *ACM Doctoral Dissertation Award* of 1987. The **definition of the collision-free motion-planning problem, in the simplest form, is to find an optimal path from a specified starting robot configuration to a specified goal configuration that avoids collisions with a known set of stationary obstacles in the workspace of the robot.** In other words, given a moving object, such as the robot end-effector or gripper, with an initial position (location and orientation), a desired goal position, and a set of obstacles located in the work space, the problem is to find a continuous path or a piecewise linear path for the object from the initial position to the goal position which avoids collisions with all obstacles along the way. Typically, the obstacles and the moving object are modeled using convex polygons (polyhedra) or the union of polygons (polyhedra). In general, the motion planning problem involves two aspects: characterizing the constraints and searching for a solution which satisfies these constraints. This problem [51] is very much different from, and more difficult to handle than, the collision detection problem, i.e. detecting if a known configuration of the robot or if the robot path would cause a collision. Motion planning is also different from real-time (on-line) obstacle avoidance, i.e. modifying a known robot path constantly so that it will avoid unforeseen obstacles real-time.

The *workspace*, Ω, of a robotic manipulator is defined as the set of all three dimensional points that can be reached by a reference point located on the robotic end-effector without violating the physical constraints of the manipulator. The reference coordinate system is called the world frame and is usually located at the base of the robot. In motion planning, Ω is often described by swept volumes [6], cellular arrays [72], octree cells [53], configuration space models [49], hierarchical orthogonal subspaces [82], Voronoi regions [11] [38], and analytic surface equations. If there are obstacles in the workspace, then the end-effector can only traverse in a subspace that is not occupied by any obstacles. This subspace (see Figure 2.1.) is called the *free space*, Ω_{free}, of the manipulator. The description of Ω_{free} may be the same as or different from that of Ω. The techniques of configuration space [49], generalized Voronoi diagram based space [11] [75], generalized cone-shaped space (also called decomposed free space) [7], constrained space [56], artificial (potential) field space [40], and octree free space [19] [31] [32] are also used to describe free space. The find-path problem is actually the search of a reasonably optimal collision-free path in the free space using graph searching algorithms and heuristics under certain criteria.

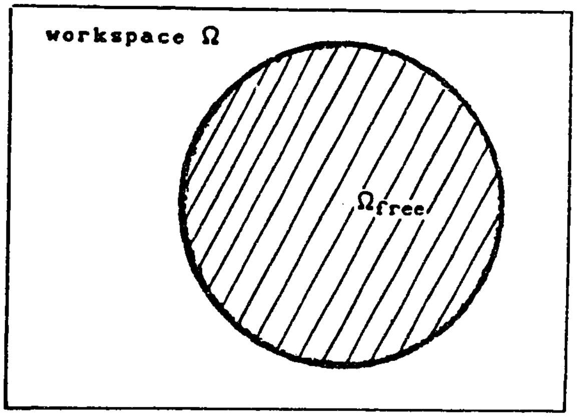


Figure 2.1. The workspace Ω and the freespace Ω_{free}.

2.2. Configuration Space (C-space) Approach

Configuration space (C-space) approach was one of the earliest and most well-known techniques of collision-free motion planning. Configuration space concepts have been a central focus in path planning research for many years. Many of the later developed algorithms are based on this concept [5] [9] [10] [19] [23] [25] [26] [35] [50] [87]. It is a complete method that is ready to be implemented for further applications. C-space approach was first introduced by Udupa in 1977 in his Ph.D dissertation [76]. Udupa formulated the obstacle avoidance problem for a planar 2-link arm in terms of an obstacle transformation which allows shrinking the moving arm by some amount and growing the obstacles by some corresponding amount. He used only rough approximations to the actual C-space obstacles and did not explicitly show how to represent constraints on more than three degrees of freedom.

Lozano-Pérez then furthered Udupa's idea in 1981 [49]. The *configuration* of a moving object A is any set of parameters that will give a complete specification of the position of every point on the object. Configuration space is then the space of all configurations of the moving object. The set of joint angles of a robot manipulator constitute a configuration. Therefore, the robot joint space can form a configuration space, whereas the Cartesian parameters of a robot's end-effector do not usually constitute a configuration because of the non-uniqueness of the inverse kinematics. C-space can be defined [55] as the set of joint configurations $(\theta_1, \theta_2, \ldots, \theta_n)$ that do not violate joint limitations, Θ. If B is an obstacle in the work space Ω of a manipulator, then *obs*(B) is defined as the set of all configurations which will cause collisions of B with any part of the manipulator. We call *obs*(B) the configuration space transform of the obstacle B, (also called *C-space obstacle*). If B_1, B_2, ..., B_m is a complete list of obstacles in Ω, then $\Omega_{free} = \Theta - obs(B_1) - obs(B_2) - \ldots - obs(B_m)$ is the free space portion of the configuration space. We are to determine those parts of the Ω_{free} which a reference point (usually a *reference vertex*, $rv_A \in A$) of the moving object can occupy without colliding with any obstacles. A path is then found for the rv_A through Ω_{free}.

Another way to specify the configuration of a rigid solid polyhedron in $\mathbb{R}^3$ is to define its location and orientation by a single six-dimensional vector

$$\underline{V} = [x\ y\ z\ o\ a\ t]' \qquad (2\text{-}1)$$

where x, y, z, are the coordinate values of a selected point of the polyhedron in $\mathbb{R}^3$ and o, a, t, are the coordinate values of the object's Euler angles [64]. The configuration of a k-dimensional polyhedral object A can be regarded as a point $x \in \mathbb{R}^d$, where $d = k + {}_kC_2$; k parameters are required to specify the location of rv_A in $\mathbb{R}^k$ and ${}_kC_2$ are required to specify the orientation of A. The d-dimensional space of configurations of A is denoted C-space$_A$. A in configuration x is $(A)_x$. The C-space$_A$ obstacle due to B, denoted $CO_A(B)$ is given as:

$$CO_A(B) \equiv \{x \in \text{C-space}_A \mid (A)_x \cap B \neq \emptyset \}, \qquad (2\text{-}2)$$

which means that if $x \in CO_A(B)$ then $(A)_x$ intersects B.

In general, a d-dimensional configuration space can be used to model any system for which the position of every point on the object(s) can be specified with d parameters. Lozano-Pérez designed a method [50] using the properties of convex polygons and polyhedra [2] and the set sum operations to map the obstacles in Ω into its configuration space. These C-space obstacles represent those configurations of the moving object that would cause collisions. Ω_{free} is then defined to be the complement of the C-space obstacles.

Conceptually, we can view Lozano-Pérez's configuration space as shrinking the moving object, a convex polygon or polyhedron, to a point while at the same time expanding the obstacles to the shape of the moving object. Thus, Ω_{free} in Ω can be easily identified. Lozano-Pérez then used the *visibility graph* (V-graph) to represent all the safe paths that can be searched. The V-graph connects those vertices of obstacles together with the start and goal points that can "see" each other, i.e. those vertices that can be connected by a straight line that does not intersect any of the obstacles as illustrated in Figure 2.2. V-graph is a convenient representation of the safe regions because it defines one-dimensional subsets of the configuration space that can be searched using a traditional graph-search method, such as A^* algorithm, to find the shortest path. More details about V-graph searching technique can be found in [48].

The V-graph method always generates a path very close to obstacles which may lead to certain danger of collisions as pointed out in [36] [73]. Also, this approach is computationally expensive, because obtaining C-space obstacles and then mapping them into a configuration space is a very time-consuming process. Each time start and goal points are changed, the V-graph has to be recomputed. The order of computation complexity is $O(n^2)$ [81], where n is the number of vertices. Fortunately it turns out that not all of the n vertices have to be necessarily included in the V-graph. In [86] an algorithm aimed at reducing n by using only "necessary" obstacles in a *subvisibility* graph is given in order to speed up the computation.

A 2-D C-space obstacle $CO_A^{xy}(B)$ can be computed in time $O(u+v)$, when the moving object A is a u-sided convex polygon and the obstacle B is a v-sided convex polygon. For 3-D convex polyhedra A, $B \subseteq \mathbb{R}^3$, each with $O(n)$ vertices, $CO_A^{xyz}(B)$ can be computed in time $O(n^2 \log n)$ [50]. A new method has been recently introduced in [5] for rapid computation of C-space obstacles using the set union property:

$$CO_A(B_1 \cup B_2) \equiv CO_A(B_1) \cup CO_A(B_2) \qquad (2\text{-}3)$$

if there are two obstacles, B_1 and B_2, and

$$CO_A(D) = \bigcup^{k} CO_A(B_i); \qquad D = \bigcup^{k} B_i \qquad (2\text{-}4)$$

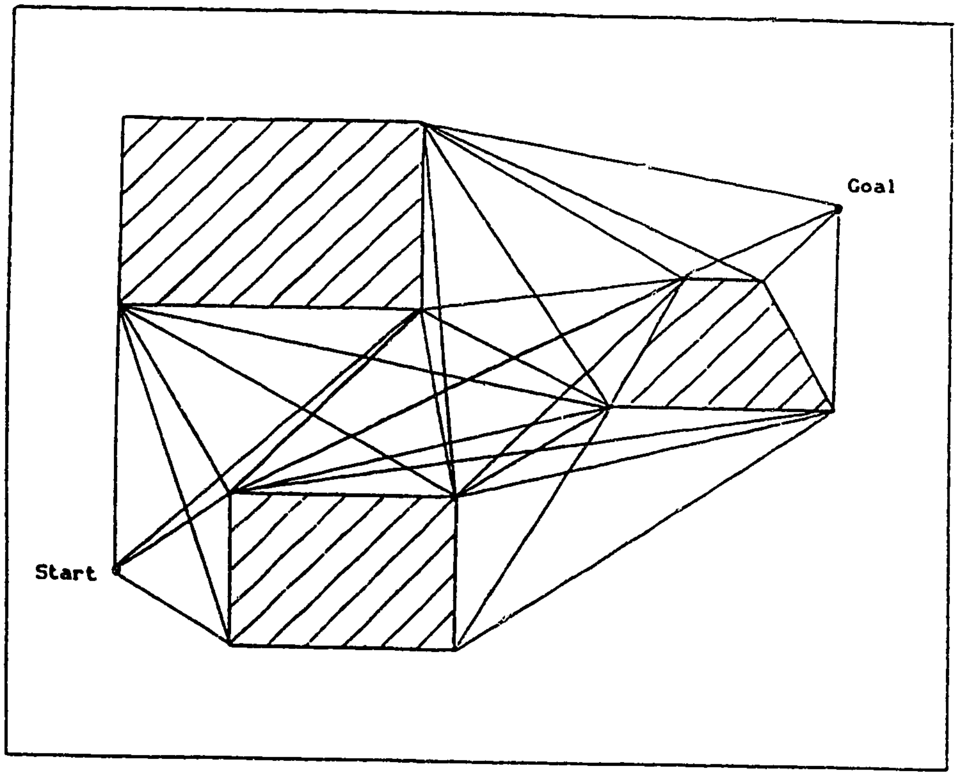


Figure 2.2. The visibility graph of three obstacles.

if there are k obstacles in the manipulator workspace. It suggests that by doing so, the collision avoidance problem is reduced to navigating a point in C-space within the set $-CO_A(D)$. A so-called point translation property is used to solve the transformation problem from Ω to C-space. A summary of the newly proposed methods for searching the V-graph efficiently can be found in [87].

Lozano-Pérez's C-space approach works well when the moving object is not allowed to rotate and can be easily applied to motion planning of Cartesian robots. If a three-dimensional solid is allowed to rotate, a complicated slice projection method [49] has to be used to approximate the high-dimensional C-space obstacles. The computation of C-space would be even more difficult if the manipulator has revolute joints. However, in many pick-and-place operations, we can ignore the rotational degrees of freedom because the rotational resolution is not really required [21]. For instance, in order to simplify the problem the orientation of the gripper can be kept constant throughout the entire pick-and-place operation.

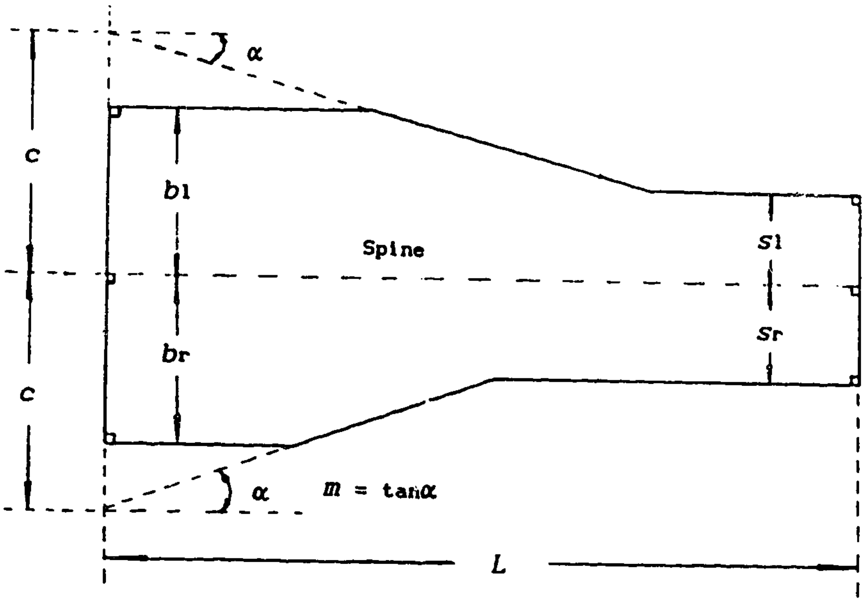
2.3. Free Space Decomposition (FSD) Approach

Instead of determining the corners (vertices) of objects that are visible to each other, Brooks [7] proposed an algorithm that isolates free areas in the form of overlapping union of generalized cones that are descriptions of swept volumes of Ω_{free} in two dimensions. This makes it easier to capture the essential effects of translating and rotating a body through Ω_{free}, which has an advantage over the C-space method. A freeway path is formed between every pair of obstacle sides which face one another. This also includes the sides of the workspace boundary. Each freeway is characterized by a straight spine which runs along its middle and by its width which varies over the range $t \in [0, L]$ and is parameterized along the spine. A freeway is a generalized cone with length L. The maximum radii of this cone away from the spine are b_l and b_r for the left and right sides of the spine at the big end. The minimal radii are s_l and s_r at the small end of the cone. Figure 2.3. shows the seven parameters (L, b_l, b_r, s_l, s_r, m, and c) which completely specify the shape and size of the generalized cone. At any point along the spine, the width defines a range of the orientations of a moving object (which is enclosed by a bounding rectangle aligned with the spine of a concerned cone) for which there is no collision. This leads to the ability to find paths that require rotation maneuvering.

Collision-free path planning is much simpler in two dimensions than in three. For this reason, Brooks [8] suggests looking at 2-D cuts of the work space at a number of 3-D elevations. Brooks' approach works well except that it cannot find possible paths in extremely cluttered situations as there are insufficient generalized cones to provide a rich choice of paths. Because this approach plans motions along the axes of free generalized cones, optimality is therefore lost. This approach also suffers from being computationally expensive, because the generation of 2-D generalized cones needs a computing time complexity of $O(n^3)$, where n is the number of edges to be traversed.

2.4. Hierarchical Orthogonal Space (HOS) Approach

Searching for a collision-free path in a 3-D space can be quite



igure 2.3. A generalized cone defined by seven parameters L, b_l, b_r, s_l, s_r, m, and c, $m<0$.

difficult and time consuming. Some of the existing approaches [8] [82] [83] avoid the pitfalls of trying to extend a 2-D algorithm to 3-D by reducing the 3-D problem into one or more problems in 2-D and solving these 2-D problems using their 2-D algorithms. Similar to Brooks' method of searching 2-D freeway paths at a number of different levels, Wong and Fu [82] introduced a method that applies a variation of the *visibility graph* method [48] to three orthogonal 2-D projections of the 3-D workspace. These three orthogonal projections are the x-y, x-z, and y-z subspaces. This approach adopted the pseudo obstacles concept from [49] to expand the obstacles in the workspace and to shrink the object to be moved to a reference point on the object. Instead of connecting the vertices of the grown obstacles to form the visibility graph, the workspace is divided up in a two-dimensional grid. This grid pattern provides evenly-spaced nodes in each of the three orthogonal 2-D projections (see Figure 2.4). All the nodes of each projection, including those within grown obstacles, are joined to form three visibility graphs; one for each projection. Nodes which are within an obstacle in one projection may be outside in another projection, so they must still be considered in the three-dimensional find-path search.

Breadth-first algorithm is used in this approach to search for the nodes in each of the three visibility graphs. Beginning from the start and goal positions to connect to their nearest 2-D nodes which are called the start and goal nodes, a tree with its children of all 2-D nodes is built. A 2-D node can only be included in the tree once. Different hierarchical level p uses different x-y coordinate. Collision-free path searching based on the breadth-first algorithm is

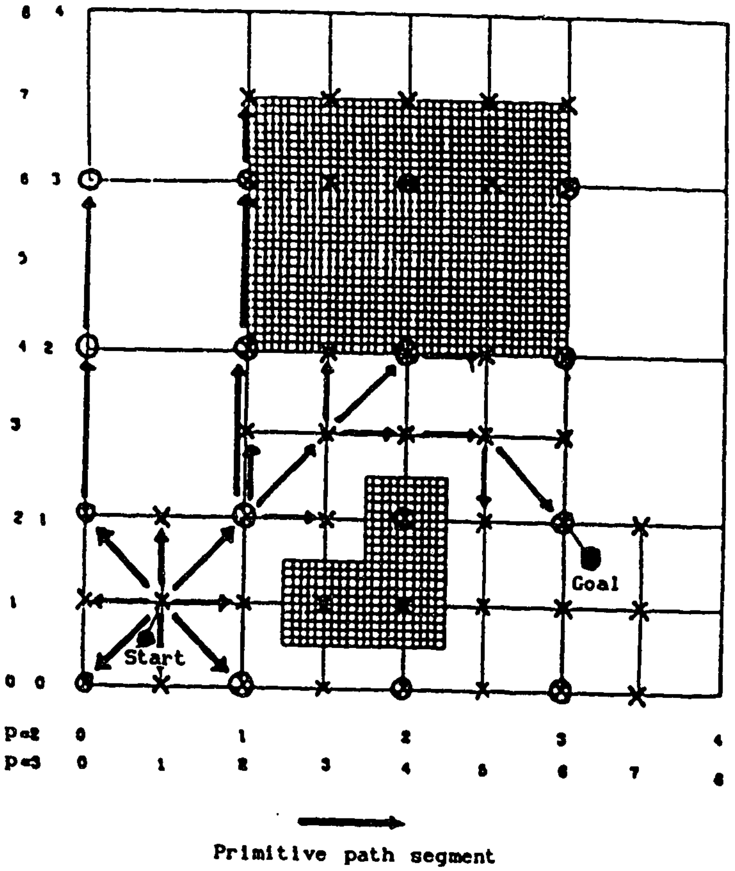


Figure 2.4. Two-dimensional grid workspace. O: 2-D nodes introduced at level p=2. X: 2-D nodes introduced at level p=3.

actually done by connecting the so called primitive path segments. At each step of the search, the results from the three projections are compared. If the primitive path segment is collision-free in any of the three projections then it is collision-free in the three-dimensional model and the search may proceed from that node. Otherwise that primitive path segment is eliminated from the graph and alternate primitive path segments are tested. The search is continued until the goal position is reached. The information from the three orthogonal graphs is combined to give the complete collision-free path in three dimensions.

HOS approach may reduce the amount of path planning time quite significantly compared with the C-space approach, but it cannot generate an optimal (shortest) path. However, it could be argued that the time saved in traveling the shortest Euclidean distance path may not be justified by the long period of time required in planning the path, and so far there has been no efficient algorithms for finding optimal paths among 3-D obstacles. As pointed out in [49], the most important heuristic for a space representation is to avoid excess details on parts of the space that do not affect the operation, which makes this approach particularly useful for planning gross transfer motions of a moving object. Another drawback of HOS is that it is also difficult in practice to use three orthogonal cameras in a cluttered environment to get the three 2-D projections. The occluding effect may be so severe that much of the information will be lost in one or more of the orthogonal subspaces. It is also not clear how HOS can deal with rotations of the moving object and it will be very difficult to implement it to plan motions for manipulators with revolute joints.

2.5. Octree Free Space (OFS) Approach

Octrees (or quadtrees in 2-D cases) have a number of properties and have led to many applications, including cartography, computer graphics, computer vision, robotics , computer aided design, etc.... It is mostly useful in image processing and pattern recognition [53] [65]. Recently, it has been introduced as a means for fast 2-D and 3-D collision-free motion planning [19] [31] [32] [62] [88]. It relies on the representation of the free space by an octree. An octree [53] is a recursive decomposition of a cubic space into subcubic spaces. Initially, the whole space is represented by a single node in the tree, called the root node. If the cubic volume is homogeneous (completely filled by an object or completely empty), then the root is not decomposed at all, and serves as the complete description of the space. Otherwise, it is split into eight equal subcubes (octants), which become the children of the root. Each octant can be recursively divided into octants leading to a tree structure of order eight. Each node of the tree is labeled according to its position with respect to the solid it represents: exterior (EMPTY), interior (FULL), or recursively decomposed (MIXED). This process is continued until all the nodes are homogeneous, or a solution has been found, or it is determined that no solution can be found, or all MIXED cells are smaller than some prespecified size, i.e. until some resolution limit is reached. Because of their regularity, octrees are likely to provide the simplest and fastest algorithms among purely enumerative schemes of motion planning.

Based on the configuration space method in [49], Faverjon [19] proposed a hierarchical description of the free space for the first three links of a manipulator by means of an octree. This provides a rough approximation for planning gross motions of the manipulator, which is believed to be sufficient for most applications. Faverjon did not describe clearly how the fine motion of the hand should be performed. The obstacles are also grown to the size of $CO_A(B)$ as discussed in Section 2.2. As shown in Figure 2.5, a cell of the octree is either free (01), occupied (11) or partially occupied (10) by obstacles. This hierarchical structure of the octree provides the basis for a very efficient search for a collision-free path. However, this algorithm is inherently exponential in the number of degrees of freedom of the robot, which makes it impossible to deal with high dimensional cases, such as robots with many links or redundant robots that are used to perform difficult tasks. Also, Faverjon pointed out in [20] that this method uses a conservative approximation of the free space in the discretized configuration space and hence cannot produce motions in very cluttered environment.

Herman [32] proposed an interesting hybrid path planning technique that uses both hill-climbing and A^* algorithms in searching for a collision-free path in a 3-D octree space which represents the workspace of a Cartesian manipulator. The path searching algorithm works as follows. Beginning at the start point, hill-climbing search is performed. The hill-climbing search uses a cost function whose value at any point (or node) in the free space is proportional to the Euclidean distance from the point to the goal, and whose value at any point inside or on the surface of an obstacle is infinite. This will always allow the robot to move to a neighboring point whose cost is

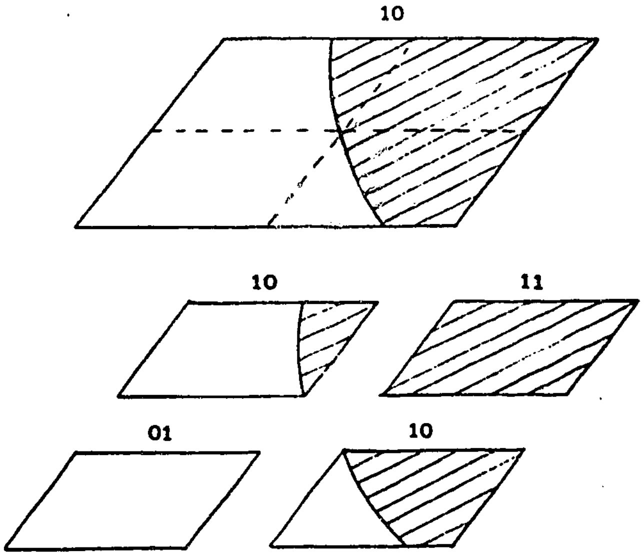


Figure 2.5. Octree cells and their values.

the minimum over other neighboring points. Hill-climbing only requires information local to each robot position and therefore it is very fast in deciding the direction in which the robot should proceed. However, it suffers from a vital problem: the robot can easily get stuck at a local minimum, that is, a point that has a lower cost than any of its neighbors. Once a local minimum is reached, A^* search is invoked, beginning at the point where hill climbing got stuck. A^* performs a global search through the entire graph representation of the octree space. Portions of many different paths may therefore be explored before a solution path is finally found. Hence, A^* is computationally a more expensive search than hill-climbing. The heuristic value ($\hat{h}$) of A^* search in Herman's approach is the Euclidean distance from a point to the goal point. After A^* has got the robot out of the local minimum, hill climbing is invoked again because the robot cannot return to the position where the local minimum occurred.

Although Herman's approach was only a proposal and has not been implemented in a practical operation, it should work well for mobile and Cartesian robots. However, it is difficult to apply it to manipulators with revolute joints, such as the PUMA robot which has a spherical workspace instead of a cubic octree workspace. Just as in the case of the HOS approach we have discussed earlier, this algorithm also cannot find a minimum cost path from the start to the goal point because hill-climbing may often lead the robot away from such a shortest path. It is also difficult to determine a resolution limit of the decomposition of the octree space so that the search for safe paths does not have to be infinite, and that the so called fast, 3-D, collision-free motion planning technique is always superior to others.

2.6. Generalized Voronoi Diagrams (GVD)

The Voronoi diagram has proved to be a useful tool in a variety of contexts in computational geometry. Besides the many uses mentioned in [46] of the Generalized Voronoi Diagram (GVD), it can also be used to represent free space [11] [38] [54] [59] [60] [73] [75]. The Voronoi diagram is often described as a strong deformation retract of free space so that free space can be continuously deformed onto the diagram. This diagram is complete for path planning, i.e. searching the original space for paths can be reduced to a search on the diagram. The Voronoi diagram has dimension one less than the original space which we use to describe the workspace of a manipulator. Reducing the dimension of the set to be searched usually reduces the time complexity of the search. Also the diagram leads to robust paths, i.e. paths that are maximally clear of obstacles (although this will lose the optimality of the path).

A generalized Voronoi diagram is defined to be the locus or the set of points which are equidistant from two or more obstacle boundaries including the workspace boundary. In other words, it is the locus of maximally distant points from the obstacles. The first proposed use of the GVD for motion planning appeared in a Ph.D thesis in 1979 by P.F. Rowat, who used it as a heuristic for motion planning in a digitized robotic workspace [84]. It was then used by Canny [11] for motion planning among a set of obstacles in configuration space called C-Voronoi diagram, and by Takahashi [75] for planning a rectangular mobile object in a 2-D planar workspace with polygonal obstacles.

In [75], a GVD representation of free space is first constructed in order to reduce the search space for finding a collision-free path for the rectangular object. To conduct an efficient search, the GVD is converted to an equivalent *graph* of nodes and arcs. Figure 2.6 is an example of a GVD graph where the small circles represent nodes, and the lines connecting them represent arcs. In the GVD graph there are three types of nodes: (1) *junction node*, is a node where three or more arcs of the GVD intersect. (2) *terminal node*, is a dead end of a GVD arc. Since the workspace itself can be regarded as a hole inside of a large obstacle, the vertices of a rectangular workspace are also terminal nodes. (3) *pseudo-node*, is a source node or goal node. Source and goal nodes are actually artificial nodes that are inserted in the graph near the source and goal positions of the moving object, respectively. Source and goal nodes represent all entry and exit points at which the moving object gets onto and off the GVD graph.

The *arcs* of the GVD graph are the sum of lines connecting pairs of nodes. Each of the arcs consists of a sequence of piecewise linear segments between *via points* which have characterized parameters (x, y, R) where (x, y) represent the coordinates of the via point and R is the radius of the GVD at the via point.

As a convenient way of representing free space, GVD has its similarity with the V-graph method. They all define one-dimensional subsets of the configuration space that can be easily searched. Figure 2.7 illustrates the differences between the GVD and V-graph representations of free space of the same set of obstacles. Once the GVD of the free workspace is obtained, the shortest path on the GVD

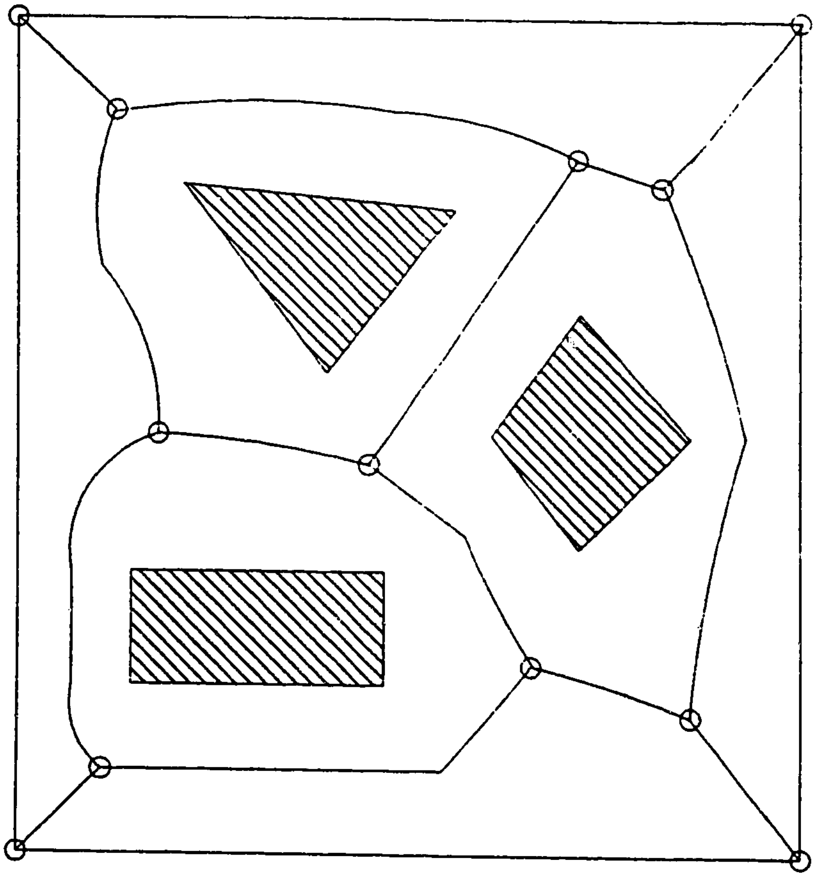


Figure 2.6. A GVD graph of three obstacles.

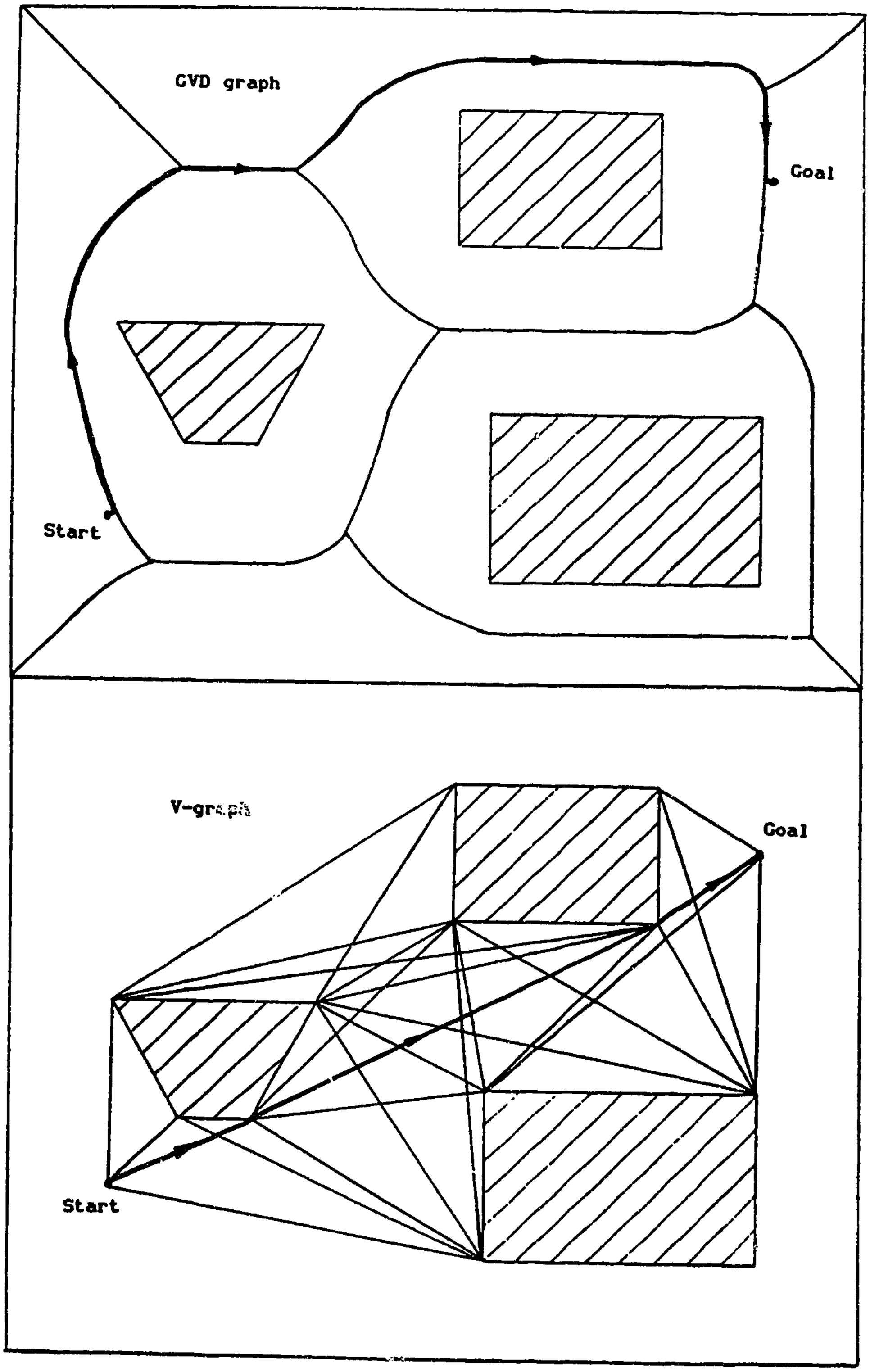


Figure 2.7. Differences between a V-graph and a GVD graph of the same set of three obstacles in the same workspace.

having an adequate radius can be found using graph theory techniques [27]. The motion heuristics will depend upon the size and shape of the moving object and the environment of a specific motion planning case. Takahashi restricted his attention to the case when his mobile object is a rectangle or can be enclosed by a rectangular hull. Like the FSD approach, GVD technique can also handle rotations of the moving object. The final motion is smoother than the FSD method in the sense that it follows parabolic curves, stays far away (sometimes too far away) from the obstacles when possible, and rotates as it translates, not just at isolated points in the workspace. Furthermore, unlike the FSD method, the GVD technique can be applied to difficult motion planning problems which have cluttered workspaces, although generating the GVD graph seems quite time consuming ($O(n^2)$ [75]). As in the FSD method traveling along the straight spine that is in the middle of the generalized cones, GVD method can never find the shortest collision-free path.

Canny [11] has implemented an algorithm for constructing the simplified Voronoi diagram for a 2-D configuration space. An arbitrary moving polygon is allowed to rotate as well as translate amidst polygonal obstacles in such a C-space.

2.7. Alternate Path Method (APM)

There are many motion planning algorithms that can be easily applied to Cartesian robots and mobile robots, such as some of the ones we have reviewed in previous sections. The problem here is simpler than dealing with a manipulator which has revolute joints because the joint space of the whole manipulator corresponds to the configuration space for motion of the end-effector alone and no transformation between Cartesian and joint space is needed.

Fletcher implemented another method for planning a Cartesian manipulator [21] in an M.Sc. project. His algorithm is mainly designed for a gantry type of robot moving in a lightly cluttered environment; only one obstacle is shown in his example. The robot gripper is enclosed by a grown sphere, which makes motion planning easier since the rotations of the hand can be ignored and the only consideration left is the hand's transfer movements among obstacles. To simplify the problem further, all the obstacles are also modeled by enclosing spheres. The algorithm works as follows.

First, the hand moving in a straight line from start to goal is considered as a swept-out cylinder with hemisphere ends (because the hand is modeled as a sphere). Then, the collision detection is simply a test for intersection between a cylinder and a sphere for each obstacle. This is similar to other pseudo obstacle concepts [49] [82], that is, the obstacle spheres are grown by the radius of the hand sphere and shrink the hand to a point. Thus, the swept volume (i.e. the hemisphere-ended cylinder) is now reduced to a straight line. If this line intersects any of the grown spheres, a potential collision has been detected and an alternative path must be proposed. The next thing is to determine an alternative collision-free path [illegible] in Figure 2.8. Two subpaths, $P_s \Rightarrow P_m$ and $P_m \Rightarrow P_g$, form the new collision-free path. Fletcher showed that the two important midpoints [illegible]th obstacle, P_m and P_i^0, can be calculated as:

$$P_i^0 = \Delta P \left[\frac{\Delta P^T (P_i - P_s)}{\Delta P^T \Delta P} \right] + P_s, \text{ where } \Delta P = P_g - P_s$$

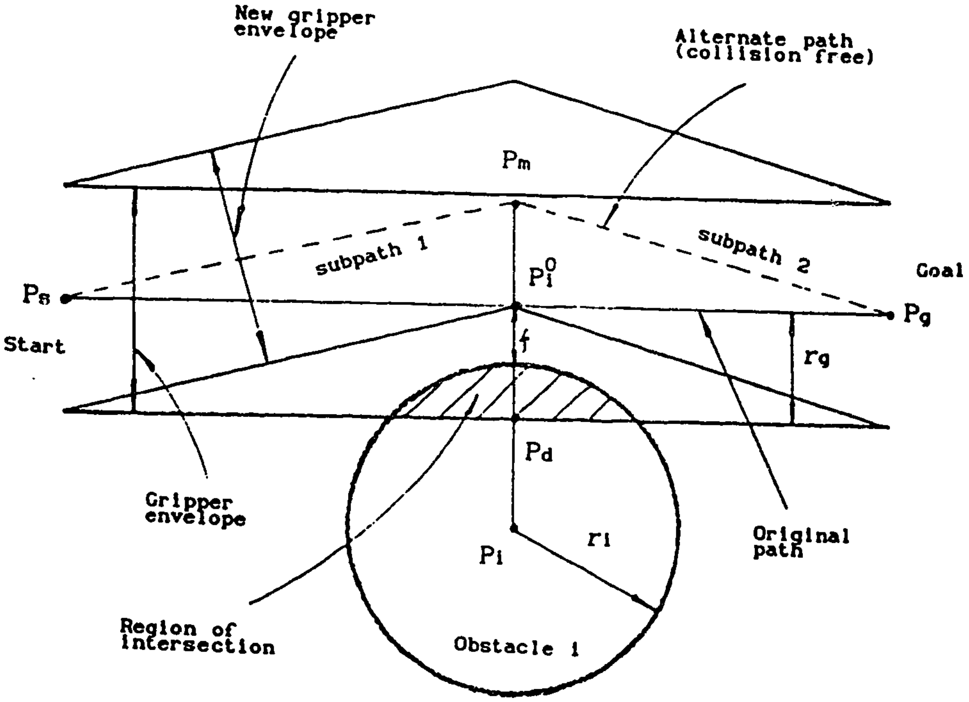


Figure 2.8. The alternate path proposed in [21].

and

$$P_m = r_g + r_i + f \tag{2-6}$$

where r_g and r_i are the radii of the hand and the ith obstacle, and f is a parameter to be determined to ensure the clearance of the hand from the obstacle. This algorithm is fairly simple and easy to implement, but it is not suitable to be used for planning motions in a constrained space which has larger grown obstacles than the grown sphere of the hand. In such a case, we would suggest a better alternate path that consists of the two tangent line segments and the arc between the two tangents of the grown obstacles (dashed line in Figure 2.9), which is what we have used in our algorithm to avoid the lower bound of the PUMA envelope, namely, the trunk of the PUMA robot. This alternate path is always shorter and smoother than the one introduced by Fletcher, and requires very little computation time.

2.8. Discussions And Summary

We have discussed some of the most popular collision-free motion planning algorithms. It can be easily understood that motion planning in a three-dimensional environment for manipulators with revolute joints is a very difficult task. Most of the approaches we have discussed are complete and ready to be implemented mainly on Cartesian (gantry) and/or mobile robots. There is still no general algorithm that will solve the find-path problem efficiently in a three-dimensional space. The collision-free motion planning algorithms which we have seen are different from manipulator to manipulator. The use of a particular approach depends entirely on the specific application and

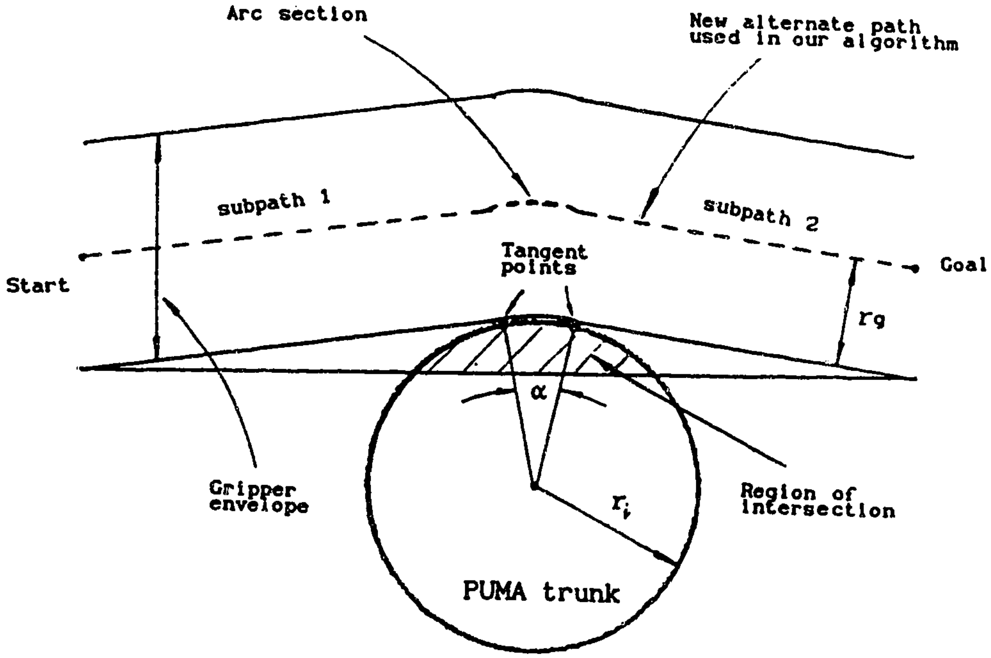


Figure 2.9. The alternate path we propose to avoid the PUMA trunk. It is always shorter and smoother than the one introduced in [21].

the environment. Lozano-Pérez's configuration space approach is well-known and has been adopted by many recently developed motion planning algorithms. Shrinking the robot end-effector to a common point is a general practice for simplifying the problem and allow less computational effort. Searching in a V-graph can often lead to the most optimal path although it is quite time consuming. Octrees and Voronoi diagrams are good substitutes of the C-space description of the robot free space. The search time for a collision-free path is often less than that of the C-space but the path may never be optimal (shortest Euclidean distance).

There are many other less significant approaches in this research subject. They are *artificial potential field* approach by Khatib [40] and Warren [80], *tube concept* approach by Suh and Bishop [74], *intersecting convex shapes* by Singh and Wagh [71], *sequential search* strategy by Gupta [29], *free space characterization* approach by Hasegawa [30], *rectilinear visibility partitioning* method by Jun and Shin [37], *constrained space* method by Muck [56], and so on. They are not as well-known simply because they are new and have not been tried by industrial robots for motion planning purposes. A detailed literature review of algorithmic motion planning in robotics can be found in [67] and [84].

CHAPTER 3. COLLISION-FREE MOTION PLANNING CRITERIA

3.1. Introduction

There are numerous optimization problems in robot motion planning. Most of them consider the following performance criteria: path planning time, travelling time, power consumption, distance to travel, square of velocity, jerk (rate of change of acceleration) of the desired trajectory, and joint torques. Path planning time depends heavily on the motion planning algorithm and the complexity of the robot work environment. The square of velocity is proportional to the kinetic energy. The jerk of the desired trajectory which is the third derivative of position was found recently [45] to adversely affect the efficiency of the control algorithms and therefore it should be minimized. Minimizing the joint torques and jerk will produce a smoothing effect favorable for the joint motors and helps to avoid excitation of elastic vibrations in the robot system.

There have been very few attempts to optimize the robot joint torques when collision-free motion planning is the major concern. This is due to the highly nonlinear nature of the manipulator's dynamics and the difficulty in modeling the dynamics precisely. Maximum speed and acceleration of a manipulator's arm vary with its position, payload, the control algorithm and the robot configuration.

3.2. Overview Of The Optimal Path Planning Problem

Time used for both planning a path and travelling along a specified collision-free path represents the productivity of a manipulator. Determining the path planning time is usually considered as an algorithmic problem in the off-line programming part of motion planning. It depends on the efficiency of the selected path planning technique. However, in the case of travelling time of a robot, it is the on-line path tracking or the robot joint torque control and switching algorithm which is often built into the robot controller that determines the performance of the manipulator. Very little work has been done in terms of a complete *minimum travelling time* solution to the motion planning problem that will also avoid collisions between the manipulator and obstacles in the workspace. Some work has been done on *near minimum-time* or *global optimal time* or *suboptimal time* control for motion planning [13] [18] [69]. To achieve time optimal control of the robot, utilization of the maximum joint torques is inevitably important. In most cases, time optimal control problem is the same as the problem of achieving the optimal travelling time, which may not always yield the shortest distance solution.

For most nonlinear systems, the well-known Taylor expansion or some other techniques can be used to linearize the system dynamics. However, this is very difficult in the case of an *n*-joint robot since there are 2*n* coupled state equations that are non-linear coupled functions of the positions, velocities and payloads. In order to cope with the nonlinearity and joint couplings in the manipulator dynamics, the model parameters of the manipulator are updated at each sampling interval on the basis of feedback information of positions and velocities. Then the *averaged dynamics* [42] concept is used in the design of an optimal controller. Since the switching curves are derived for the current interval and then updated at the next sampling interval with feedback of position and velocity, the approximation error at a sampling instant is implicitly compensated, and the averaging process can effectively handle the nonlinearity and joint couplings in the manipulator dynamics. This technique is both simple and fast. Therefore it can be very useful for collision-free motion planning and even real-time implementations. Next we shall investigate two important optimal motion planning problems: 1) **near minimum-time** problem [18] [42], and 2) **shortest distance** problem [27] [58] [68].

3.2.1 Near Minimum-Time Motion Planning

Planning a near minimum-time motion involves the following considerations [18]: First, the robot end-effector is required to move from an initial to a goal position in minimum travelling time. Second, there are obstacles in the workspace with given geometrical configurations and locations. Third, there are control torque constraints for each joint, $-U_{im} \leq U_i \leq +U_{im}$, where *i* is the *i*th joint.

3.2.1.1. Averaged Dynamics Method For Dynamics Linearization

For a simple *n* joint manipulator, the Lagrange-Euler method [64] can be used to describe its dynamics.

$$\underline{U} = D(\underline{q})\ddot{\underline{q}} + H(\underline{q},\dot{\underline{q}}) + G(\underline{q}) + F \tag{3-1}$$

where $\underline{q}$, $\dot{\underline{q}}$, $\ddot{\underline{q}}$ are $n \times 1$ vectors whose *i*th elements are the *i*th joint variable q_i, joint velocity $\dot{q}_i$, and joint acceleration $\ddot{q}_i$, respectively. $\underline{U}$ is an $n \times 1$ torque vector whose *i*th element is the generalized control torque for joint *i*, U_i. $D(\underline{q})$ is an $n \times n$ symmetric inertial matrix. $H(\underline{q},\dot{\underline{q}})$ is an $n \times 1$ coriolis and centrifugal torque vector. $G(\underline{q})$ is an *nx1* gravitational torque vector. F is an *nx1* static friction vector which is often ignored for noncompliant motion, i.e., free-space motion without the end-effector in contact with any object or no force control on the end-effector.

If we disregard the friction terms, equation (3-1) can be rearranged into *n* equations as

$$\ddot{\underline{q}}_i = \zeta_i(\underline{q})U_i + \xi_i(\underline{q},\dot{\underline{q}},\underline{U}) \qquad \text{for } i = 1,\ 2,\ \ldots,\ n \qquad (3\text{-}2)$$

where

$$\zeta_i(\underline{q}) = D^{-1}_{ii}(\underline{q}),\ \ \xi_i(\underline{q},\dot{\underline{q}},\underline{U}) = \sum_{i=1,\, j\neq i}^{n} D^{-1}_{ij}\, U_j - \sum_{j=1}^{n} D^{-1}_{ij}(H_j + G_j) \qquad (3\text{-}3)$$

and $D^{-1}_{ij}(\underline{q})$ is the $(i,\ j)$th element of $D^{-1}(\underline{q})$. $\xi_i(\cdot)$ represents the coupling effects from other joints, coriolis, centrifugal, and gravitational torques. Both ζ_i and ξ_i are nonlinear functions of the manipulator position, velocity, and input torque.

For the case when ζ_i and ξ_i are time-invariant, the averaged dynamics method [42] can be used to average the behavior of manipulator dynamics by assuming both ζ_i and ξ_i to be constant over one sampling interval, and modifying them at each update time to include the preceding nonlinear dependence on $\underline{q}$, $\dot{\underline{q}}$, and $\underline{U}$. This update is continuous if we know the present and final state information. The result is the following set of uncoupled simpler equations:

$$\ddot{q}_i = \bar{\zeta}_i\, U_i + \bar{\xi}_i \qquad \text{for joint } i, \quad i=1,2,\ldots,n. \qquad (3\text{-}4)$$

$\bar{\zeta}_i$ and $\bar{\xi}_i$ are calculated as follows. (1) compute $\zeta_i(t)$ and $\xi_i(t)$ at time t using equations (3-3) with the current states $q_i(t)$ and $\dot{q}_i(t)$, and the previous input, $U_i(t-\Delta t)$, where Δt is the sampling interval and t is the current time for joint i. (2) compute $\zeta_i(t_f)$ and $\xi_i(t_f)$

using the same method as before but using the final target states. Note that the final state of the manipulator is often known and therefore $\zeta_i(t_f)$ and $\xi_i(t_f)$ can be determined a priori. (3) compute $\bar{\zeta}_i$ and $\bar{\xi}_i$ by the arithmetic average of these values at time t

$$\bar{\zeta}_i = \frac{\zeta_i(t)+\zeta_i(t_f)}{2}, \qquad \bar{\xi}_i = \frac{\xi_i(t)+\xi_i(t_f)}{2}. \tag{3-5}$$

3.2.1.2. Minimum-Time Control (With No Obstacles)

We will first ignore the obstacles in the workspace and apply the averaged dynamics method to our problem to linearize the dynamics of the manipulator. Each joint of the manipulator is driven by an electric motor or by a hydraulic motor, and naturally there exist certain limits to the magnitudes of driving forces or torques, i.e.

$$-U_{im} \le U_i \le +U_{im}, \qquad i = 1, 2, \ldots, n. \tag{3-6}$$

Subject to (3-4) and (3-6), we can compute the input torque U_i^* which moves the system from the initial state to the final target state in minimum time for joint i.

Equation (3-4) can be represented by state variables using a $2n$-dimensional state vector $\underline{X}_i = \begin{bmatrix} X_{1i} \\ X_{2i} \end{bmatrix} = \begin{bmatrix} q_i \\ \dot{q}_i \end{bmatrix}$ as [28]:

$$\dot{X}_{1i} = X_{2i}, \tag{3-7}$$

$$\dot{X}_{2i} = \bar{\zeta}_i U_i + \bar{\xi}_i \tag{3-8}$$

Solving (3-7) and (3-8), we get

$$X_{1i} = (\bar{\zeta}_i U_i + \bar{\xi}_i)t^2/2 + X_{2i}(0)t + X_{1i}(0) \tag{3-9}$$

$$X_{2i} = (\bar{\zeta}_i U_i + \bar{\xi}_i)t + X_{2i}(0), \tag{3-10}$$

Then the ith joint trajectories can be computed as

$$X_{1i} - X_{1i}(0) = \frac{X_{2i}^2 - X_{2i}^2(0)}{2(\bar{\zeta}_i U_i + \bar{\xi}_i)}. \tag{3-11}$$

To achieve minimum-time control, the maximum joint torques at each direction should be applied, i.e., $U_i^* = +U_{im}$, or $U_i^* = -U_{im}$. The switching curves can then be determined as follows [18]:

$$\gamma- : \quad X_{1i} = \frac{1}{2}\left[\frac{X_{2i}^2}{+\bar{\zeta}_i U_{im} + \bar{\xi}_i}\right], \qquad \text{for } U_i^* = +U_{im} \tag{3-12}$$

$$\gamma+ : \quad X_{1i} = \frac{1}{2}\left[\frac{X_{2i}^2}{-\bar{\zeta}_i U_{im} + \bar{\xi}_i}\right], \qquad \text{for } U_i^* = -U_{im} \tag{3-13}$$

In the state plane of (X_{1i}, X_{2i}), $\gamma-$ is equivalent to $X_{2i} \le 0$, $\gamma+$ is equivalent to $X_{2i} \ge 0$. Thus, the minimum-time bang-bang control law is

$$U_i^* = \begin{cases} +U_{im} & \text{if } X_{2i} \le 0 \\ -U_{im} & \text{if } X_{2i} \ge 0 \end{cases} \tag{3-14}$$

And the resulting trajectory for the *i*th joint is

$$X_{1i} = \frac{X_{2i}^2 - X_{2i}^2(0)}{2(+\bar{\zeta}_i U_{im} + \bar{\xi}_i)} + X_{1i}(0), \qquad \text{if } U_i = +U_{im} \tag{3-15}$$

$$X_{1i} = \frac{X_{2i}^2 - X_{2i}^2(0)}{2(-\bar{\zeta}_i U_{im} + \bar{\xi}_i)} + X_{1i}(0), \qquad \text{if } U_i = -U_{im} \tag{3-16}$$

where $X_{1i}(0)$ is the initial joint position and $X_{2i}(0)$ is the initial joint velocity for joint *i*.

Finally, the minimum execution time t_{if}^* for the *i*th joint to transfer the manipulator from the initial state $\underline{X}_i(0)$ to the final state $\underline{X}_i(f)$ (assuming the final state is the origin of the state plane, viz. $\underline{X}_i(f)=\underline{0}$ or $q_i(f)=0$ and $\dot{q}_i(f)=0$) may be determined as [18]:

$$t_{if}^* = \begin{cases} [1/(-\bar{\zeta}_i U_{im}+\bar{\xi}_i) - 1/(\bar{\zeta}_i U_{im}+\bar{\xi}_i)]C^- - X_{2i}(0)/(-\bar{\zeta}_i U_{im}+\bar{\xi}_i), & \text{if } X_{2i} \ge 0 \\ [1/(\bar{\zeta}_i U_{im}+\bar{\xi}_i) - 1/(-\bar{\zeta}_i U_{im}+\bar{\xi}_i)]C^+ - X_{2i}(0)/(\bar{\zeta}_i U_{im}+\bar{\xi}_i), & \text{if } X_{2i} \le 0 \end{cases} \tag{3-17}$$

where

$$C^- = -\sqrt{[2X_{1i}(0) - X_{2i}^2(0)/(-\bar{\zeta}_i U_{im}+\bar{\xi}_i)](\bar{\zeta}_i^2 U_{im}^2 - \bar{\xi}_i^2)/2\bar{\zeta}_i U_{im}}\ ,$$

$$C^{+} = -\sqrt{[2X_{11}(0)-X_{21}^{2}(0)/(\bar{\zeta}_1 U_{1m}+\bar{\xi}_1)](\bar{\xi}_1^{2}-\bar{\zeta}_1^{2}U_{1m}^{2})/2\bar{\zeta}_1 U_{1m}} ,$$

and the initial state is

$$\underline{X}_1(0) = \begin{bmatrix} X_{11}(o) \\ X_{21}(o) \end{bmatrix} = \begin{bmatrix} q_1(o) \\ \dot{q}_1(o) \end{bmatrix} .$$

3.2.1.3. Near Minimum Time Control (With Obstacles)

Now let's introduce obstacles into the manipulator workspace. If the positions (locations and orientations) of the obstacles are known (usually in Cartesian space), they can be transformed into joint space using the inverse kinematics formulation [22] as

$$Obs(x,y,z) \longrightarrow Obs(q_1,q_2,\ldots,q_n). \qquad (3\text{-}18)$$

If $Obs(\underline{q})$ represents the set of joint variables which belong to the obstacles, then $Obs(\underline{q})$ may be considered as geometric constraints of the manipulator, i.e. $Obs(\underline{q})$ is the inadmissible region in the state space. The solution for $obs(\underline{q})$ is usually not unique for manipulators with more than two joints, therefore, $obs(\underline{q})$ should include all solutions for a given $obs(x,y,z)$.

In real applications, to achieve minimum-time control over the entire work space is next to impossible if the manipulator has geometric constraints. However, if we can apply the maximum joint torque throughout the whole collision-free path, it may be reasonable to consider the shortest distance path (minimum norm) to be a so called near minimum-time trajectory. This near minimum-time collision-free path consists of some straight line segments in the joint space that avoid all the inadmissible regions $obs(\underline{q})$. The minimum-time control law introduced above can be used for each segment and it applies the maximum joint torque (3-14) with appropriate sign of switching which can be determined from equation (3-12) and (3-13).

Suppose $tf_{\max}$ is the maximum execution time for all the joints,

$$tf_{\max} = \max(t_{1f},\ t_{2f}, \ldots,\ t_{nf}), \tag{3-19}$$

where t_{if} is the execution time for the ith joint as described in equation (3-17), then the minimum-time for all joints to finish their motions from its initial state X(0) to the final state X(f) at the same instant can be taken as $tf_{\max}$. The trajectory for each joint will have the same shape [18] in the state plane as described in (3-15) and (3-16) with the same execution time $tf_{\max}$. The resulting trajectory of the manipulator in the joint space is then a straight line.

Summarizing the above, the determination of the near minimum-time collision-free control algorithm involves the following steps [18]:

(a) Solve the inverse kinematics problem (3-18) to determine the inadmissible regions *obs*($\underline{q}$). This can be done off-line. Find also the initial and final joint configurations, $\underline{q}(0)$ and $\underline{q}(f)$.

(b) Find (off-line) the near minimum-time collision-free path in the joint space that consists of straight line segments which avoid all obstacles defined by *obs*($\underline{q}$). This trajectory is the same as

(minimum-time trajectory in joint space) - *obs*($\underline{q}$).

If the manipulator travels through *n-1* intermediate points $\underline{q}_k$, for $k=1,2,\ldots,n-1$ in order to avoid *obs*($\underline{q}$), then the collision-free path has *n* straight line segments. For the *k*th line segment, the initial and final joint configurations are $\underline{q}_{(k-1)}$, and $\underline{q}_k$.

(c) For each $\underline{q}_k$, use the average dynamics method in Section 3.2.1.1 to compute $\bar{\zeta}_i$ and $\bar{\xi}_i$, for the ith joint using equation (3-5), starting from the first line segment, $k=1$.

(d) Compute (on-line) t_{if} for the ith joint using (3-17), and $t_{f\max}$ using (3-19). Assume t_{jf} of joint j be $t_{f_{\max}}$.

(e) Find (on-line) U_i^* for the ith joint, where $i \neq j$, by solving equation (3-17) which makes $t_{if}=t_{f\max}$, and $+U_j^*=-U_{jm}$ and $-U_j^*=-U_{jm}$.

(f) Determine (on-line) the switching curves and optimal input torques $\pm U^*$ (we are only to determine the sign of the input) from (3-12), (3-13) and (3-14), and by setting $+U_j^*=-U_{jm}$ and $-U_j^*=-U_{jm}$.

(g) Repeat (on-line) steps c through f until $\underline{q}_k-\underline{q} \le \varepsilon$, a predefined final joint position error, then set $k=k+1$ until $k=n$, and go to step c.

In general this algorithm gives a practical way for time optimal control since it uses the manipulator's full capability (application of the maximum torque). The problem is that it uses a bang-bang control law which makes it impossible to control the intermediate error and it will not work well if there are several obstacles and a more accurate tracking is required. Another disadvantage of this method is that it cannot handle cluttered situations for manipulators with more than two degrees of freedom when the inverse kinematics solutions are not unique and the inadmissible regions of $obs(\underline{q})$ would make it impossible to find a collision-free path of reasonably "near" minimum travelling time from start to goal positions.

3.2.2 Shortest Distance Motion Planning

Planning a collision-free motion of a manipulator is a lot easier if we do not have to take into account its dynamics or we need not control its joint torques. Most of the well known motion planning methods, such as the ones we discussed in Chapter 2, do not consider the time profile of the joint input torques. The only motion planning criterion for selecting optimal paths is the minimization of total Euclidean distance travelled in either Cartesian space or joint space regardless of the behavior of the manipulator's dynamics. If the motion planning is in two dimensional space, the minimum-distance path can be found fairly easily using one of the well-known graph searching method [27] [58] [68]. However, if it is in a higher dimensional space, general solutions of finding a minimum-distance path in the presence of obstacles have not been found. Many efforts have been made to find a reasonably short collision-free path instead of the minimum-distance path [9] [19] [49] [83]. In other words, once the description of the free space is obtained by some specific technique, such as C-space, octree space, decomposed free space, orthogonal projection space, etc., find-path problem would then be reduced to a shortest-path search problem between two nodes in a graph or in a tree as have been introduced in many AI books and papers.

The shortest path problem can be stated as follows:

Given a graph $G = [X, U]$ which is defined [27] as a set X whose elements are called *nodes* (or *vertices*) and a set U whose elements $u \in U$ are ordered pairs of vertices called *arcs* (or *edges*), we associate with each arc $u \in U$ a number $l(u) \in \mathbb{R}$ called the length of the arc ($\mathbb{R}$ is the set of real numbers). We say that G has a value $l(u)$. The shortest path between two nodes i and j is defined as a path $\mu(i,j)$ from i to j whose total Euclidean length is a minimum, i.e.

$$l(\mu) = \min\Big(\sum_{u \in \mu(i,j)} l(u) \Big)$$

This problem has its importance, because $l(u)$ may equally well be considered as being a cost of transportation along u, or the effort involved in u, or the time required to path through u, and so on.

Almost all graph searching strategies can be modeled by the following process [57]:

(1) A *start node* is associated with the initial state description of the manipulator. (2) The successors of a node are calculated using the operators that are applicable to the state description associated with the node. Let Γ be a special operator that calculates *all* of the successors of a node. We shall call the process of applying Γ to a node *expanding* a node. (3) *Pointers* are set up from each successor back to its parent node. These pointers indicate a path back to the start node when a goal node is finally found. (4) The successor nodes are checked to see if they are goal nodes. (i.e., the associated state descriptions are checked to see if they describe goal states.) If a goal node has not yet been found, the process of expanding nodes (and setting up pointers) continues. When a goal node is found, the pointers are traced back to the start to produce a solution path. The state-description operators associated with the arcs of this path are then assembled into a solution sequence.

The steps listed above merely describe the major elements of the search process in a manner similar to the description provided by the flow chart of a nondeterministic program. A complete specification of a search process must describe the order in which the nodes are to be expanded. If we expand the nodes in the order in which they are generated, we have what is called a *breadth-first* process. If we expand the most recently generated nodes first, we have what is called a *depth-first* process. Breadth-first and depth-first methods are actually *blind-search* procedures since the order in which nodes are expanded is unaffected by the location of the goal. The *hierarchical orthogonal space* approach uses breadth-first search algorithm.

If we can predict the cost $\hat{h}(n)$ from any node n to the goal node, it would be more efficient to use this inferred (or heuristic) knowledge to search for a minimum-distance path. Hill-climbing and A^* algorithms are often used in searching a graph. The cost from any location of the manipulator n to the goal can be inferred using the following formula:

$$\hat{h}(n) = |x \text{ coordinate value of goal}-x \text{ coordinate value of point } n| + \\ |y \text{ coordinate value of goal}-y \text{ coordinate value of point } n|$$

Faverjon [19] used A^* algorithm to search the graph of neighbors of his octree in joint space. Herman's [32] octree space searching strategy uses both hill-climbing and A^* algorithms. Lozano-Perez [48] [49] [50] used A^* algorithm to search for the solution path from his V-graph formed by connecting all the C-space obstacle vertices. Brooks [7] [8] [9] also used A^* method to search a collision-free path along his generalized cones which describe the free space in two dimensions.

There are many other algorithms for solving the shortest path problem in a graph. Gondran [27] has given a very good summary of the algorithms by Moore (1957), Dijkstra (1959), Bellman (1958) (1962), Ford (1956) (1962), and Karp (1977).

The find-path algorithm which we will present in Chapter 4 also ignores the dynamics of a manipulator. Our searching strategy uses the well-known Bellman's principle of optimality [1] which is widely used in dynamic programming for solving optimal control problems for nonlinear, time-varying systems. Bellman's method is a type of breadth-first search process [58,p95]. We only use Bellman's principle to search an optimal path locally, i.e., to search a minimum-distance path between each pair of local start and local goal points. We also enhance this technique by the use of heuristics specific to our problem in searching for an optimal path globally.

3.3. Summary

In this chapter we examined two important optimal control strategies in robot motion planning: near minimum travelling time and shortest Euclidean distance. Minimum-time control of a manipulator requires the control of joint torques. When collision-avoidance is added to the minimum-time control problem, it is only possible to get a near minimum-time control solution assuming ζ_i and ξ_i in (3-2) to be time invariant and the robot motion to be non-compliant. The algorithm which we have discussed may be of some use only for simple and less cluttered environment, but may not work in complex situations. In most of the well known motion planning strategies, the robot dynamics are usually ignored. Thus we can use a graph (or tree) searching technique to solve a shortest path planning problem either in two-dimensional Cartesian space or joint variable space. Breadth-first, depth-first, hill-climbing and A^* algorithms are often used in searching such an optimal path. One thing we should note is that the shortest path is not necessarily the minimum-time path for multi-joint manipulators.

CHAPTER 4. SEARCHING FOR MINIMUM DISTANCE PATHS - A NEW ALGORITHM

4.1. Introduction

It was stated in Chapter 3 that searching for the shortest path without considering the dynamics of a manipulator is a practical way for solving the find-path problem. In this chapter we will propose a new algorithm for this case. This algorithm has been implemented on a PUMA type robot to find a collision-free path with minimum Euclidean distance criterion in a two-dimensional Cartesian workspace. The details of implementation are described in Chapter 5. Obstacles are modeled as or enclosed by rectangular hulls all with the same orientation (Figure 4.1.), which will reduce significantly the path searching time without wasting much free space. This is different from other approaches where the contour of the obstacles are followed. The C-space [49] (or pseudo [83]) obstacle idea is used to shrink the moving object (the robot gripper) to a reference point on the object and grow or enlarge the obstacles to the shape of the object. We use Bellman's principle of optimality to search for a graph similar to the V-graph in [48]. We also introduce the concept of *reduced space* defined first by the global start and goal points. The search is from the outmost pair of obstacles within the *reduced space*, and it is continued from the outermost pair of obstacles within an updated new reduced space of the local start and goal points as determined by the previous pair of obstacles. This search process goes on inwards until no more obstacles are found. Such a procedure turns out to be simpler and faster than a blind search of the entire workspace of the robot.

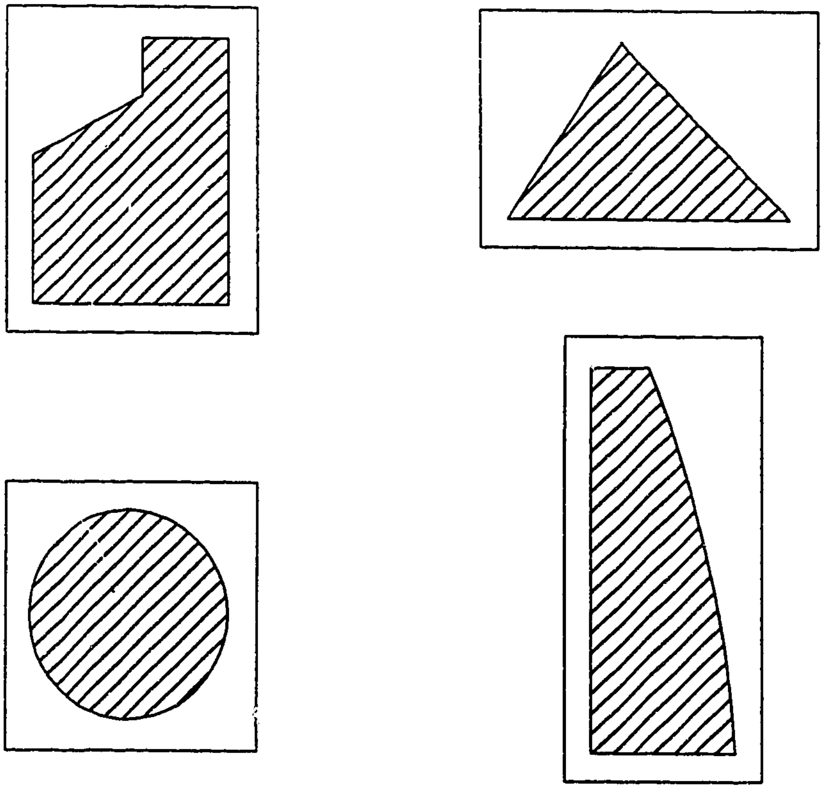


Figure 4.1. Enclosing obstacles by rectangular hulls.

4.2. Searching For An Optimal Path In Cartesian Space Using Bellman's Principle Of Optimality

An important part of off-line motion planning is the generation of a safe path. In order to search for an optimal collision-free path, we need to perform the following operations: modeling the moving object and obstacles to simplify the description of the robot environment, building a search (free) space in a graph, and finally searching for a safe path in the graph efficiently. Since the initial and final configurations of the robot hand and obstacles are often given in Cartesian space, all of the above should be done in Cartesian space to reduce the complexity of our problem. In order to achieve this, the manipulator should be able to handle its own kinematics computations, which is possible for almost all industrial robots. Motion planning in joint space is simple if the robot has only two or three links and the inverse kinematics has only unique solutions, but the task can be very difficult if more degrees of freedom have to be considered. Also the mapping of the robot and obstacle configurations from Cartesian space to joint space could be time consuming.

4.2.1. Modeling Of Objects Using Primitive Shapes

Interference detection is a means of detecting potential collisions of an object with others. In the case of planning motions for a manipulator, preventing potential collision means that a non-zero distance is maintained between the links of the manipulator and the obstacles. Usually we choose from a set of simple shapes called the primitive shapes to model solid objects such as the robot hand and the obstacles in the workspace. Primitive shapes include points, lines, rectangles, squares, circles, spheres, (right) prisms, tetragonal prisms, ellipsoids, cones, cylinders, cylspheres, parallelepipeds, cuboids, etc. Because they are simple to compute, building a free space representation will require less time. This approach can reduce significantly the computation time for collision detection.

A common way to approximate the irregularly shaped robot hand is to use a sphere or hemisphere. Thus we can easily apply the C-space or pseudo obstacle idea in [49] [83] by shrinking the robot hand to a common point which is the center of the sphere. In a two-dimensional case, the sphere is a circle, and the common point is the center of the circle. The obstacles are grown by the radius of the circle.

In a general case, C-space obstacles often have different orientations from the ones we have shown in Figure 4.1. If A is the moving object (in our case, the robot hand) with fixed orientation and B is the set of obstacles, A and B are both n-sided convex polygons, then C-space obstacles $CO_A^{xy}(B)$ in (x,y) coordinate of $\mathbb{R}^2$ can be proved to be the following set difference [50]

$$CO_A^{xy}(B) = B \ominus (A)_o \qquad (4\text{-}1)$$

using the set properties:

$$A \oplus B = \{a + b \mid a \in A,\ b \in B\}$$

$$A \ominus B = \{a - b \mid a \in A,\ b \in B\}$$

$$\ominus A = \{-a \mid a \in A\}$$

$(A)_o$ is the initial configuration of the moving object A. The *supporting lines*, $\pi(A,u)$ and $\pi(B,u)$, have the property [2, p37 & p90],

$$\pi(A \oplus B, u) \cap (A \oplus B) = (\pi(A,u) \cap A) \oplus (\pi(B,u) \cap B) \qquad (4\text{-}2)$$

where u is the outward arbitrary unit normal of A and B. All of A, B and $A \oplus B$ are enclosed by half spaces bounded by $\pi(A,u)$, $\pi(B,u)$ and $\pi(A \oplus B, u)$ respectively. $\pi(A,u) \cap A$ is the set of boundary points of A on the supporting line $\pi(A,u)$, similarly $\pi(B,u) \cap B$ is the set of boundary points of B on $\pi(B,u)$. $\pi(A \oplus B, u) \cap (A \oplus B)$ is that of $A \oplus B$ on $\pi(A \oplus B, u)$. If A is a u-gon convex and B is a v-gon convex, the C-space obstacle $CO_A^{xy}(B)$ requires $O(u+v)$ computation time [50].

From the above we can see that reducing the number of sides of a polygonal object will reduce the computation time for translating the object to a C-space obstacle. Since there are usually many obstacles in the robot workspace, this can save a lot of path planning time in a general sense. In real applications, obstacles are modeled as simply as possible. [21] used spheres (or circles in 2-D case) to approximate obstacles. Spheres (or circles) may not model long and thin tetragonal prisms (or rectangle) obstacles properly and may waste a lot of free space. Enclosing all the obstacles by rectangular hulls as shown in Figure 4.1, makes it very convenient to build the V-graph and to search an optimal path in the graph without losing much free space. Most obstacles, compared with the robot workspace, are much smaller. Therefore losing some free space in order to simplify the find-path problem and hence speed up the computation is better than an accurate description of all the obstacles in the configuration space. This supports the idea that modeling our obstacles with rectangles or rectangular hulls will suffice.

If we use a circle to approximate the robot hand and then shrink it to a reference point which is the center of the circle, C-space obstacles, $CO_A^{xy}(B)$, (see Figure 4.2.) can be easily computed by enlarging each side of the rectangle by *2r*. That is for the *i*th C-space obstacle, $CO_A^{xy}(B_i)$, we have

$$X_{ci} = X_i + 2r \text{ and } Y_{ci} = Y_i + 2r \tag{4-3}$$

where X_i and Y_i are the widths of the *i*th obstacle in (X, Y) coordinate and r is the radius of the circle. If k is the number of obstacles in the workspace, then

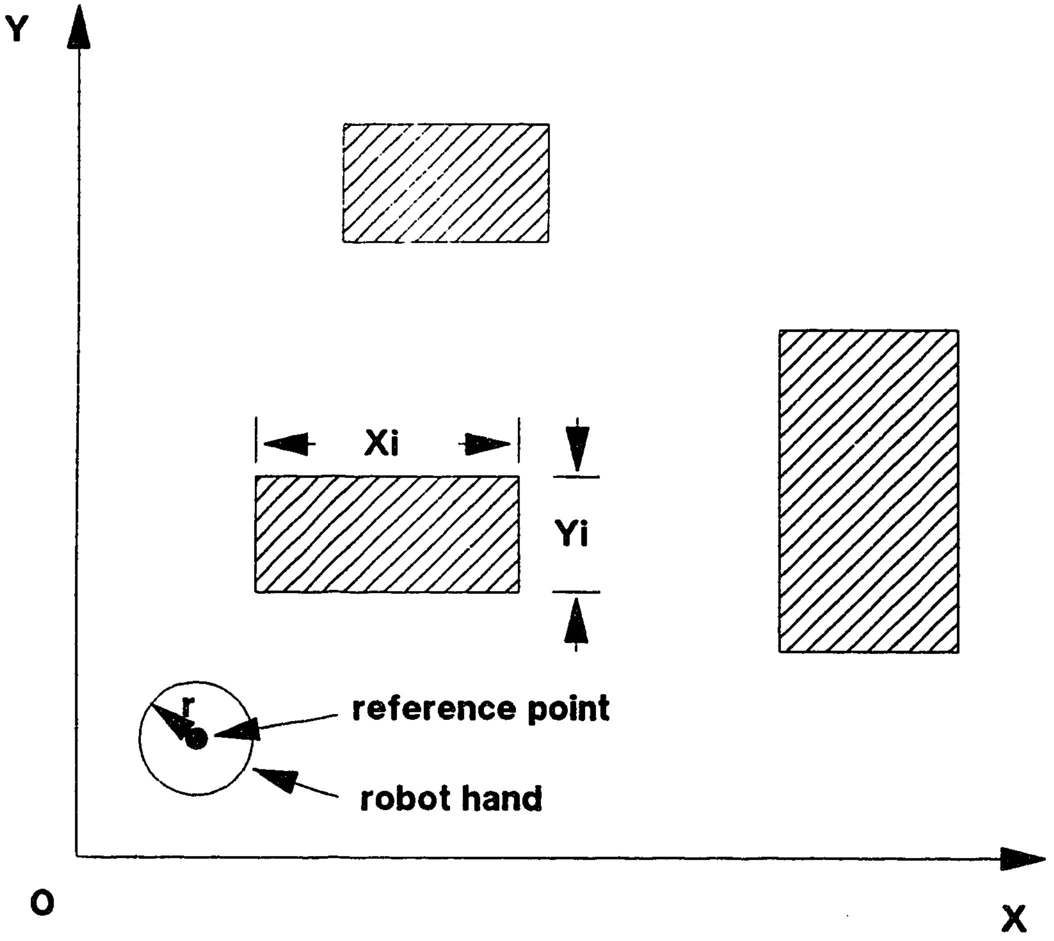
$$CO_A^{xy}(B) = \bigcup_{i=1}^{k} CO_A^{xy}(B_i) \tag{4-4}$$

Another advantage of using a circle to model the moving object is that rotations of the object are effectively decoupled from translational movements and as a result the motion planning problem is simplified.

4.2.2. Building A Search Graph

The way we build our free-space search graph that represents all safe paths is similar to the *visibility graph* (V-graph) method. That is, we connect those vertices of all obstacles that can "*see*" each other. This is same as connecting those vertices by a straight line, (called *connection line*,) that does not penetrate into any of the obstacles.

However, our search graph differs from the V-graph in that it eliminates the connection lines to a vertex that is between two other vertices with connection lines on the same obstacle, because the length of one side of a triangle is always less than the sum of the length of the other two sides. As shown in Figure 4.3., connection lines marked with "*" are to be eliminated from the search graph. In



(a)

Figure 4.2. (a) Original workspace obstacles.
(b) Transformed workspace with C-space obstacles.

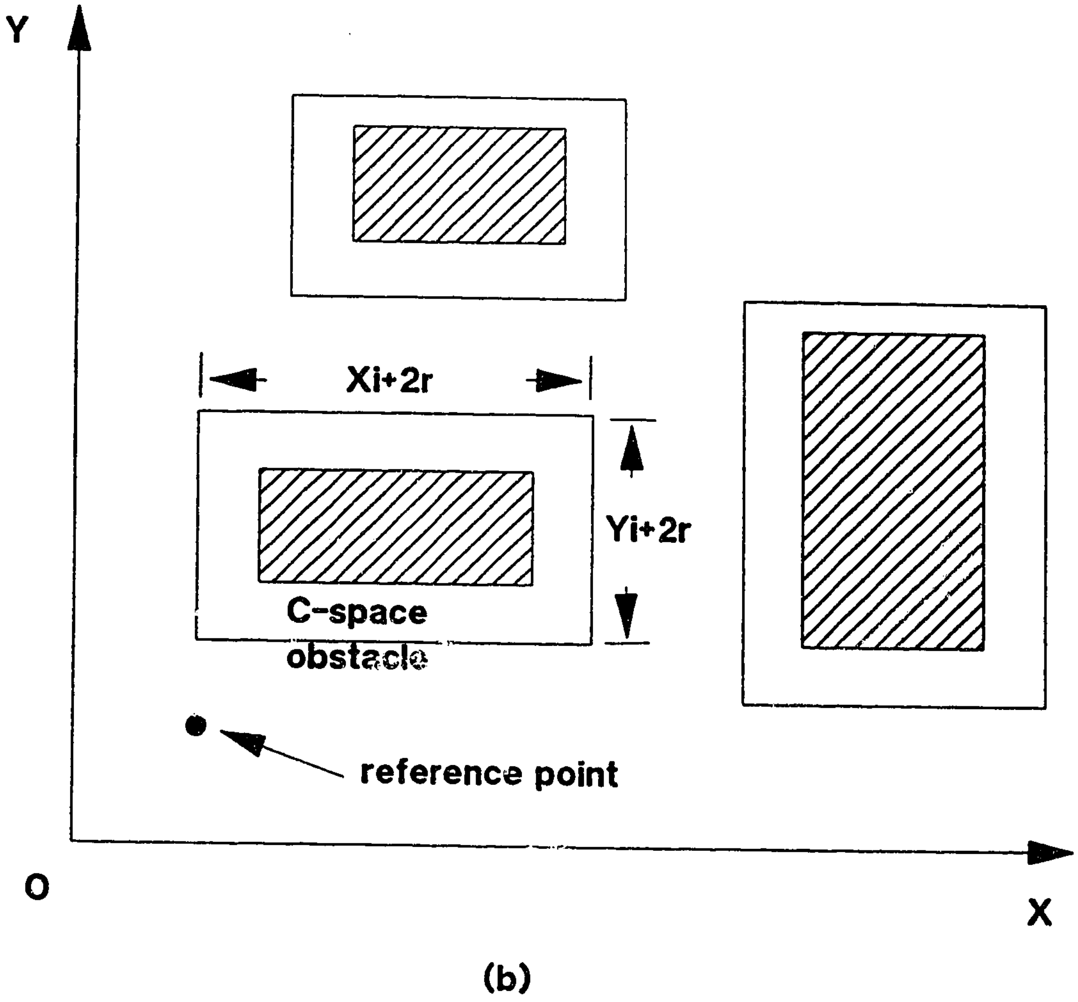


Figure 4.2. (Continued.)

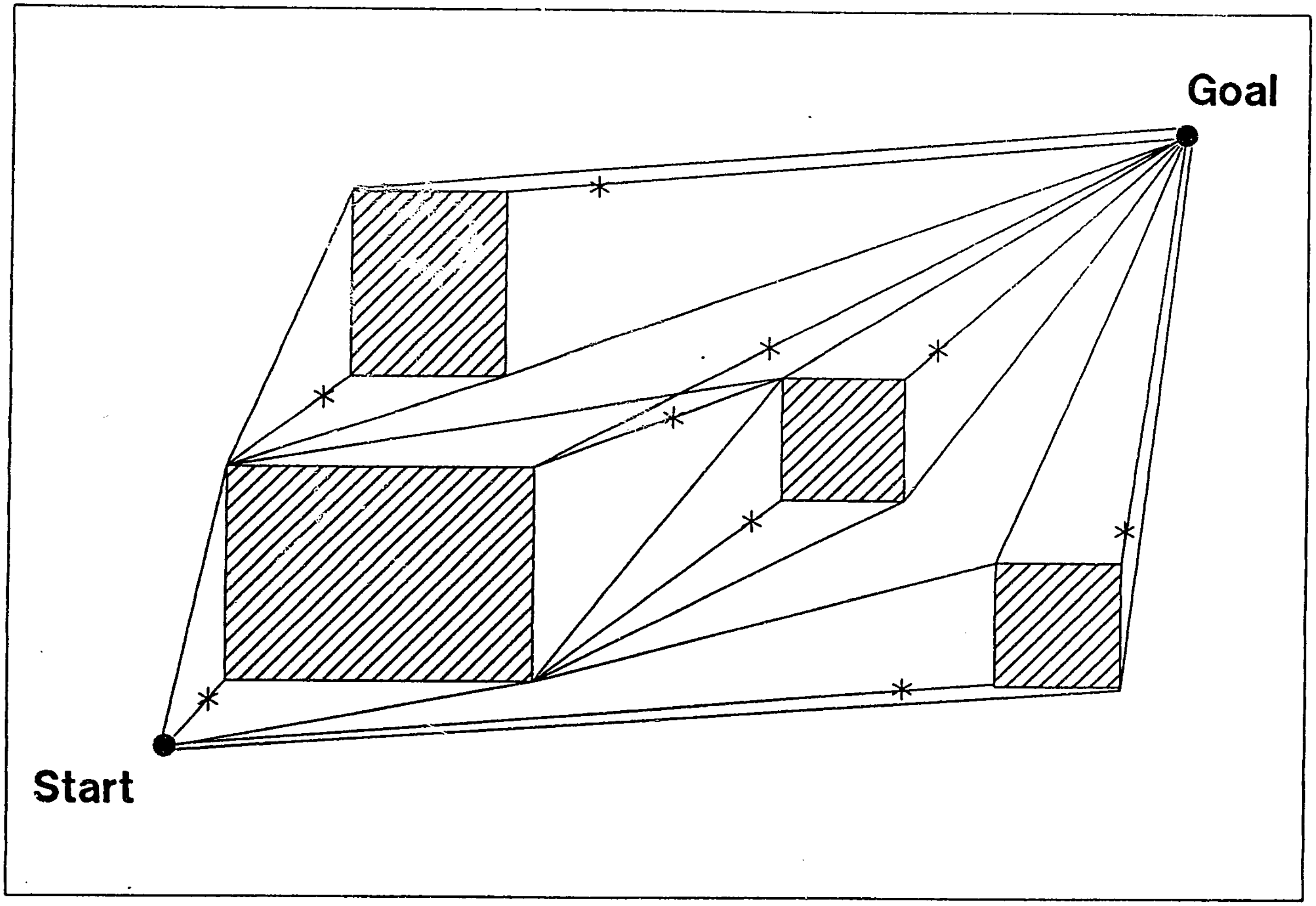


Figure 4.3. A search graph with connection lines marked with "*" that are to be eliminated from the graph.

order to do an optimal search in this graph, it is essential that the computer program keeps track of the traveling direction from the start point to the goal point. The direction is either NE-SW/SW-NE-ward or SE-NW/NW-SE-ward, where NE, SW, SE, and NW denote northeast, southwest, southeast and northwest directions respectively.

If the traveling direction is NE-SW-ward or SW-NE-ward, then it has a slope greater than zero and therefore all of the southeast and northwest vertices of all obstacles will be identified as *valid vertices*, denoted as $VERT_{v+}$, that may be connected by straight lines if they can "see" each other. If the traveling direction is SE-NW-ward or NW-SE-ward, then it has a slope less than zero and therefore all of the southeast and northwest vertices will be identified as *valid vertices*, denoted as $VERT_{v-}$, that may be connected. Valid vertices, $VERT_v$, may either be $VERT_{v+}$ or $VERT_{v-}$ but cannot be both, i.e.

$$\begin{aligned} VERT_v &= VERT_{v+}, \quad \text{if slope of } \textit{start-goal} \geq 0 \\ VERT_v &= VERT_{v-}, \quad \text{if slope of } \textit{start-goal} < 0 \end{aligned} \qquad (4\text{-}5)$$

A small value (f=0.00001 in our program) will be automatically added to X_s of the *start* point (X_s, Y_s) to avoid 90 degree slopes that cause numerical computation problems. The four sides of each obstacle are also qualified as the candidates of connection lines. Figure 4.4. shows such an example where

$$VERT_v = VERT_{v+} = \{\ a,\ d\ \}$$

but the connection line is c-d. This will be determined by heuristics in our algorithm.

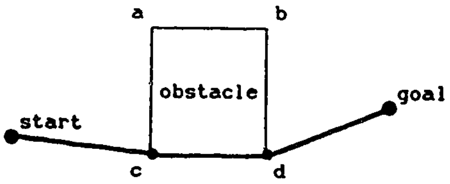


Figure 4.4. Obstacle side c-d is a connection line.

4.2.3. Searching For A Minimum Distance Path With Heuristics

The next step in the procedure is to find an optimal path, given a search graph. We will use the well-known Bellman's principle of optimality [1] which is used in dynamic programming for solving optimal control problems.

Bellman's principle of optimality is as follows:

> An optimal policy has the property that no matter what the previous decisions have been, the remaining decisions must constitute an optimal policy with regard to the state resulting from those previous decisions.

This means that in order to find the optimal path, we must work backward, i.e., the search should begin from the goal point to the start point. The locations of start and goal points, and all obstacles and their sizes are supposed to be known before we start our path planning process. Thus the cost which is the Euclidean distance between each valid vertex, $(VERT_v)_j$, and the goal can be calculated, and it is marked on the vertex (Figure 4.5). The minimum-distance path is then found as the dashed line using the principle of optimality.

4.2.3.1. The Reduced Search Space $\Omega(CO_v)$ And Valid Obstacles CO_v And Their Valid Vertices $VERT_v$

It often happens that some of the obstacles in the workspace, Ω, are not in the space enclosed by the global start point, $start^0$, and global goal point, $goal^0$, as shown in Figure 4.6. In such cases, building a search graph and performing an optimal path search in the graph can be done in this *reduced space* (denoted as $\Omega^0(CO)$ where CO

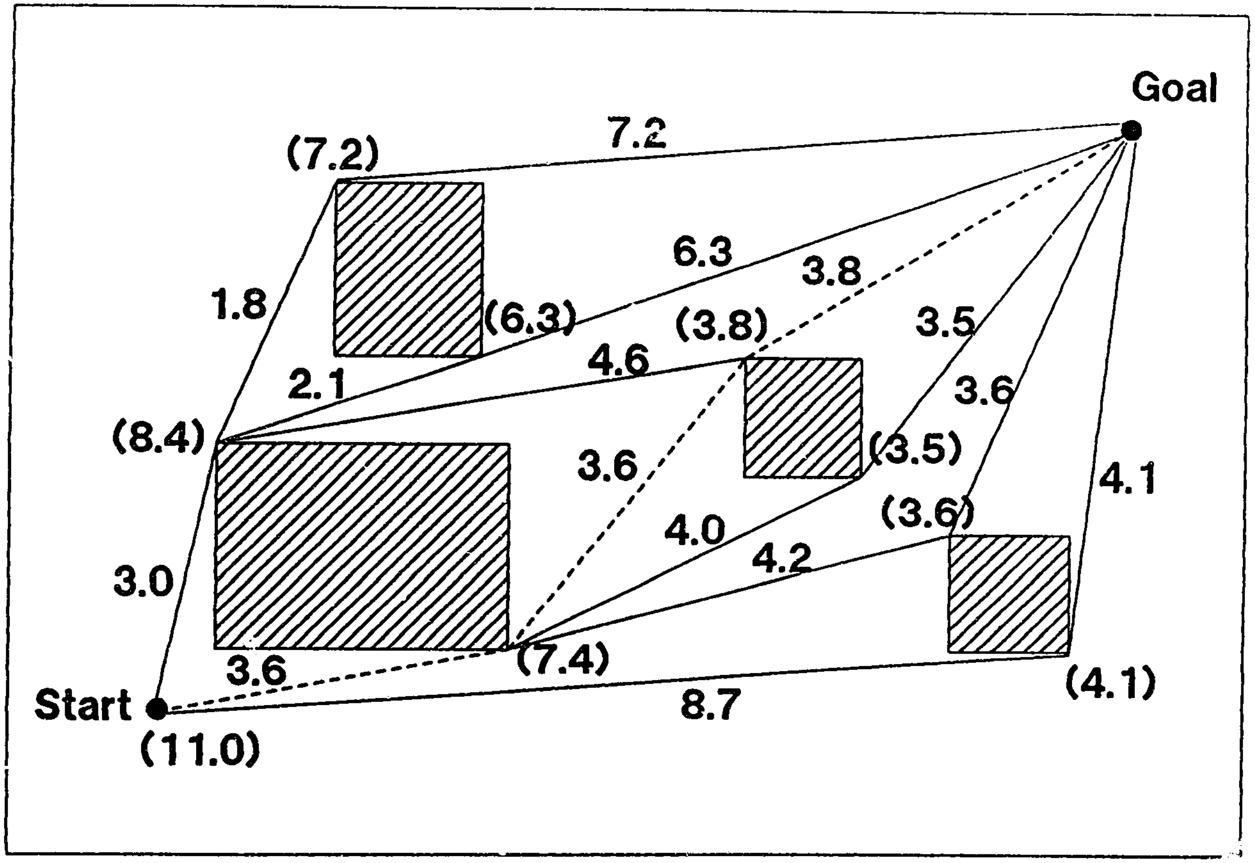


Figure 4.5. A search graph with cost values on all valid vertices.

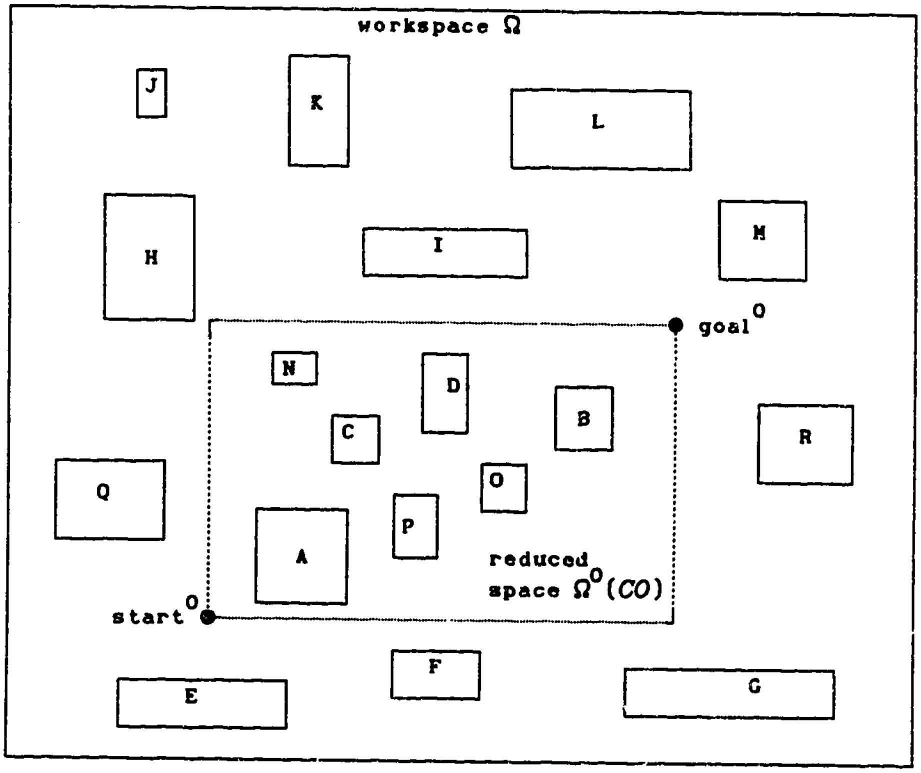


Figure 4.6. Robot workspace and the reduced space.

stands for C-space obstacles, and $\Omega^0(CO) \subseteq \Omega$), thus increasing the efficiency of our path planning algorithm. Note that all the obstacles outside of $\Omega^0(CO)$ are ignored.

To simplify the problem further, we build the search graph and perform the optimal path search only among the outermost pair of obstacles within $\Omega^0(CO)$ defined by $start^0$ and $goal^0$ points. Obstacle A and B in Figure 4.6 form the outermost pair of obstacles in $\Omega^0(CO)$. They will be labelled $CO_v^1 = CO_v^1(S) \cup CO_v^1(G)$ as the first pair of *valid obstacles*. The optimal path found among CO_v^1 in $\Omega^0(CO)$ is labelled $path\text{-}l^1$ as the first level *local optimal path*.

In the general case for the *i*th level of the search space we have

$$CO_v^{i+1} = CO_v^{i+1}(S) \cup CO_v^{i+1}(G), \quad \forall CO_v^{i+1}(S) \in \Omega^i(CO_v^i),\ CO_v^{i+1}(G) \in \Omega^i(CO_v^i),$$

$$CO_v^i(S) \cap CO_v^i(G) = \emptyset, \quad i = 0, 1, 2, \ldots$$

and

$$\Omega(CO_v) = \bigcup_{i=1}^{n} \Omega^i(CO_v^i), \qquad CO_v = \bigcup_{i=0}^{n} CO_v^{i+1}, \tag{4-6}$$

where *n* is the last layer of search in the worst case determined by the complexity of the robot work environment, $i=0$ defines the subspace $\Omega^0(CO)$ enclosed by global start and goal points, $start^0$ *and* $goal^0$, $i>0$ is the *i*th layer inward from $start^0$ *and* $goal^0$ and $CO_v^{i+1}(S)$ is the valid obstacle with valid vertices $VERT_v^{i+1}(S)$ closest to $start^i$ in $\Omega^i(CO_v^i)$ and $CO_v^{i+1}(G)$ is the valid obstacle with $VERT_v^{i+1}(G)$ closest to $goal^i$, and they do not overlap. Also note

$$VERT_v^{i+1} = VERT_v^{i+1}(S) \cup VERT_v^{i+1}(G), \quad VERT_v^{i+1}(S) \in CO_v^{i+1}(S), \tag{4-7}$$

$$VERT_v^{i+1}(G) \in CO_v^{i+1}(G).$$

where $VERT_v^{i+1}(S)$ and $VERT_v^{i+1}(G)$ can be determined by (4-5). $start^i$, $goal^i$, $VERT_v^{i+1}(S)$ and $VERT_v^{i+1}(G)$ are actually the nodes that form the search graph in $\Omega^i(CO_v^i)$. It is very efficient to find an optimal path in such a simplified search graph using Bellman's principle for at most only six nodes exist in the graph.

4.2.3.2. The Definitions Of mid_s, mid_g, mid_1, And mid_2

Before we continue our discussion on the optimal path searching algorithm, some additional information of determining the next search space $\Omega^{i+1}(CO_v^{i+1})$ should be given if more obstacles are found within a space enclosed by mid_1^{i+1} and mid_2^{i+1} in $\Omega^i(CO_v^i)$, where $i=0, 1, 2, \ldots$.

Figure 4.7. illustrates the definitions of points mid_s^{i+1}, mid_g^{i+1}, mid_1^{i+1}, and mid_2^{i+1} in search space $\Omega^i(CO_v^i)$. They are

$$mid_1^{i+1} \in VERT_v^{i+1}(S), \qquad mid_2^{i+1} \in VERT_v^{i+1}(G), \tag{4-8}$$

$$mid_s^{i+1} \in \begin{cases} VERT_{v-}^{i+1}(S), & \text{if } mid_1^{i+1} \in VERT_{v+}^{i+1}(S) \\ VERT_{v+}^{i+1}(S), & \text{if } mid_1^{i+1} \in VERT_{v-}^{i+1}(S) \end{cases} \tag{4-9}$$

$$mid_g^{i+1} \in \begin{cases} VERT_{v-}^{i+1}(G), & \text{if } mid_2^{i+1} \in VERT_{v+}^{i+1}(G) \\ VERT_{v+}^{i+1}(G), & \text{if } mid_2^{i+1} \in VERT_{v-}^{i+1}(G) \end{cases} \tag{4-10}$$

In most cases, mid_s^{i+1} is the invalid vertex on $CO_v^{i+1}(S)$ closest to $start^i$, and mid_g^{i+1} is the invalid vertex on $CO_v^{i+1}(G)$ closest to $goal^i$. Figure 4.8. shows a simpler case when only mid_1^{i+1} and mid_2^{i+1} exist in $\Omega^i(CO_v^i)$. There are also cases when only mid_s^{i+1}, mid_1^{i+1} and mid_2^{i+1} exist, and when only mid_g^{i+1}, mid_1^{i+1} and mid_2^{i+1} exist in $\Omega^i(CO_v^i)$. mid_s^{i+1}, mid_g^{i+1}, mid_1^{i+1}, and mid_2^{i+1} are important parameters. They help us build $\Omega^{i+1}(CO_v^{i+1})$ and the search graph associated with it, which will be seen in our algorithm later.

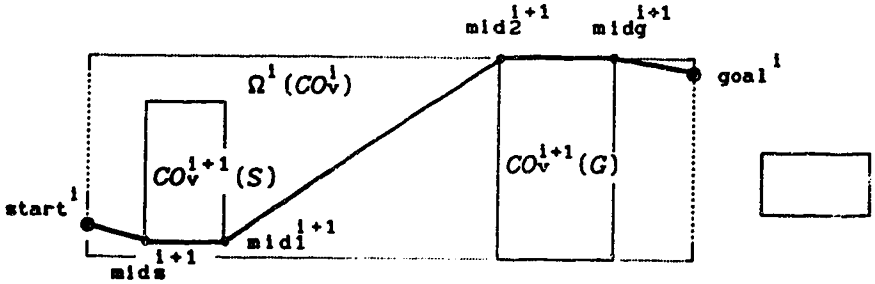


Figure 4.7. Definitions of mid_s, mid_g, mid_1, and mid_2.

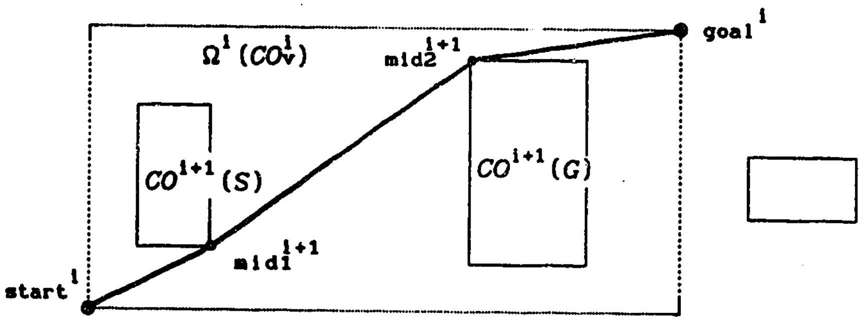


Figure 4.8. A case when only mid_1 and mid_2 exist.

4.2.3.3. Description Of The Search Algorithm

There are four cases that the search algorithm should consider: (1) no obstacle in $\Omega^0(CO)$, (2) only one obstacle in $\Omega^0(CO)$, (3) two obstacles in $\Omega^0(CO)$, and (4) more than two obstacles in $\Omega^0(CO)$, i.e., $\Omega^i(CO_v^i)$ for $i>0$ will be built and searched.

If there is no obstacle within $\Omega^0(CO)$, then a straight line can be simply connected from $start^0$ to $goal^0$ to become the optimal path, *path*. No computations will be required for building a search graph in Ω and therefore no searching is necessary. This is the advantage of using the idea of the reduced search space $\Omega^0(CO)$, although there may be many obstacles outside of $\Omega^0(CO)$ but inside the workspace Ω.

If it is found that there is only one obstacle within $\Omega^0(CO)$, then the problem becomes very simple. This obstacle becomes CO_v^1. All that finding the minimum distance path would require is a simple comparison between the lengths of two paths:

$$\begin{aligned}&\|goal^0 - \text{one element of } VERT_v^1 \text{ on } CO_v^1\|_2 + \\ &\|\text{one element of } VERT_v^1 \text{ on } CO_v^1 - start^0\|_2\end{aligned} \qquad (4\text{-}11)$$

and

$$\begin{aligned}&\|goal^0 - \text{another element of } VERT_v^1 \text{ on } CO_v^1\|_2 + \\ &\|\text{another element of } VERT_v^1 \text{ on } CO_v^1 - start^0\|_2\end{aligned}$$

where $\|\cdot\|_2$ is the 2-norm (Euclidean norm) in $\Re^2$. The smaller one is therefore the optimal path, *path*.

Once we have calculated $\Omega^0(CO)$, CO_v^1, and $path\text{-}l^1$, we shall move inward from $start^0$ and $goal^0$ to the next search space $\Omega^1(CO_v^1)$ if more obstacles are found within $\Omega^1(CO_v^1)$ which is enclosed by mid_1^1 and mid_2^1

on CO_v^i, in order to find the next local optimal path, $path\text{-}l^2$. The searching procedure is exactly the same as before. In general, local start point, $start^i$, and local goal point, $goal^i$, when $i>0$, will have to be determined by heuristics. The rules are:

(1) $start^i$ may be $mid_s^i \in VERT_v^i(S)$ or $mid_1^i \in VERT_v^i(S)$ if mid_s^i does not exists.

(2) $goal^i$ may be $mid_g^i \in VERT_v^i(G)$ or $mid_2^i \in VERT_v^i(G)$ if mid_g^i does not exists.

Each time a local path $path\text{-}l^{i+1}$ is generated, it replaces the portion between mid_1^i and mid_2^i of the previous local path, $path\text{-}l^i$. Therefore the global path, *path*, is actually the sum of all the modified local paths. Each $path\text{-}l^i$ contains a mid-section altered by $path\text{-}l^{i+1}$. Figure 4.9. illustrates an example of $path\text{-}l^i$ with modified section between $mid_1{}^i$ and $mid_2{}^i$ by $path\text{-}l^{i+1}$.

Now we can summarize a general algorithm of finding local optimal path $path\text{-}l^{i+1}$in the ith $(i>0)$ reduced search space $\Omega^i(CO_v^i)$ as follows:

(1) Determine $start^i$ and $goal^i$ using the above rule when more obstacles are found.

(2) mid_1^i and mid_2^i form a new reduced search space $\Omega^i(CO_v^i)$.

(3) Within this new search space find CO_v^{i+1}.

(4) Find $VERT_v^{i+1}$ for all CO_v^{i+1}.

(5) Build the search graph in $\Omega^i(CO_v^i)$ by connecting $VERT_v^{i+1}$ and $start^i$ and $goal^i$ using connection lines as we have discussed in Section 4.2.2..

(6) Search for a local optimal path, $path\text{-}l^{i+1}$, in the search graph, starting from $goal^i$ to $start^i$ using Bellman's principle of optimality.

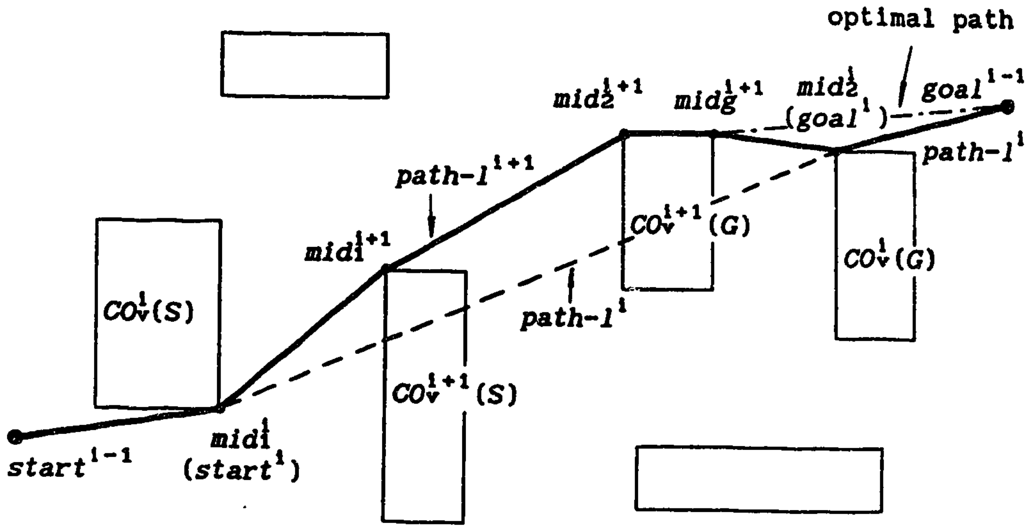


Figure 4.9. $path\text{-}l^{i}$ with modified mid-section by $path\text{-}l^{i+1}$.

(7) If no obstacle is found within $\Omega^i(CO_v^i)$, then connect $start^i$ and $goal^i$ using a straight line to form $path\text{-}l^{i+1}$.

(8) If there is only one obstacle in $\Omega^i(CO_v^i)$, then use (4-11) to find $path\text{-}l^{i+1}$.

(9) Use $path\text{-}l^{i+1}$ to substitute the mid-section of $path\text{-}l^i$ between $start^i$ and $goal^i$.

The above process continues until no more obstacles are found within the last and smallest search space. Now we have the global optimal path, *path*, which is composed of all the local paths, $path\text{-}l^i$. This global path may not necessarily be, in all cases, the globally minimum-distance path because using $path\text{-}l^{i+1}$ to substitute the mid-portion of $path\text{-}l^i$ between $start^i$ and $goal^i$ is just a secure way of avoiding obstacles within $\Omega^i(CO_v^i)$. The example in Figure 4.9. shows such a case. Ideally, the global minimum-distance path should be:

$$start^{i-1} \longrightarrow mid_1^i \longrightarrow mid_1^{i+1} \longrightarrow mid_2^{i+1} \longrightarrow mid_g^{i+1} \longrightarrow goal^{i-1}$$

instead of the one found using our algorithm, i.e.,

$$start^{i-1} \longrightarrow mid_1^i \longrightarrow mid_1^{i+1} \longrightarrow mid_2^{i+1} \longrightarrow mid_g^{i+1} \longrightarrow mid_2^i \longrightarrow$$

$$\longrightarrow goal^{i-1}$$

However, since all the local paths, $path\text{-}l^i$, are optimal, connecting them together would give a very satisfactory collision-free sub-minimum distance path which is almost the same as the ideal minimum-distance path that will take a lot longer time to compute if we use a global search technique, such as the A^* algorithm.

Table 4.1. lists a brief description of our algorithm. It has been implemented using the C programming language.

As we can see, the collision-free path is actually composed of some connected line segments each of which contains many position points of the manipulator end-effector in Cartesian space. Those points should be separated evenly so that the hand movement can be smooth throughout the motion. This also allows us to use joint-interpolated-motion mode for all joint motors. Joint interpolation makes the manipulator travel from one position point to the other along a small arc instead of a straight line even in Cartesian space. It make full use of the manipulator workspace envelope and therefore result in a bigger free space for manipulators made of revolute joints such as the PUMA 560 robot. For Cartesian manipulators, this is not the case, as there are only transfer movements in the motion and joint interpolations are not needed.

4.3. Discussion And Summary

We have presented in this chapter a new algorithm for solving the find-path problem in a two-dimensional environment. Potential collision is prevented by maintaining a non-zero distance between the links of the manipulator and the obstacles. In order to do this, an interference detection technique is used. The design is based on a scheme for representing solid objects, i.e. the robot links and the obstacles in the work space. The swept volume representing the robot path is also taken into consideration.

An important feature of our algorithm is the simple method of approximating the description of C-space obstacles by rectangular hulls enclosing the workspace obstacles. Such an approximation wastes some valuable free space of the manipulator. However, this is better than using circles as in [21] which results in loss of more free space in most cases. It also avoids the difficult task of building a search graph (such as the V-graph) among circular objects. Moreover, our free space loss is more than compensated by the saving in the amount of computation needed for an accurate description of all the obstacles in the configuration space by some other methods. As stated in [83], the most important thing for a space representation is to avoid excess details and time spent on parts of the space that do not affect the operation. It is not necessary to maintain a perfectly detailed model everywhere in the workspace.

If we use a circle to model the manipulator gripper, then we could easily apply the C-space obstacle idea in [49] so that the gripper can be shrunk to a reference point on the gripper (which is the center of the circle) and enlarge the obstacles to the shape of the gripper. Then we can choose from one of the many well-known graph building and searching techniques to find a collision-free path with the traveling distance as its optimization criterion.

We use a method similar to the V-graph technique and add heuristics to the building of our search graph by selecting *COv(S)* and *COv(G)* at each search level. It simplifies the problem even further as the graph becomes so simple that an local optimal path can be found quickly using Bellman's principle of optimality. The global optimal path is a simple connection of all the local optimal paths. It may not necessarily be the ideal path in terms of the minimum Euclidean distance from start to goal. However, it is very close to the ideal path and it is especially efficient for planning collision-free motions using V-graph method. Also as was argued in [49] that the time saved in traveling a shortest distance path may not be justified by the long period of time wasted in planning the path, which is the main idea of our entire work up till now.

The disadvantage of the V-graph method as we have discussed in Chapter 2 is that it always generates a path very close to the boundaries of the C-space obstacles which may lead to certain danger of collisions. However, this is not a problem in our case because we use a symmetric circle for approximating the robot hand. Our program will automatically add a small value of five to ten millimeters for safety clearance to the radius of the hand which will be therefore transferred to the grown parts of all C-space obstacles. This ensures all the time a safe collision-free path for our robot.

This method can be easily applied directly to motion planning for Cartesian and mobile robots in two dimensions. It can also be applied to manipulators with revolute joints by fixing the orientation (o, a, t) of the gripper. Details of such an application to a PUMA 560 will be shown in the next chapter. The algorithm may be extended to applications of three-dimensional motion planning for Cartesian type of robots using an alternative strategy introduced in [49], and for mobile robots using the articulated cylinder idea [73] to represent the free space. In such a case, the 3-D robot hand can be modeled as a sphere [21] [73] and the obstacles can be modeled as tetragonal prisms so that the C-space representation can be very simple.

Table 4.1.

The Minimum-distance Path Searching Algorithm.

Assume: (1) The initial and final configurations of robot hand are known as $start^0 = [X_s, Y_s, Z_s, O_s, A_s, T_s]'$ and

$$goal^0 = [X_g, Y_g, Z_g, O_g, A_g, T_g]'.$$

The values of Z_s, O_s, A_s, T_s, Z_g, O_g, A_g, T_g are fixed throughout the motion to $Z_s=Z_g=Z_f$, $O_s=O_g=O_f$, $A_s=A_g=A_f$, and $T_s=T_g=T_f$ for simulating a two-dimensional environment.

(2) The robot hand radius is r.

(3) Obstacle sizes and locations are known.

(4) Robot workspace is given.

All of the above are stored in a input data file, called *file*, as a priori.

(5) n the worst case number of search layers depending on the clutterness of the environment is given.

```
/***********************************************************************/
{
   Read file to get $start^0$, $goal^0$, $\Omega$, $r$, and obstacle
   sizes and locations. In this stage obstacles are
   transformed to C-space obstacles using (4-3);                     (T1-1)
   Test if $start^0$ and/or $goal^0$ are outside of $\Omega$,
          If outside $\Omega$ return error message and exit;
          Else continue;                                             (T1-2)
   Find the number of obstacles in $\Omega^0(CO)$, objnum;           (T1-3)
   If objnum $\le$ 2     /*if less than three obstacles are in $\Omega^0(CO)$*/
     {
     Determine $CO_v^1$ in $\Omega^0(CO)$ as in Section 4.2.3.1.;    (T1-4)
     If no $CO_v^1$ exists in $\Omega^0(CO)$
       {
       Connect $start^0$ and $goal^0$ with a straight line to
```

```
        form path;                                                      (T1-5)
        Return the number of points, $k$, generated for path;          (T1-6)
        }
      Else
        {
        If there is only one $CO^1_v$ in $\Omega^0(CO)$,
                do (4-11) and (T1-6);                                   (T1-7)
        Else
          {
          Determine $CO^1_v(S)$ and $CO^1_v(G)$;                        (T1-8)
          Calculate the slope of $start^0$—$goal^0$ in order to
          decide $VERT^1_v$ in (4-7). $start^0$, $goal^0$ and $VERT^1_v$
          form a complete set of nodes in $\Omega^0(CO)$;               (T1-9)
          Put cost values (distance to $goal^0$) on $VERT^1_v$;         (T1-10)
          Determine $mid^1_s$, $mid^1_g$, $mid^1_1$, and $mid^1_2$;     (T1-11)
          Connect all the nodes to build the search graph;              (T1-12)
          Perform an optimal path search for path (=$path\text{-}l^1$)
          in the graph using Bellman's principle;                       (T1-13)
          Do (T1-6);                                                    (T1-14)
          }
      }
    Else         /* if objnum > 2 in $\Omega^0(CO)$ */
      {
      Do (T1-7) (T1-8) (T1-9) (T1-10) (T1-11) and (T1-12);              (T1-15)
      For $i$ = 1 to $n$, step +1, Do:
        {
        Update $\Omega^1(CO^1_v)$ using $mid^1_1$ and $mid^1_2$;        (T1-16)
```

```
            Determine $CO_v^{i+1}(S)$ and $CO_v^{i+1}(G)$ in $\Omega^i(CO_v^i)$;            (T1-17)
            Calculate the slope of $start^i$—$goal^i$ in order to
            decide $VERT_v^{i+1}$ in (4-7). $start^i$, $goal^i$ and $VERT_v^{i+1}$
            form a complete set of nodes in $\Omega^i(CO)$;                     (T1-18)
            Put cost values (distance to $goal^i$) on $VERT_v^{i+1}$;           (T1-19)
            Connect all nodes to build the $i$th search graph;                  (T1-20)
            Perform a local optimal path search in the graph
            using Bellman's principle to find $path\text{-}l^{i+1}$ which
            substitutes the mid-section of $path\text{-}l^i$ between
            $start^i$ and $goal^i$;                                             (T1-21)
            Store the number of points, $k_i$, generated for
            $path\text{-}l^{i+1}$;                                              (T1-22)
            Next $i$;                                                           (T1-23)
            }
        Sum up the total number of path points: $k = \sum k_i$;                 (T1-24)
        Connect all $path\text{-}l^i$ to form $path$;                           (T1-25)
        Return $k$;                                                             (T1-26)
        }
    If those $k$ points are not within the robot physical reach,
            send an error message and exit;                                     (T1-27)
    Translate those $k$ configuration points of the robot
    hand to robot motion commands to execute the planned
    collision-free motion, using joint interpolation mode;                      (T1-28)
}
/************************ END OF ALGORITHM ***************************/
```

CHAPTER 5. COLLISION-FREE PATH: APPLICATION OF THE ALGORITHM TO THE PUMA 560 MANIPULATOR

5.1. Introduction

In this chapter we will discuss some practical aspects of the motion planning algorithm described in Chapter 4. Application of this algorithm to a Cartesian manipulator would be quite straight forward and it would even be possible to consider 3-D motion planning tasks. However, we only have access to a PUMA 560 manipulator which has six revolute joints. Therefore special considerations have to be made for this manipulator. The PUMA 560 is a six degree of freedom industrial robot (Figure 5.1.) Generally speaking, at any point in the interior of the working volume of the PUMA 560 the hand (gripper) can be made to follow an arbitrary curve in 3-D space with arbitrary reorientation. However, even for simple motions such as a straight line with fixed orientation, all of the first three joints of the manipulator are involved. For instance, the elbow moves when the hand undergoes translational motions. The direction and magnitude of the elbow movement is a complex function of the direction and location of the hand motion segment. Thus, in considering a motion which enables the hand to avoid an obstacle it is necessary to consider the collision behavior of the elbow at a distant locale [8]. Analysis of joint motions would be simpler if they considered to be uncoupled. Unfortunately the elbow behavior cannot be generally characterized over the range of possible hand motions in any simple way. This means that the path planning problem cannot be decomposed by considering the

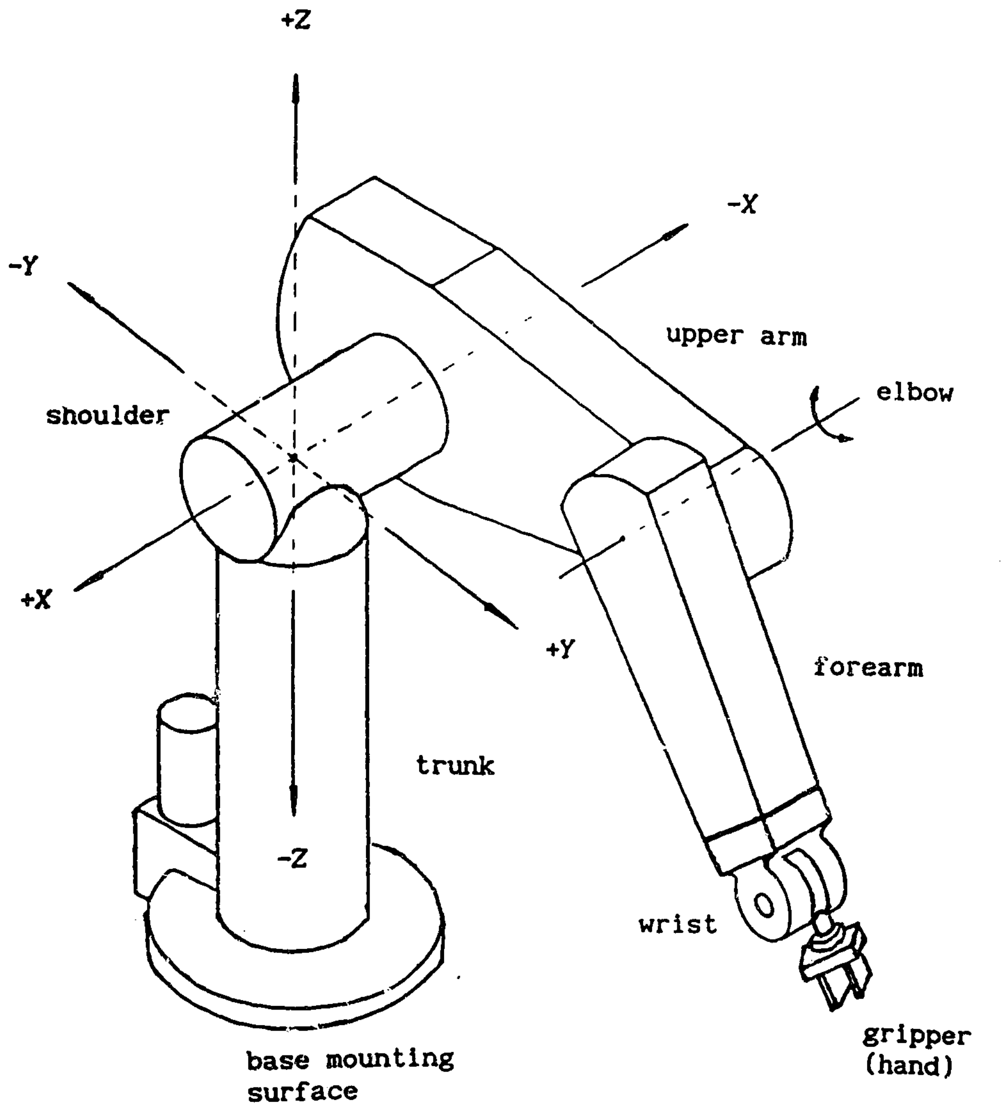


Figure 5.1. PUMA 560 manipulator and its world coordinate system X-Y-Z (Note: Z=0 is not at the base mounting surface).

hand and the elbow separately.

For this reason, application of the algorithm to the PUMA robot has to be limited to a two dimensional working environment. This is done by restricting motions of the PUMA gripper only to the X-Y plane, i.e. Z coordinate has a fixed value, and the orientation (O, A, T) of the hand has also to be fixed throughout the motions among obstacles in the entire workspace.

The PUMA robots use a control language called VAL II. VAL II has some limitations and is not powerful enough for certain applications such as off-line planning of collision-free motions. Because of this, we use a SUN workstation (SUN 3/160) to remotely control the VAL II system which is built into the PUMA controller. The motion planning program is written in C language on the SUN. The generated collision-free paths are translated to the VAL II transformation configurations with the VAL II motion commands which are then sent to the PUMA controller via the supervisor (RS232-C) port. This requires an interpreter that does the communication between the SUN and the PUMA, which we will discuss later in this chapter.

5.2. Brief Description Of The PUMA 560 Robot, Its Workspace And VAL II

The PUMA robot is, perhaps, one of the most popular commercial robot in both industry and research laboratories. The first PUMA (Programmable Universal Machine for Assembly) robot was introduced by Unimation Inc. in 1978. One year later, Victor Scheinman and Bruce Shimano of Unimation designed the first commercial robot control language for the PUMA series robots. It is called VAL which stands for Victor's Assembly Language. As a general computer language VAL is quite weak because it does not support floating-point numbers or character strings, and its subroutines cannot pass arguments. A more recent version, VAL II [70], came out in 1984. It can provide most of these features. The version we are using is the VAL II 2.0 updated in 1986 [79], which is part of the PUMA system and is stored permanently in the controller.

The PUMA 560 (Figure 5.1.) is a 6-link manipulator with six degrees of freedom and all rotational joints. It has eight arm configurations (RIGHTY and LEFTY arm, elbow ABOVE and BELOW arm, WRIST DOWN and UP, and wrist FLIPped and NOFLIPped) for the same end-effector position which makes the inverse kinematics non-unique. To teach the PUMA 560 robot end-effector (corresponding to the last joint) to follow a motion path, either of the two procedures can be used. They are both slow and tedious.

(1) Use the teach pendant to direct manually the movements of the end-effector through each step of the task. These steps are recorded and then stored in the controller memory.

(2) Write a program using the VAL II motion instructions. Position (location and orientation) data and software programs are entered into the controller memory through keyboard or teach pendant.

These are actually point-to-point motions. The PUMA cannot be taught through the teach pendant to follow a continuous path as this will require a large amount of memory to store the path information. All taught points are stored in three forms of data structures. They are (1) *transformations* (referenced to a coordinate system fixed relative to the stationary robot base), (2) *precision points* (referenced to joint angles), and (3) *compound transformations* (referenced to previous locations as a Cartesian coordinate system fixed relative to the tool mounting surface). Compound transformations are not often used. Precision points are robot dependent and can only be used with a robot of the same model. Therefore transformations are probably the most convenient in representing locations and orientations of the end-effector since they are determined with respect to workspace coordinates. Robot independence is thus achieved by defining end-effector locations in terms of a Cartesian (X-Y-Z) reference frame fixed to the base of the robot. The orientations are defined by three angles called Euler angles [64] o, a, and t measured from the world (base) and tool (end-effector) coordinate axes. The values of these three angles are modified in the VAL II representation by the amounts (all angles are measured in degrees):

$$\begin{aligned} O &= o + 90 \\ A &= a - 90 \\ T &= t. \end{aligned} \tag{5-1}$$

Thus, a null or identity transformation of (0, 0, 0, 0, 0, 0) in [64] corresponds to a VAL II transformation:

$$\text{NULL} = (0,\ 0,\ 0,\ 90,\ -90,\ 0) \tag{5-2}$$

(X, Y, Z, O, A, T) defines a complete description of a 3-D position as we have discussed in Section 2.2. of Chapter 2. This is the description that we use in our applications. More about the definitions of O, A, and T can be found in [78,p3-25].

The PUMA 560 has a spherical workspace with the following joint limits [22] :

$$\begin{aligned} -160 &\le \theta_1 \le 160 \\ -225 &\le \theta_2 \le 45 \end{aligned} \tag{5-3}$$

$-45 \leq \theta_3 \leq 225$

$-110 \leq \theta_4 \leq 170$

$-100 \leq \theta_5 \leq 100$

$-266 \leq \theta_6 \leq 266$

The end-effector home position for $(\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6)$ is (0, -90, 90, 0, 0, 0) which corresponds to a transformation at

$$[X, Y, Z, O, A, T]' = [-20.41\text{mm}, 149.09\text{mm}, 921.13\text{mm}, 90, -90, 0]'. \qquad (5\text{-}4)$$

This position can be arrived by running a VAL II command, "DO READY". Since we are simulating a two-dimensional environment, we keep the robot hand at a fixed Z-coordinate value with fixed orientation *O*, *A*, *T*. For most pick-and-place tasks, the robot hand is often facing down towards the working table on which objects are placed. Thus, we choose to maintain the robot hand vertically down throughout the operation. This corresponds to $[O, A, T]' = [0, 90, 0]'$. *Z* value can be anything from -921.13mm to 921.13mm. Z=0 gives the largest workspace. However the working table is often at the same level as the robot base mounting surface at Z=-671.83mm. If we leave some space for the maximum height of objects (or obstacles) and select Z=-400mm with $[O, A, T]' = [0, 90, 0]'$, then the PUMA 560 has a sector like workspace with the lower bound radius R_{min}=250mm and upper bound radius R_{max}=800mm, as shown in Figure 5.2. R_{min} is due to the PUMA 560 trunk. R_{max} is the maximum reach with joint-interpolated motion.

5.3. Communication Between The Supervisory Computer And VAL II

Although VAL II is easy to learn, its instructions are clear, concise and self-explanatory, and it can be quickly programmed by

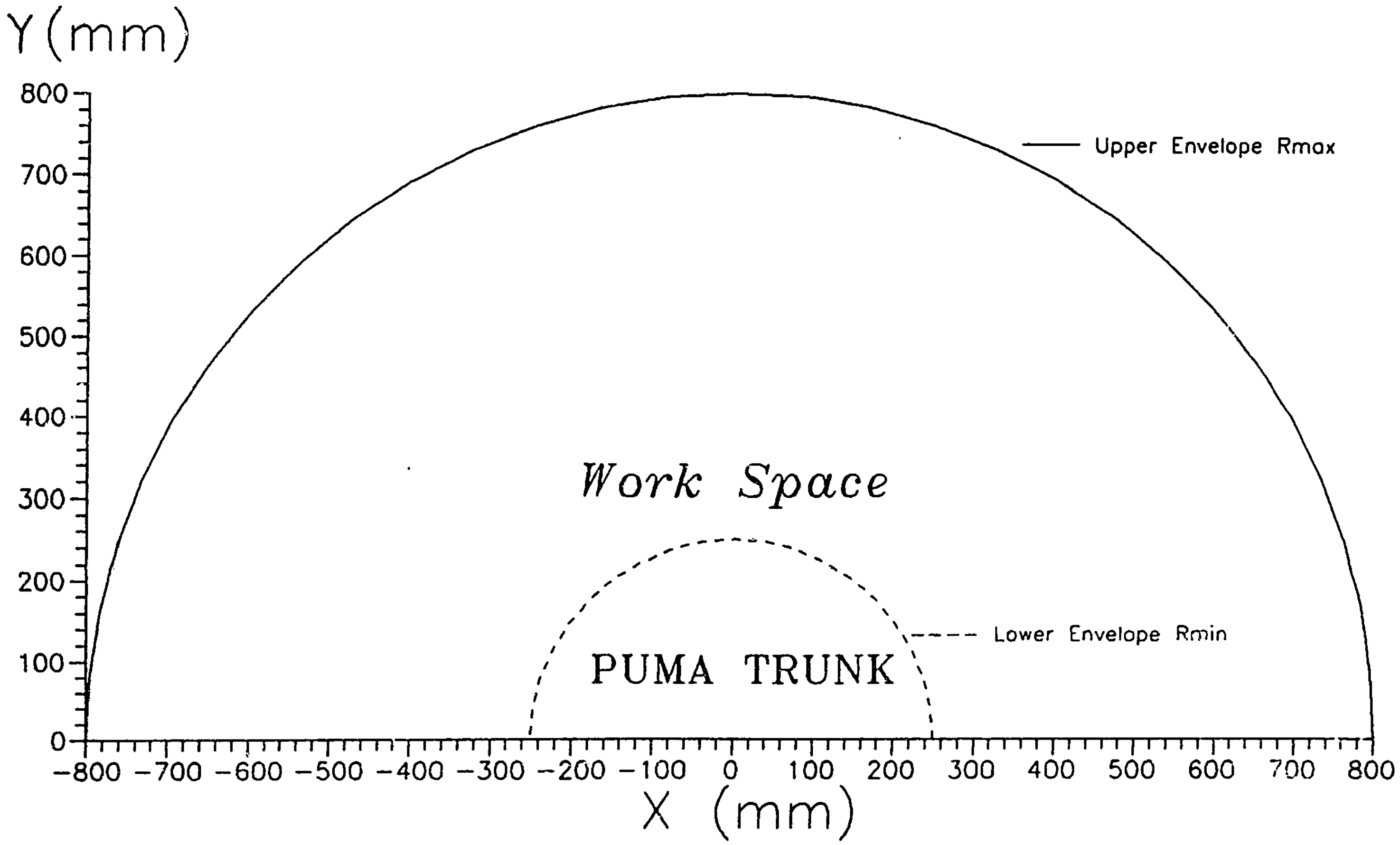


Figure 5.2. The workspace of the PUMA 560 defined by the upper bound envelope R_{max} and lower bound envelope R_{min}.

combining predefined subtasks to perform many complex operations, it suffers from the following shortcomings:

— VAL II has only a line editor. It is quite inconvenient to create and modify large application programs.

— VAL II has a limited library of mathematical functions. It can not pass addresses of variables. It does not have the ability of building and searching a graph.

— VAL II cannot accept any functions or subroutines written in high level languages, such as PASCAL and C.

— VAL II controller has a limited memory to store large programs and data files.

For the above reasons, we use a supervisory computer (SUN 3/160) to do all the programming, data processing and computations. The motion path is generated independent of VAL II and the communication between VAL II and the SUN takes place only once, to upload the generated path data from the SUN to VAL II. It is done through a communication software that was developed for a VAX computer in [34] with extensive modifications for the SUN made based on our application requirements. This software is a package of many thousand lines of C source code. (A similar work of communication between a VAX-11/780 and a PUMA 560 was done in [17] for robot binary vision using a multiprocessing controller under NRC(Canada)'s Harmony operating system.) Figure 5.3. illustrates the PUMA/SUN system. The following is a brief description of the communication software.

VAL II permits the connection of a supervisory computer via a RS232-C line (at a speed of 9600 baud) using DDCMP communication protocols [15] to ensure the messages sent and received are error

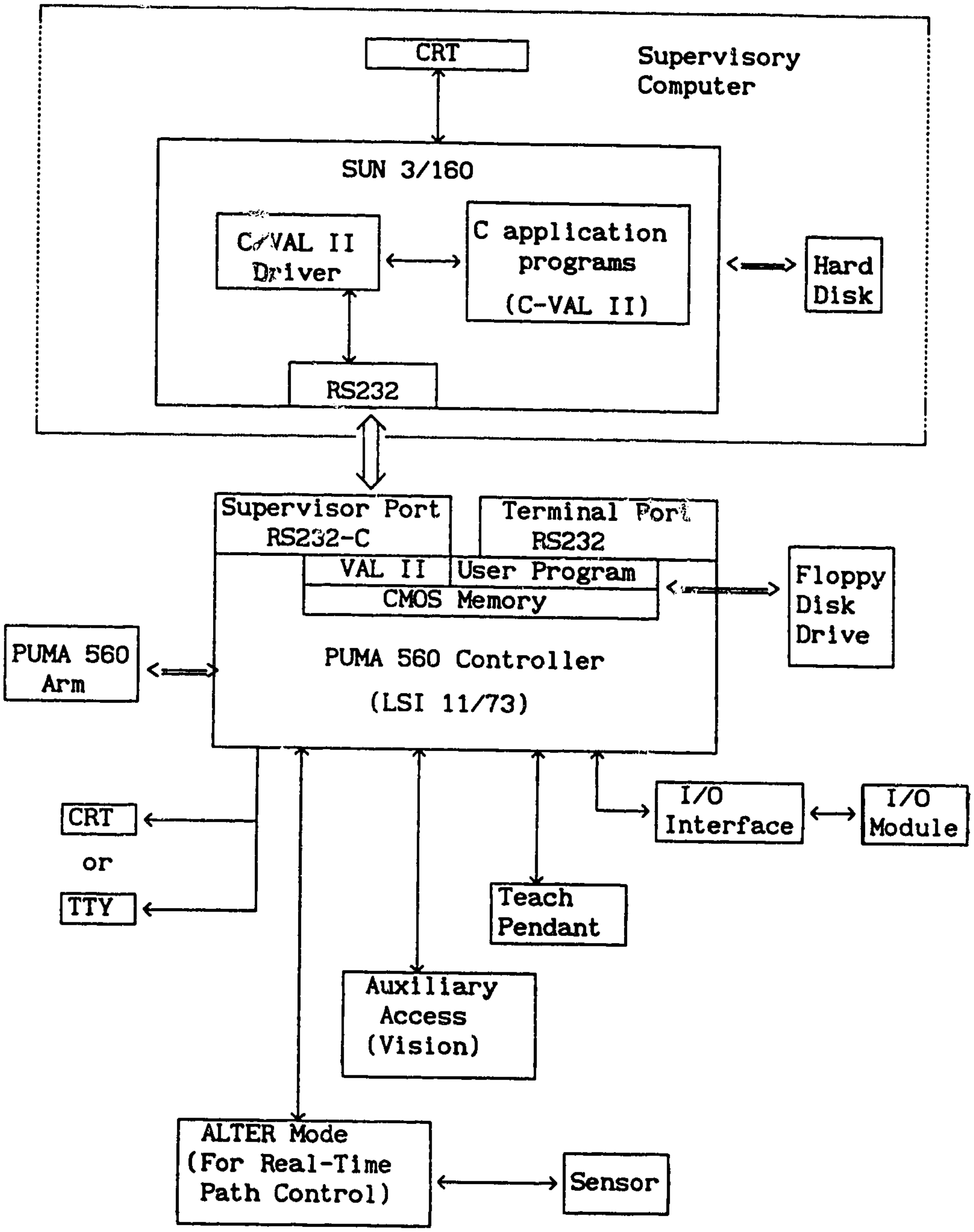


Figure 5.3. The PUMA 560 controller and its supervisory computer.

free. We should note that such a communication can be quite slow in transferring large data files for planning complicated motions (refer to Chapter 6 for more discussions). The pin to pin connections between the PUMA and the SUN are as follows:

From the PUMA 560 (supervisor port)	To the SUN (RS232 Port A)	
pin 5 ⟷	pin 7	Signal ground
pin 6 ⟷	pin 3	LSI receive +
pin 7 ⟷	pin 2	LSI transmit +

The messages sent through this line are structured in three layers:

(1) Bottom layer: performs the transmission using Digital Equipment Corporation's DDCMP protocol [15].

(2) Middle layer: permits the division of messages into logical units corresponding to different operating modes such as executions of programs or commands, loading and storing of files, etc.

(3) Top layer: permits the differentiation of the messages corresponding to the same logical unit. For example, we can differentiate between the messages that ask the user for data, or give data to the user, etc.

In this fashion the SUN can completely control the VAL II system remotely. The communication software is now able to do the following tasks: (1) to send VAL II commands from the SUN directly, (2) to allow a user to write application programs on the SUN using C functions that create the appropriate VAL II commands. We call those functions C-VAL II commands.

5.3.1. Direct Control Of The PUMA From The SUN

Direct control of the PUMA from the SUN by software is briefly described in Appendix 1. It allows VAL II commands to be sent to the PUMA from the SUN just as if the SUN were a dumb terminal.

This software is useful since we can use SUNTOOLS to monitor VAL II operations and the PUMA hand positions with multiwindows. Another use of this software is to edit VAL II commands and data files. It is very convenient to use a UNIX visual editor, such as "vi", to create and modify an ASCII file on the SUN, which is a rather difficult task VAL II line editor. We can also download or upload files using gram. For example, if we want to download a file from the VAL oller memory to the SUN we need simply to do the following:

Step 1: start controller (power on both the controller and the PUMA arm),

— Step 2: CA (calibrate the PUMA),

— Step 3: ENABLE NETWORK,SUPERVISOR,DISK.NET
ENABLE REMOTE.PIN (allows result display on the SUN),

— Step 4: PSTORE or STORE SUN-filename=VAL-filename.

If we want to upload a file from the SUN to the VAL II controller memory, then do Step 1—3 as before, plus

— Step 4: LOAD SUN-filename (e.g. LOAD MOTION.V2.VAL, then MOTION.V2.VAL will be transferred and stored in the VAL II controller memory).

Because downloading or uploading uses the RS232 serial line at 9600 baud rate, it will take a few seconds to finish the operation. Another way to speed up this is to generate all the necessary VAL II commands as ASCII character strings from a C program in the SUN and dump them to the terminal port of the PUMA controller. This requires no DDCMP and error checks and is much faster but not safe for transferring large data files. More on this can be found in Section 6.1.

5.3.2. Communicating From A C-Application Program Using C-VAL II

As another way of remote control of the PUMA 560 from the SUN, C functions (we call them C-VAL II) that are similar to VAL II commands can be called from a library that also contains DDCMP routines for communicating from a C program to the PUMA. Many user needed VAL II commands can also be written in C and appended to the library. These C-VAL II functions are very useful for implementing motion planning algorithms on the PUMA 560 robot by issuing VAL II commands from a C program. Appendix 2 describes the details of this library and some of the most useful C-VAL II functions.

There are two types of data structures used in C-VAL II. They correspond to the first two of the three data types in VAL II that we have discussed in Section 5.2, which are:

(a) *transformation* (location) type of data structure as

```
          struct location
                {
                char *quality="location";                    (5-5)
                double x, y, z, o, a, t;
                };
```

The parameters correspond to the end-effector location (*X*, *Y*, *Z*) and its orientation (*O*, *A*, *T*) (i.e. the Euler angles in the VAL II representation).

(b) *precision point* type of data structure as

```
struct location
     {
     char *quality="ppoint";                          (5-6)
     double x, y, z, o, a, t;
     };
```

The six parameters (x, y, z, o, a, t) in this case correspond to the six joint values (θ_1, θ_2, θ_3, θ_4, θ_5, θ_6).

In real applications, transformation type of data structure is more convenient to define a position point of the gripper. Therefore all of our generated path data are formatted in transformation structure as shown in Appendix 4.

5.4. Finding A Safe Path For The PUMA 560 Among Obstacles

The find-path algorithm we proposed in Chapter 4 can be implemented on the PUMA 560 robot by generating the safe path off-line, using the communication software to upload the path data file to the PUMA controller and then running a small prestored VAL II program to execute the motion according to the path data information uploaded. Since the PUMA 560 has a lower bound envelope (the trunk) which must be avoided, the algorithm in Chapter 4 will have to be modified. This envelope is a half circle of R_{min}=250mm as in Figure 5.2. In Section 2.7, We proposed a method of avoiding circular objects that is better than the method in [21]; the details are discussed below.

5.4.1. Avoiding The Lower And Upper Bounds Of The PUMA Workspace

The upper bound of the PUMA workspace is easy to avoid. In most cases, if the initial ($start^0$) and final ($goal^0$) positions of the PUMA hand are within R_{max} (see Figure 5.2.), all intermediate positions (i.e. the intermediated path points generated by our program) will also be inside R_{max}. If either $start^0$ or $goal^0$ is outside of the envelope defined by R_{max}, a safe path will not exist because the path contains portions that are out of the PUMA's reach. This simple test can be placed at the beginning of the program. An error message will occur and the program will terminate if the test finds the norm of $start^0$*-origin* and/or $goal^0$*-origin* is greater than R_{max}.

Using the notation in Chapter 4 for the lower bound envelope R_{min}, a subroutine will be called in our program each time an $\Omega(CO_v^1)$ is constructed and a search is required between $start^1$ and $goal^1$ to compute an alternate path as shown in Figure 2.9 if it finds that an intermediate point of the local optimal path $path\text{-}l^1$ is within R_{min}. The computation is fairly simple and takes little time.

Figure 5.4 illustrates a circle and its two tangent lines passing through the two points (x_s, y_s) and (x_g, y_g). We wish to find the two tangent points (x_{c1}, y_{c1}) and (x_{c2}, y_{c2}) so that a path consists of

$$(x_s, y_s) \xrightarrow{\text{line}} (x_{c1}, y_{c1}) \xrightarrow{\text{arc}} (x_{c2}, y_{c2}) \xrightarrow{\text{line}} (x_g, y_g) \qquad (5\text{-}7)$$

can be found.

Let (a, b) be the center of the circle, r be the radius, and l_1 be the tangent line segment from (x_s, y_s) to (x_{c1}, y_{c1}). Thus we have:

$$\text{circle:} \quad (x_{c1}-a)^2 + (y_{c1}-b)^2 = r^2 \qquad (5\text{-}8)$$

$$l_1: \quad y_{c1} = -\left[\frac{x_{c1}-a}{y_{c1}-b}\right](x_{c1}-x_s) + y_s \qquad (5\text{-}9)$$

x_{c1} can then be solved from

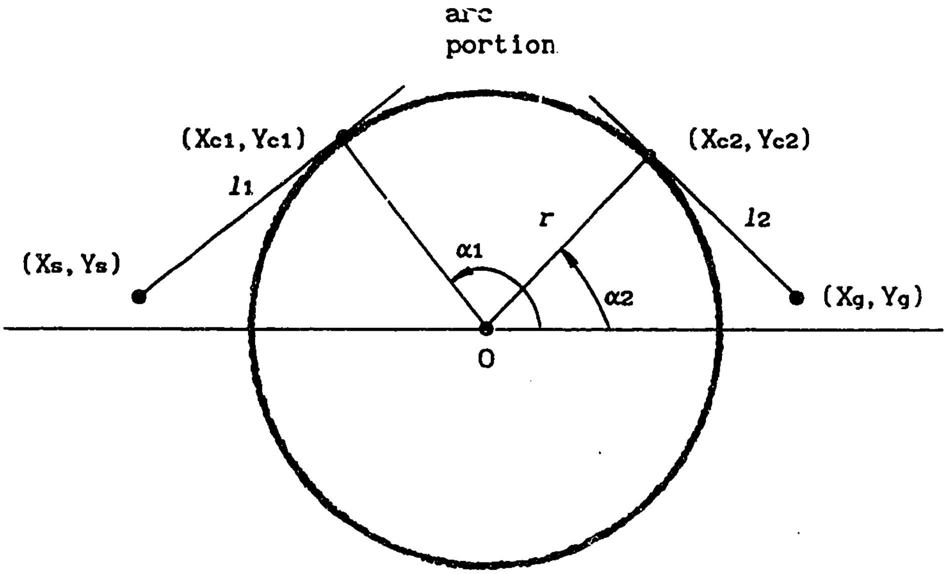


Figure 5.4. Finding an alternate path for avoiding the PUMA's lower-bound envelope R_{min}.

$$A\,X_{c1}^{2} + B\,X_{c1} + C = 0 \tag{5-10}$$

where

$$A = (x_s-a)^2 + (y_s-b)^2 \tag{5-11}$$

$$B = 2(a-x_s)(r^2-a^2+ax_s) - 2a(y_s-b)^2$$

$$C = ax_s(2r^2+ax_s-2a^2) + (r^2-a^2)\left[(r^2-a^2)-(y_s-b)^2\right].$$

and y_{c1} can be found from

$$y_{c1} = \frac{r^2-(x_{c1}-a)(x_s-a)}{y_s-b} + b. \tag{5-12}$$

Similarly, x_{c2} and y_{c2} of l_2 can be solved by substituting x_s and y_s with x_g and y_g in (5-8), (5-9), (5-10), (5-11) and (5-12).

In the case of the PUMA 560, we can set a=0 and b=0, then (5-11) and (5-12) can be simplified as

$$A = x_s^2 + y_s^2 \tag{5-13}$$

$$B = -2x_s r^2$$

$$C = r^2(r^2-y_s^2)$$

and

$$y_{c1} = \frac{r^2-x_{c1}x_s}{y_s} \tag{5-14}$$

The arc portion is computed as follows

$$\alpha_1 = \mathrm{atan2}(y_{c1},x_{c1}) \tag{5-15}$$

$$\alpha_2 = \mathrm{atan2}(y_{c2},x_{c2})$$

$$\Delta\alpha = (\alpha_2 - \alpha_1)/P_{arc}$$

where P_{arc} is the number of points used to describe the arc. If we define a location structure as the *transformation* type in (5-1),

struct loc {double x, y, z, o, a, t;}; (5-16)

then those points can be stored in an array of "struct loc point[]" by the following

```
for $i$=1 to $P_{arc}$, step +1, do:                                (5-17)
    {
    $\alpha_i = \alpha_i + \Delta\alpha$;
    point[$i$].x = r*cos($\alpha_i$);
    point[$i$].y = r*sin($\alpha_i$);
    point[$i$].z = -400;
    point[$i$].o = 0;
    point[$i$].a = 90;
    point[$i$].t = 0;
    NEXT $i$;
    }
```

This alternate path will substitute the unsafe portion of $path\text{-}l^{i}$ between $start^{i}$ and $goal^{i}$. It can also be used to substitute the modified mid-section $path\text{-}l^{i+1}$ of $path\text{-}l^{i}$ (refer to Figure 4.9.) if any portion of $path\text{-}l^{i+1}$ between $start^{i+1}$ and $goal^{i+1}$ is found to be within the envelope defined by R_{min}.

5.4.2. Description Of The Find-Path Algorithm For The PUMA 560

The technique of planning collision-free motions for the PUMA 560 is a combination of (1) the search algorithm discussed in Chapter 4 that generates an optimal path in a reduced search graph among obstacles enclosed by rectangular hulls, (2) the alternate path algorithm for avoiding the PUMA lower bound envelope, (3) the uploading of the path data to the PUMA controller using C-VAL II commands via the supervisor port, and (4) the execution of the motions by calling a small VAL II routine stored in the PUMA's CMOS memory.

An application program called "av" performs the above operations. A header file "multi.h" that contains all the definitions is included in the same directory as "av". Before running the program, an input file must be written first by the programmer. This file contains information of the locations of all obstacles, the initial and final positions of the PUMA hand, and the resolution of the path (namely, the number of evenly separated position points required to approximate the path). The format of an input file is given in Appendix 4. Higher resolution means more position points generated and gives a more accurate approximation of the path but takes a longer time to be uploaded from the SUN to the PUMA. Usually we choose the resolution as 50mm between every two points. This program can find a collision-free path among up to nine obstacles in the PUMA workspace. It has been compiled using a GCC (a better compiler than CC) on the SUN 3/160 computer (it has also been compiled on IBM-PCs). In most cases, each run takes less than one second to generate an optimal collision-free path. To ensure that the path is really safe, we also include an option of displaying the path and the environment on a SUNTOOL window using a 2-D SunCGI graphics program. This will show the path on the screen before uploading the path data to the PUMA and executing the motion.

Table 5.1.— Table 5.3. contain a brief description of the complete algorithm. It is a modified version of Table 4.1. for the PUMA 560 applications. A list of all the routines with some explanations, and the "makefile" for compilation on the SUN 3/160 are given in the Appendix 3.

Table 5.1.

The Motion Planning Algorithm for the PUMA 560.

Assume: (1) The initial and final configurations of the PUMA 560 hand are known as *start*^{0} = $[X_s, Y_s, Z_s, O_s, A_s, T_s]'$ and

goal^0 = $[X_g, Y_g, Z_g, O_g, A_g, T_g]'$.

The values of Z_s, O_s, A_s, T_s, Z_g, O_g, A_g, T_g are fixed throughout the motion to Z_s=Z_g=-400mm, O_s=O_g=0, A_s=A_g=90, and T_s=T_g=0 in order to simulate a two-dimensional environment.

(2) The robot hand in our example has a small radius of r=5mm.

(3) Obstacle sizes and locations are known and the maximum number of obstacles is nine.

All of the above are stored in a input data file, called "*input*".

(4) Robot workspace is given as in Figure 5.2. with R_{min}=250mm and R_{max}=800mm.

(5) n the worst case number of search layers depending on the clutterness of the environment is given.

(6) TRUE ≡ 1.

```
/**********************************************************************/
void main()
{
   Read "input" to get $start^0$, $goal^0$, $\Omega$, $r$, and obstacle
   sizes and locations. In this stage  obstacles  are
   transformed to C-space obstacles using (4-3);                (T2-1)
```

```
    Test if $start^0$ and/or $goal^0$ are outside of $\Omega$,          (T2-2)
           If outside $\Omega$ return error message and exit;
           Else continue;
      While (TRUE)
           {
           Enter a character (g,e,l,r,d,p,q);
           case 'g': k=generate_path();            /* see Table 5.2. */
           case 'e': execute_motion();             /* see Table 5.3. */
           case 'l': list_all(k);  /* list all generated path points */
           case 'r': k=read_pathfile();/* read a stored optimal path */
           case 'd': display_input();      /* display input data file */
           case 'p': plot_path(); /*plot the generated path on screen*/
           case 'q': exit();                      /* quit the program */
           }
}
/************************* END OF MAIN() *****************************/
/************************ END OF ALGORITHM **************************/
```

Table 5.2.

The Subroutine for Generating the Optimal Path.

```
/****************************************************************************/

int generate_path()      /*generate the optimal path*/
{
   Find the number of obstacles in $\Omega^0(CO)$, objnum;                (T2-3)
   If objnum $\le$ 2  /*if less than three obstacles are in $\Omega^0(CO)$*/
     {
     Determine $CO_v^1$ in $\Omega^0(CO)$ as in Section 4.2.3.1.;          (T2-4)
     If no $CO_v^1$ exists in $\Omega^0(CO)$
       {
       Connect $start^0$ and $goal^0$ with a straight line to
       form path;                                                   (T2-5)
       Test if any part of path is within $R_{min}$;                    (T2-5a)
          If (TRUE)
               Find the alternate path to avoid the lower           (T2-5b)
               bound envelope of the PUMA as we discussed
               in Section 5.4.1.;
       Return the number of points, k, generated for path,
       and store those points in an output file, "output",
       in the transformation format as (5-16);                      (T2-6)
       }
     Else
       {
        If there is only one $CO_v^1$ in $\Omega^0(CO)$
          {
          Do (4-11) to find path;                                        (T2-7)
          Test if any part of path is within $R_{min}$;                  (T2-7a)
              If (TRUE) do (T2-5b);                                      (T2-7b)
          Do (T2-6);                                                     (T2-7c)
          }
        Else
          {
          Determine $CO_v^1(S)$ and $CO_v^1(G)$;                         (T2-8)
          Calculate the slope of $start^0$—$goal^0$ in order to
          decide $VERT_v^1$ in (4-7). $start^0$, $goal^0$ and $VERT_v^1$
          form a complete set of nodes in $\Omega^0(CO)$;                (T2-9)
          Put cost values (distance to $goal^0$) on $VERT_v^1$;          (T2-10)
          Determine $mid_s^1$, $mid_g^1$, $mid_1^1$, and $mid_2^1$;      (T2-11)
          Connect all the nodes to build the search graph;               (T2-12)
          Perform an optimal path search for path (=$path\text{-}l^1$)
          in the graph using Bellman's principle;                        (T2-13)
          Do (T2-5a) and if (TRUE) do (T2-5b);                           (T2-13a)
          Do (T2-6);                                                     (T2-14)
          }
      }
    Else          /* if objnum > 2 in $\Omega^0(CO)$ */
        {
        Do (T2-7) (T2-8) (T2-9) (T2-10) (T2-11) and (T2-12);             (T2-15)
        For $i$ = 1 to $n$, step +1, Do:
          {
            Update $\Omega^i(CO_v^i)$ using $mid_1^i$ and $mid_2^i$;                (T2-16)
            Determine $CO_v^{i+1}(S)$ and $CO_v^{i+1}(G)$ in $\Omega^i(CO_v^i)$;     (T2-17)
            Calculate the slope of $start^i$—$goal^i$ in order to
            decide $VERT_v^{i+1}$ in (4-7). $start^i$, $goal^i$ and $VERT_v^{i+1}$
            form a complete set of nodes in $\Omega^i(CO)$;                          (T2-18)
            Put cost values (distance to $goal^i$) on $VERT_v^{i+1}$;              (T2-19)
            Connect all nodes to build the $i$th search graph;                       (T2-20)
            Perform a local optimal path search in the graph
            using Bellman's principle to find $path\text{-}l^{i+1}$ which
            substitutes the mid-section of $path\text{-}l^i$ between
            $start^i$ and $goal^i$;                                                  (T2-21)
            Test if any part of $path\text{-}l^{i+1}$ is within $R_{min}$;           (T2-21a)
                If (TRUE) do (T2-5b);                                                (T2-21b)
            Store the number of points, $k_i$, generated for
            $path\text{-}l^{i+1}$;                                                   (T2-22)
            Next $i$;                                                                (T2-23)
            }
        Sum up the total number of path points: $k = \sum k_i$;                      (T2-24)
        If those $k$ points are not within the robot physical reach $R_{max}$,
                send an error message and exit;                                      (T2-24a)
        Connect all $path\text{-}l^i$ to form $path$;                                (T2-25)
        Do (T2-6);                                                                   (T2-26)
        }
}
/********************** END OF GENERATE_PATH() ************************/
```

Table 5.3.

The Subroutine for Executing The PUMA Motions.

Assume: (1) A collision-free path has been generated and stored in a file called "*output*" in transformation format.

(2) The auto-start program in Table A5.1 has been executed.

(3) Path data will be transferred to "move4[]" in VAL memory.

(4) A small VAL II program called "motion" is stored in the VAL II memory. It will move the PUMA in joint interpolation mode to follow the collision-free path that has been transferred to "move4[]":

```
.PROGRAM motion
FOR INDEX=0 to LLAST(move4[]) ;LLAST indicates the last
     MOVE move4[INDEX]        ;location variable
.END
```

```
/*****************************************************************/
void execute_motion()
{
     Declare all location points as transformation type
     as in (5-5);                                             (T2-27)
     Start_supervisor() to initialize the communication;      (T2-28)
     Read the number of location points of the generated
     path in file "output", k;                                (T2-29)
     Move the PUMA arm to its home position as in (5-4);      (T2-30)
     Delete the contents of all variables in "move4[]".       (T2-31)
```

```
    Upload the k points from "output" in the SUN to
    "move4[]" in the VAL II memory.                              (T2-32)
    Execute the VAL II program "motion" once.                    (T2-33)
    When "motion" is done, display the final hand
    configuration to verify if it is the desired goal.           (T2-34)
    Move PUMA back to home position for safety purpose.          (T2-35)
    Abort_supervisor() to terminate DDCMP communication.         (T2-36)
}
/******************* END OF EXECUTE_MOTION() ************************/
```

5.5. Examples and Results

In this section we present some experimental results of our find-path algorithm. Before we do this, a brief summary of the procedure is given.

First use "suntools" option to open a suntool window. Place up to nine obstacles in the PUMA workspace and enclose each one with an appropriate rectangular hull as in Figure 4.1. Decide the start and goal positions of the PUMA hand. Next simply type

"av *input*"

where "*input*" must be written in the format as in Appendix 4. Then a menu will appear as follows:

```
****************************************************

'd' == display input data file

'e' == execute collision-free motion

'g' == generate path (location points)
```

```
'l' == list all location points
'p' == plot the generated path on screen
'q' == quit the program
'r' == read an already generated path file
*** Enter your choice now (d,e,g,l,p,q,r):
*****************************************************
```

Answer 'g' for generate, and in less than one second the optimal collision-free path will be generated and stored in an *output* file "path.rec". A message will tell the programmer if the path is within the PUMA's physical reach. If it is a safe path the same menu will appear again. This time answer 'l' to list all the points (locations and orientations) of the path. Then answer 'p' for plot, which displays the path and the PUMA environment on the suntool window. To interrupt the display type the carriage return key which brings us back to the menu again. Now turn on the PUMA controller power and execute the auto-start program in Table A5.1., then answer 'e' to execute the motion. C-VAL II commands in Table 5.3. will start the communication and upload the path location points to VAL II. If no CRC errors are found, the VAL II program "motion" will be in operation to move the PUMA in joint-interpolated fashion for all joint motors.

The following cases were tried on the PUMA 560. The results are all satisfactory, i.e. they are safe and reasonably optimal.

Case 1: A particular obstacle configuration is chosen and different start and goal points are tried. The sizes of the obstacles (not their locations) are changed and the experiments are repeated.

Figure 5.5. (a) shows the optimal collision-free path among nine obstacles. Figure 5.5. (b) has the same obstacles and start position as (a) but a different goal position. The resulting path is different from Figure 5.5. (a). Figure 5.5. (c) has the same start and goal as (a) but some of the obstacles have bigger sizes than that in (a) which makes the environment more cluttered. The resulting path is of course different from that of (a).

Figure 5.6. (a) and (b) show the different paths for another set of the same obstacles but different start and goal points.

Case 2: In a very cluttered environment (Figure 5.7.).

Case 3: In a less cluttered environment when the only thing to avoid is the PUMA trunk, i.e. the lower bound envelope R_{min} (Figure 5.8.). Note in Figure 5.8., all obstacles are outside the reduced space $\Omega^0(CO)$ as defined in Section 4.2.3.1. of Chapter 4. Therefore no searching computation is needed except the computation for avoiding R_{min}.

Case 4: Avoiding both obstacles and R_{min}.

In order to demonstrate clearly how the lower bound envelope is avoided, we can increase R_{min} from 250mm to 390mm. Examples of avoiding R_{min} are shown in Figure 5.9.

Case 5: Avoiding obstacles with the help of mid_s and/or mid_g mid-points (Figure 5.10.). This has been discussed in Section 4.2.3.2.

5.6. Evaluation And Discussion

In this section we discuss and compare the differences between our approach and some other approaches that we have examined in Chapter 2.

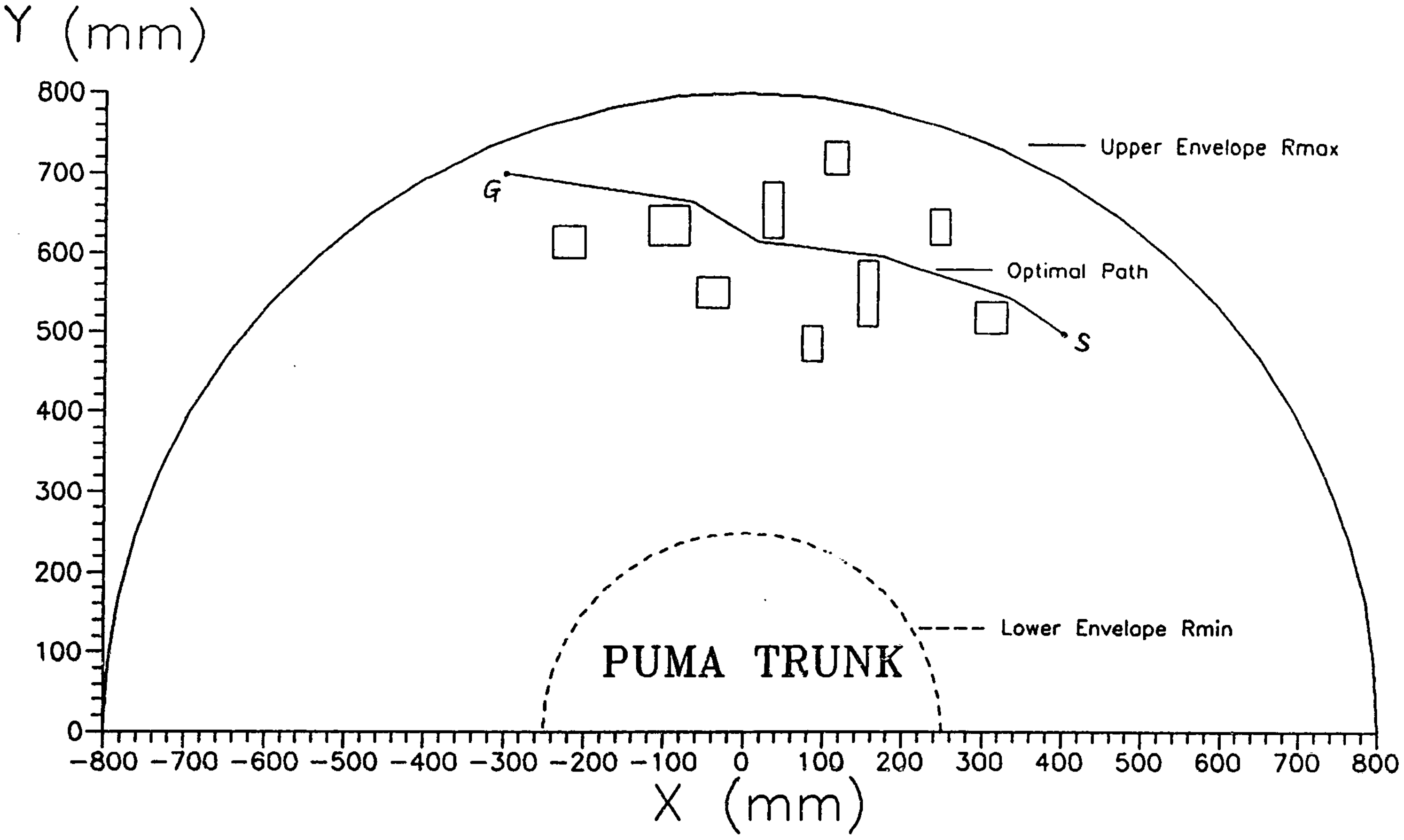
The approach we propose here is similar to Lozano-Pérez's C-space approach [48] [49] [50] in that both methods shrink the moving object to a reference point on the object and enlarge the obstacles to the shape of the object, and they both search for a collision-free path in a visibility graph. However, our approach is simpler because all workspace obstacles are enclosed by rectangular hulls which have the same orientation. This reduces the computation of C-space obstacles significantly without losing much free space. Another difference is that our V-graph is built in a reduced space $\Omega^i(CO_v^i)$ with heuristics. Each search of the optimal path is local to its reduced space among at most six nodes in the simplified search graph, which speeds up the searching process.

In Section 2.7. we compared our method of avoiding a circular object with Fletcher's method in [21]. It is easy to see that our alternate path is shorter and smoother (see Figure 2.8. and Figure 2.9.), and it takes very little time to compute the path as described in Section 5.4.1. This advantage is quite significant when the circular object is larger than the moving object, namely the robot hand.

Direct comparisons with many of the existing motion planning algorithms are not so straight forward because many of them only work in their own specific circumstances. However, we were able to compare our algorithm with the Octree Free Space (OFS) approach proposed by Herman in [32]. We have discussed Herman's algorithm in Section 2.5. which is designed for Cartesian robots. The biggest disadvantage of OFS is that it cannot find an optimal path. Also at a local minimum, that is, a point which has a lower cost (shorter Euclidean distance to the goal) than any of its neighboring points, hill climbing method fails and A^* algorithm will have to be invoked to perform an expensive global search through the graph. It is also difficult to evaluate the efficiency of Herman's approach since he did not mention how to determine a resolution limit of the decomposition of the octree space so that the search for a collision-free path does not have to be infinite. Figure 5.11.(a) and (b) show the paths obtained by the OFS method and our algorithm. Both have the same start and goal positions. It is easily seen that the path in Figure 5.11.(b) which is created by our algorithm demonstrates a better quality than that in Figure 5.11.(a) which is created by OFS. As can be seen from Figure 5.11.(a), A^* search was invoked three times to help hill climbing get out of the local minimums, which means that global search was performed three times through the entire graph. Whereas in our case, only three local searches were required starting from the outmost pair A-and-B to C-and-D and last to E-and-F, which are much more efficient than OFS.

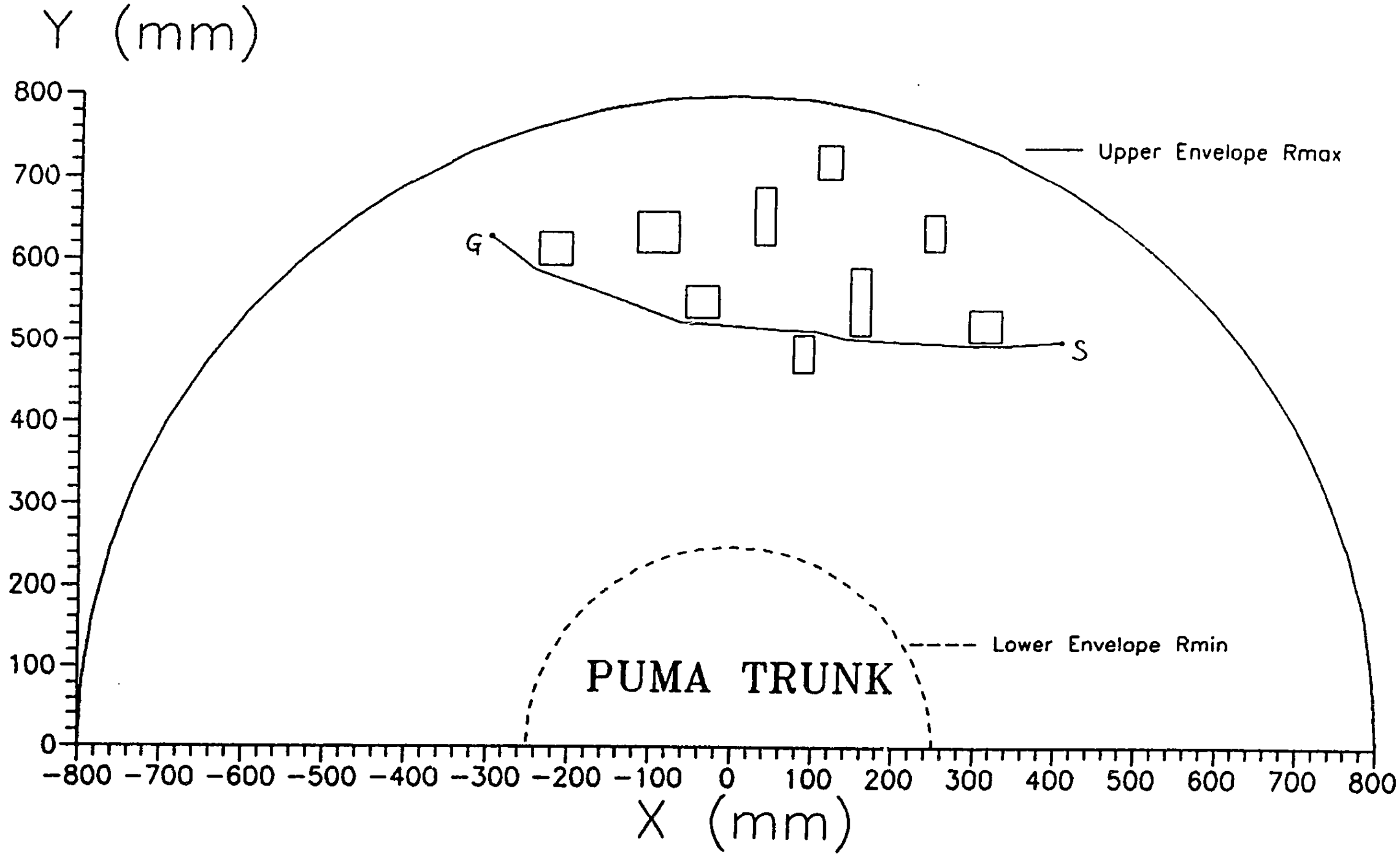
Our algorithm is very simple and requires little computation time (always less than a second to generate a path on the SUN 3/160 with an MC68881 floating-point coprocessor). We have implemented our algorithm using the C language, which makes the application program quite long (about five thousand lines of source code). The advantage of using C is its flexibility which allows us to link our program with the C-VAL II library which can remotely control the PUMA motions from the SUN supervisory computer.

Because our algorithm is based on Lozano-Pérez's C-space concept and V-graph search strategy, it should work just as well as his algorithm when the obstacles are two-dimensional, that is, *C*-space obstacles are $CO_A(B_i)=CO_A^{xy}(B_i)$. Our V-graph search technique in the reduced space $\Omega^1(CO_V^1)$ can be extended directly to deal with three-dimensional C-space obstacles for Cartesian type of manipulators, i.e., $CO_A(B_i)=CO_A^{xyz}(B_i)$. However, it will suffer from the same drawbacks as Lozano-Pérez's algorithm when the obstacles are three-dimensional. These are (1) shortest paths do not usually traverse along the vertices of $CO_A^{xyz}(B_i)$, which means the paths found are not in general optimal paths, (2) there may not exist a safe path via the vertices $CO_A^{xyz}(B_i)$ at all within the entire workspace Ω, which often happens when those vertices are inaccessible. Although other types of collision-free paths (via edges of $CO_A^{xyz}(B_i)$ for example) within Ω may exist, V-graph search algorithm will not find them. In [49] an alternate and complete searching strategy for finding safe suboptimal paths for a Cartesian robot is proposed, which may be applicable to our algorithm.



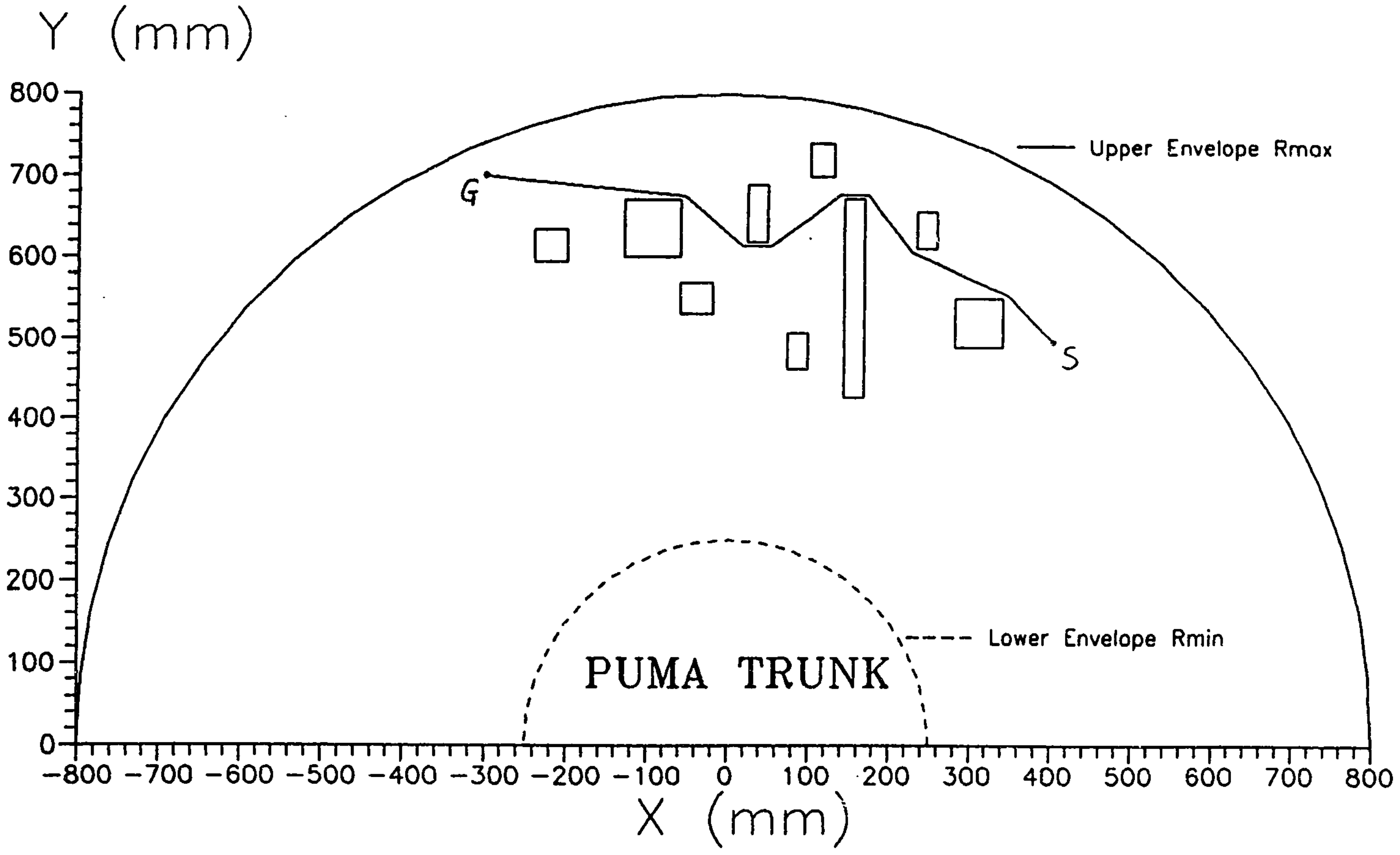
(a)

Figure 5.5. Collision-free paths generated in different situations: (b) has different goal from (a), (c) has a different obstacle from (a).



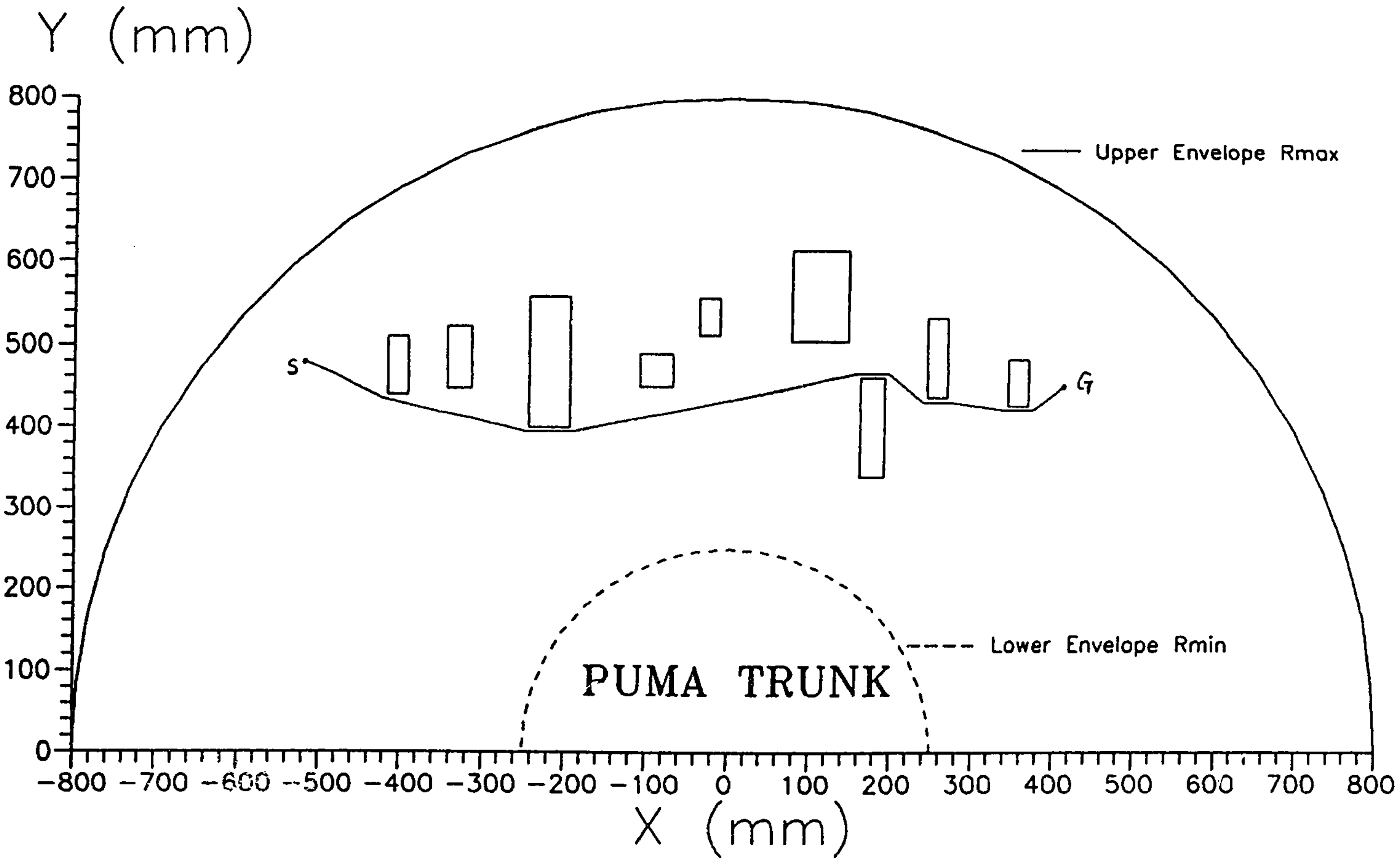
(b)

Figure 5.5. (Continued)



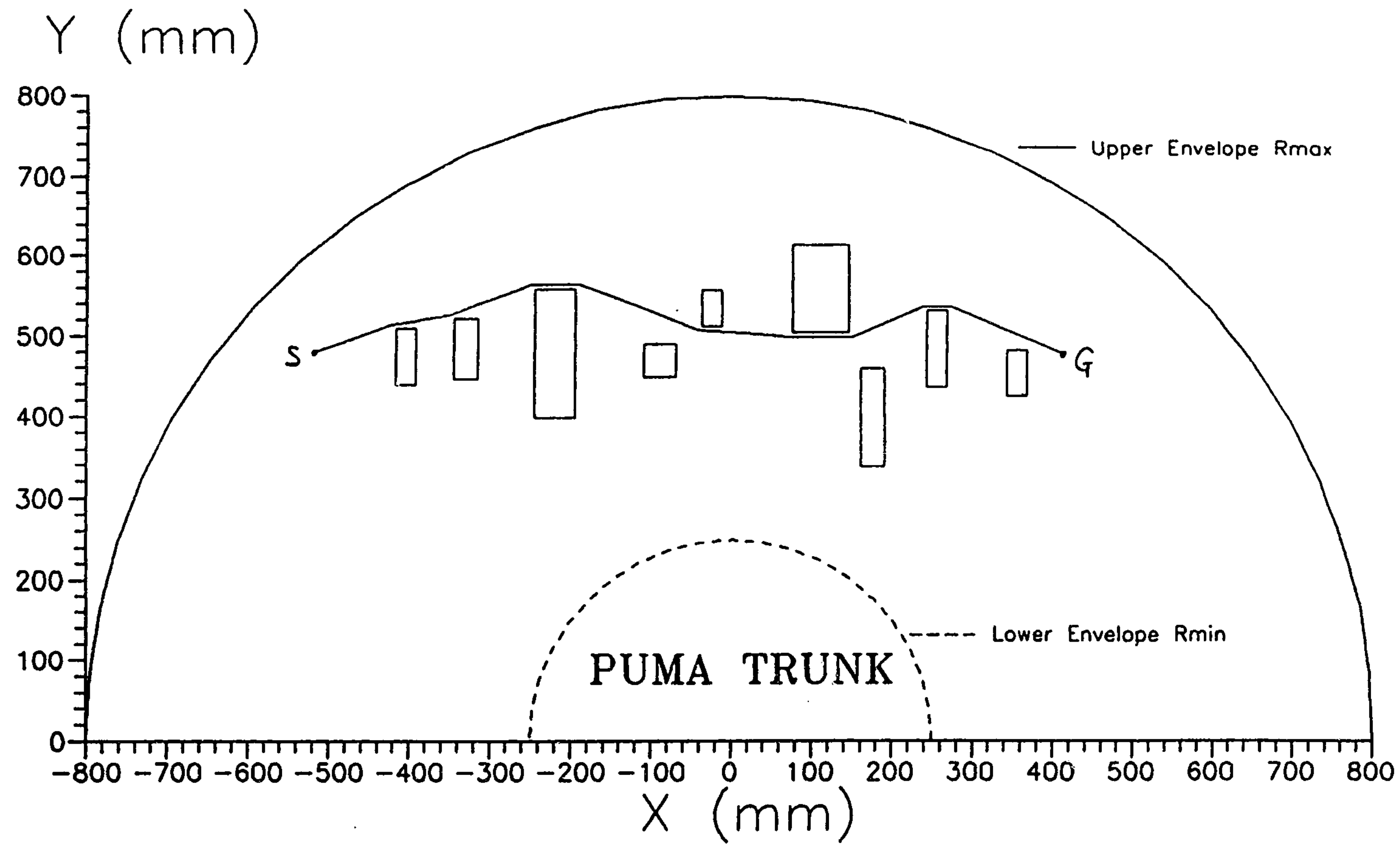
(c)

Figure 5.5. (Continued)



(a)

Figure 5.6. Different collision-free paths generated in the same environment but different start and goal points.



(b)

Figure 5.6. (Continued)

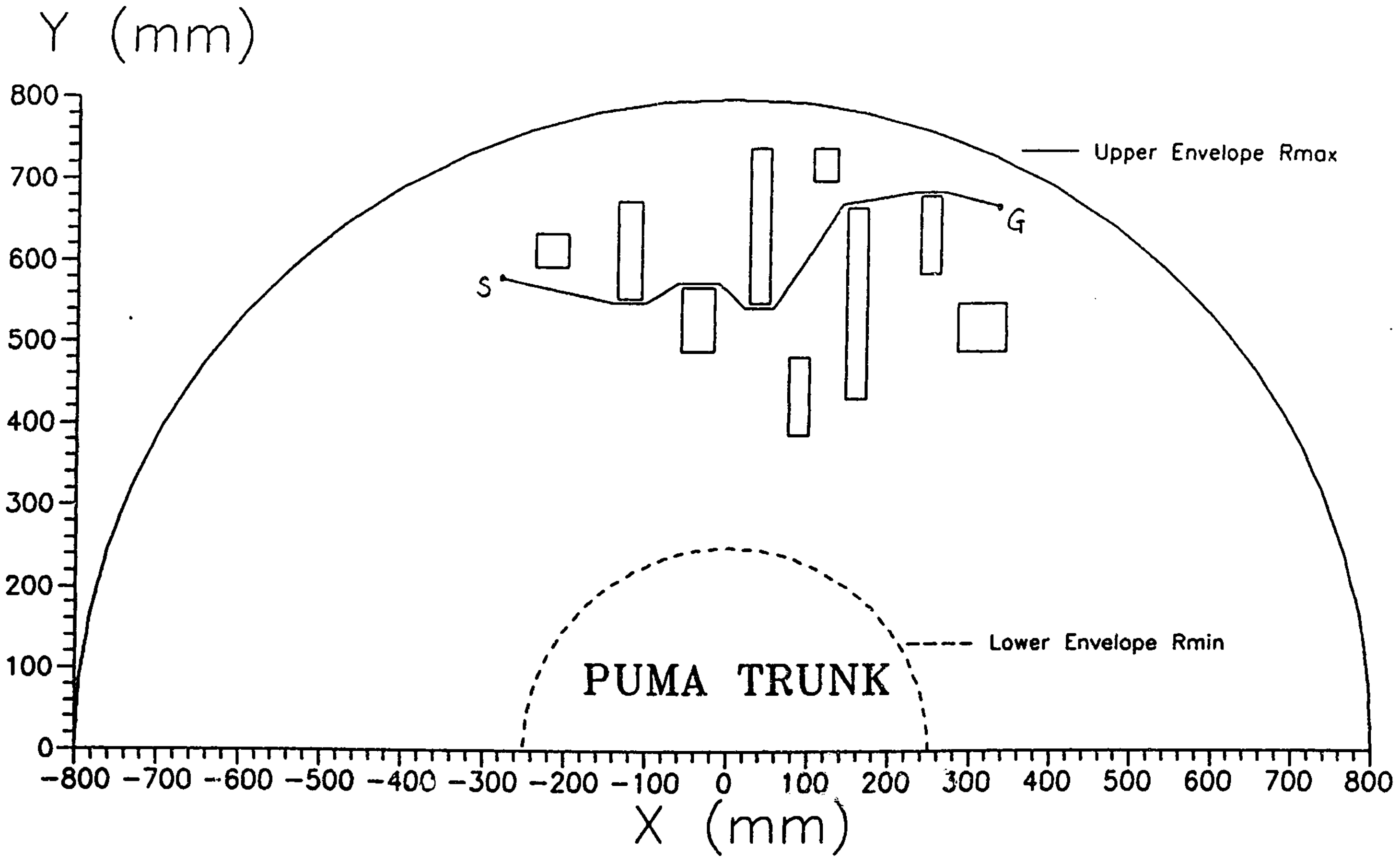


Figure 5.7. A collision-free path generated in a very cluttered environment.

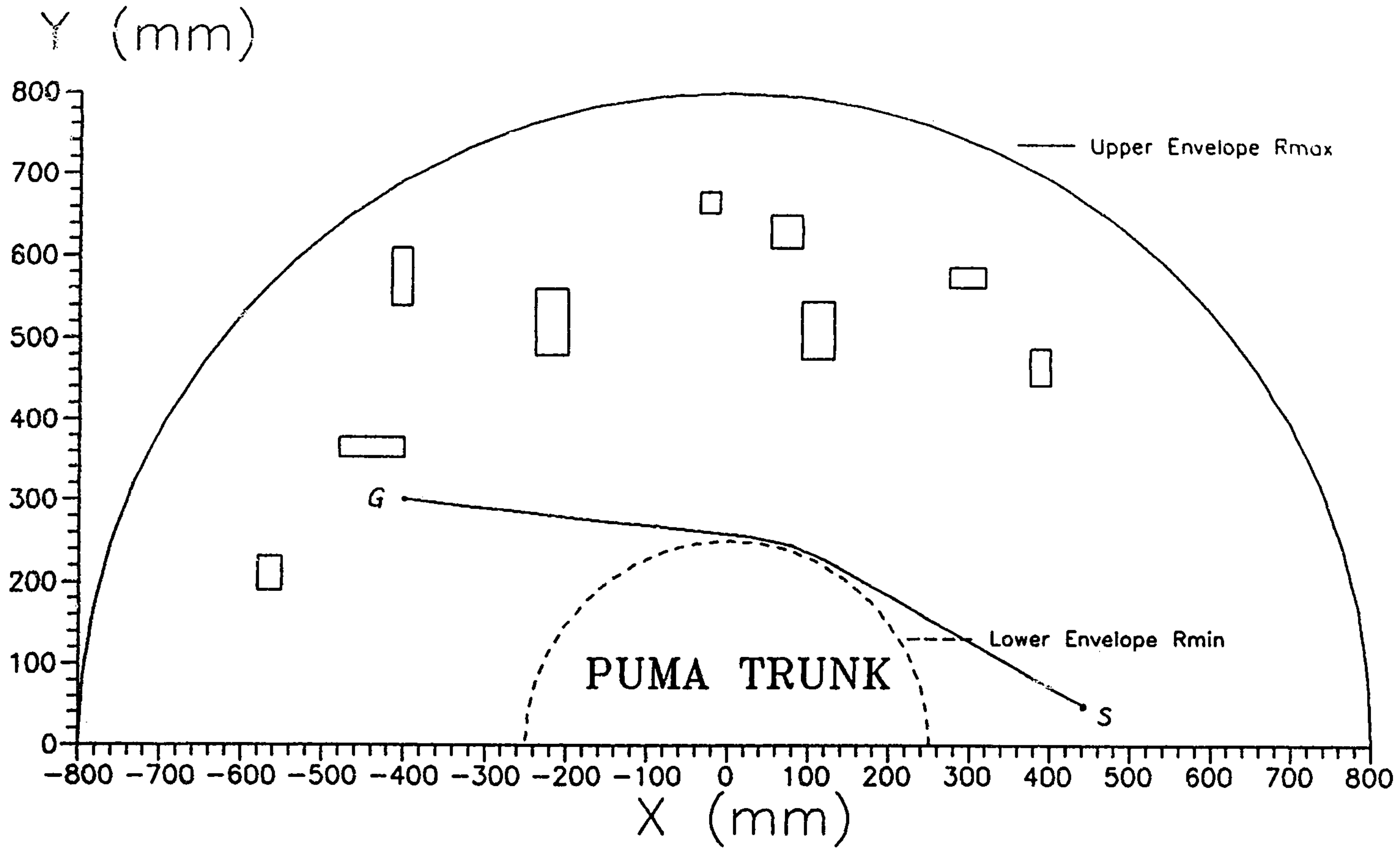
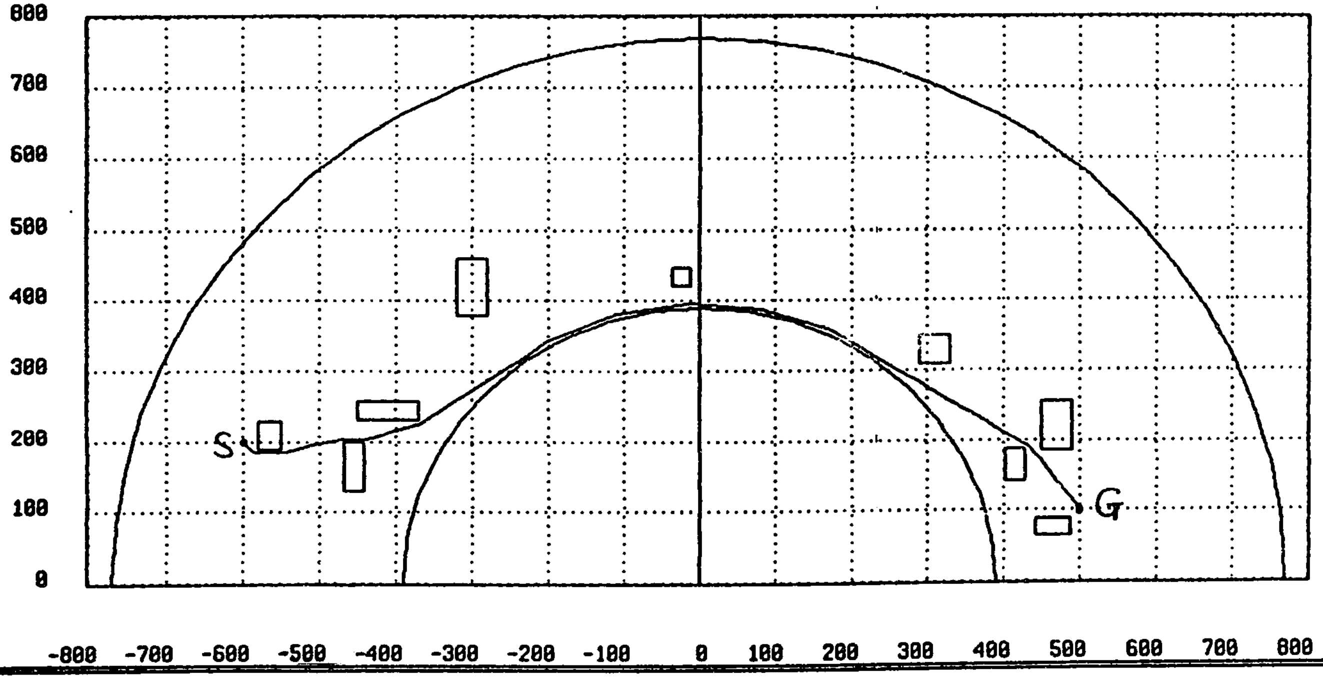
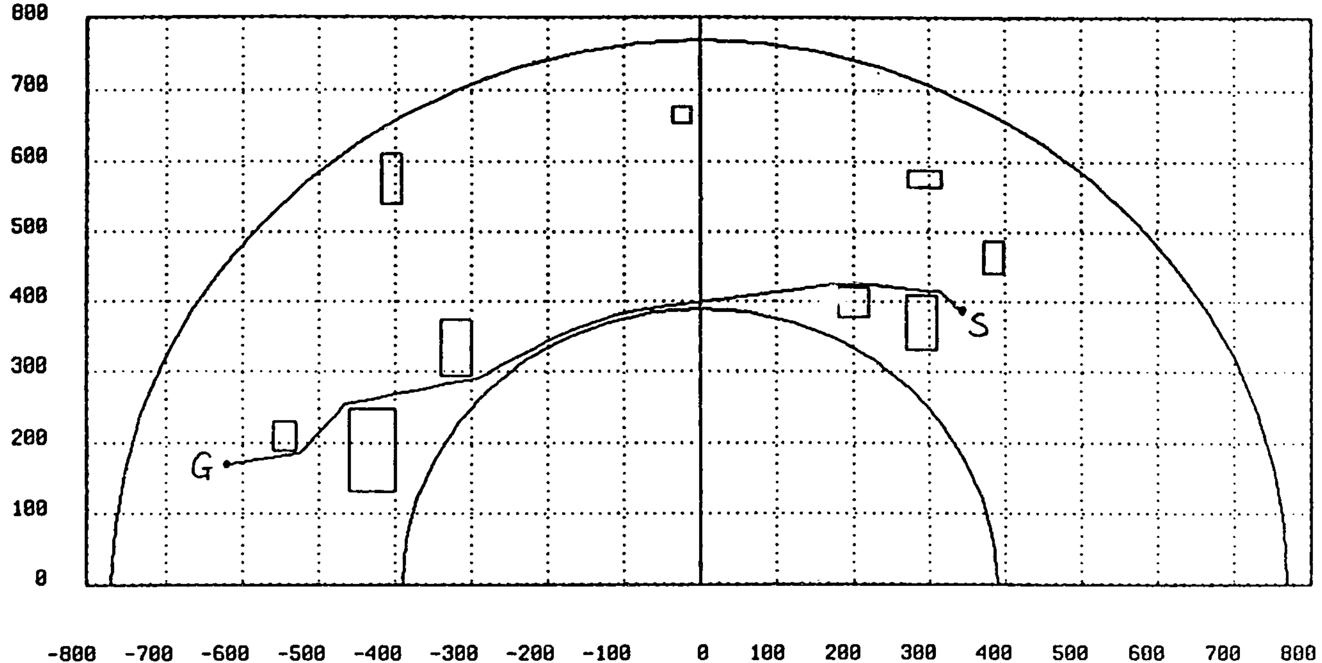


Figure 5.8. A collision-free path generated in a less cluttered environment when the only thing to avoid is the PUMA trunk.



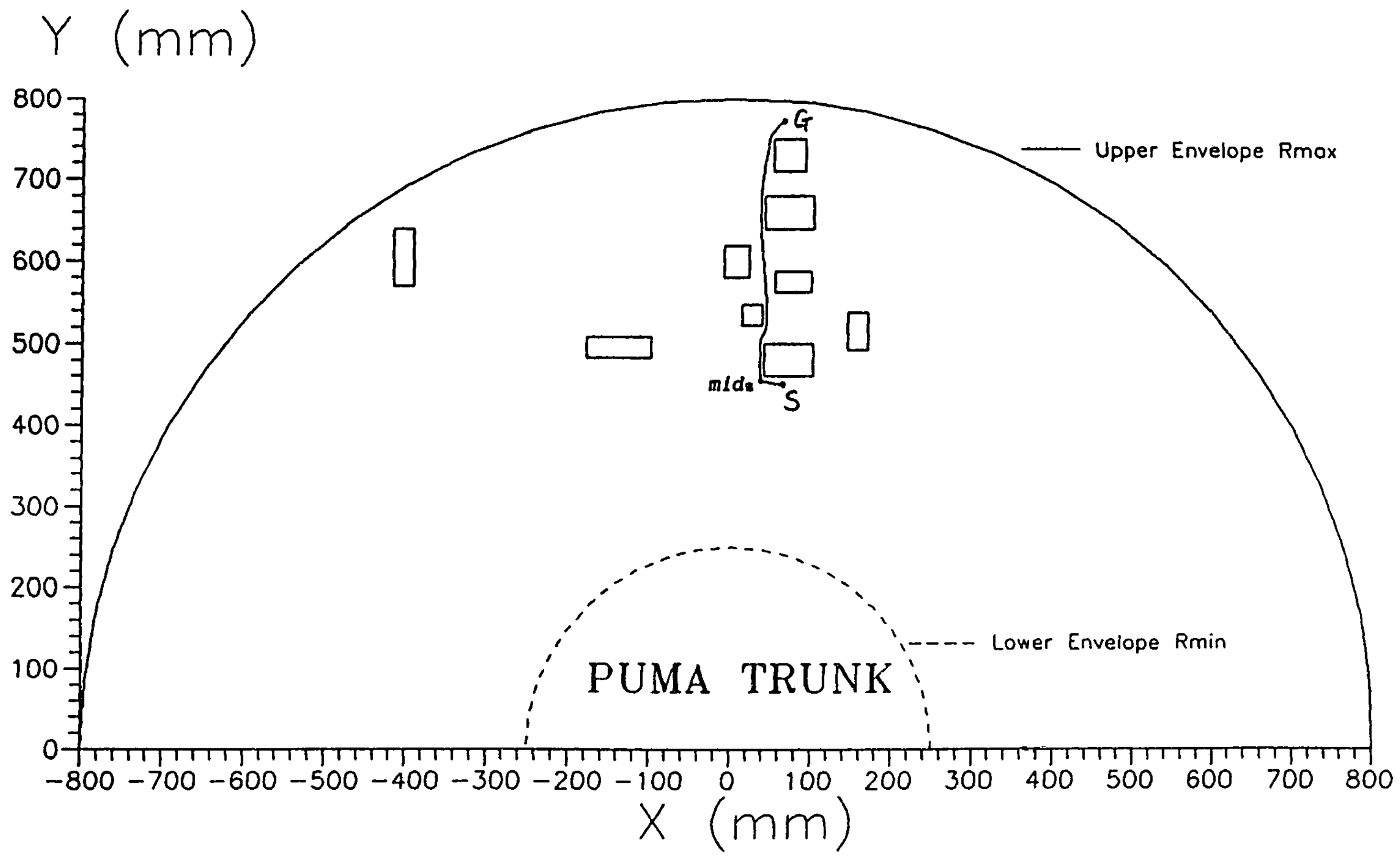
(a)

Figure 5.9. Examples of avoiding the lower bound envelope R_{min}.



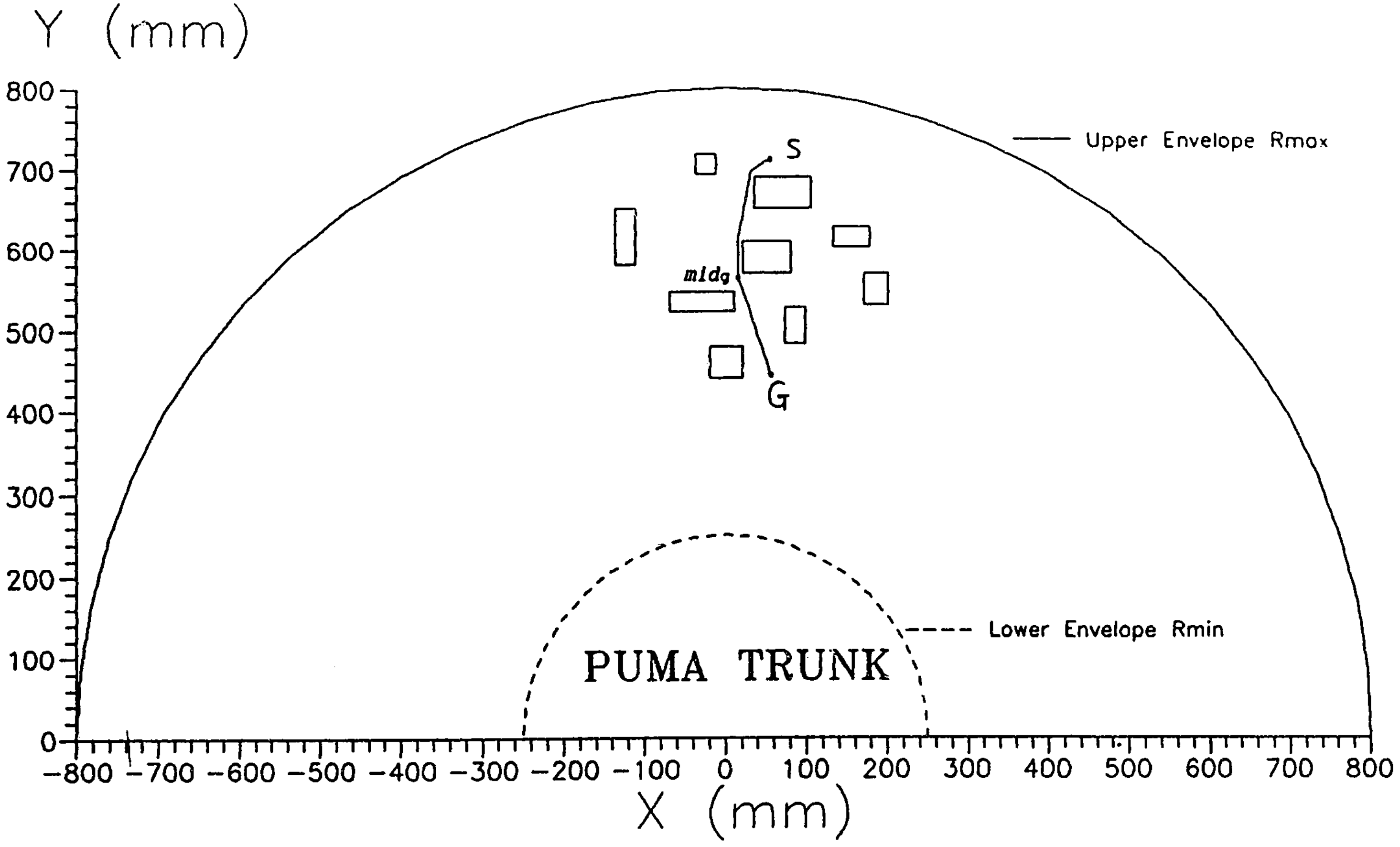
(b)

Figure 5.9. (Continued)



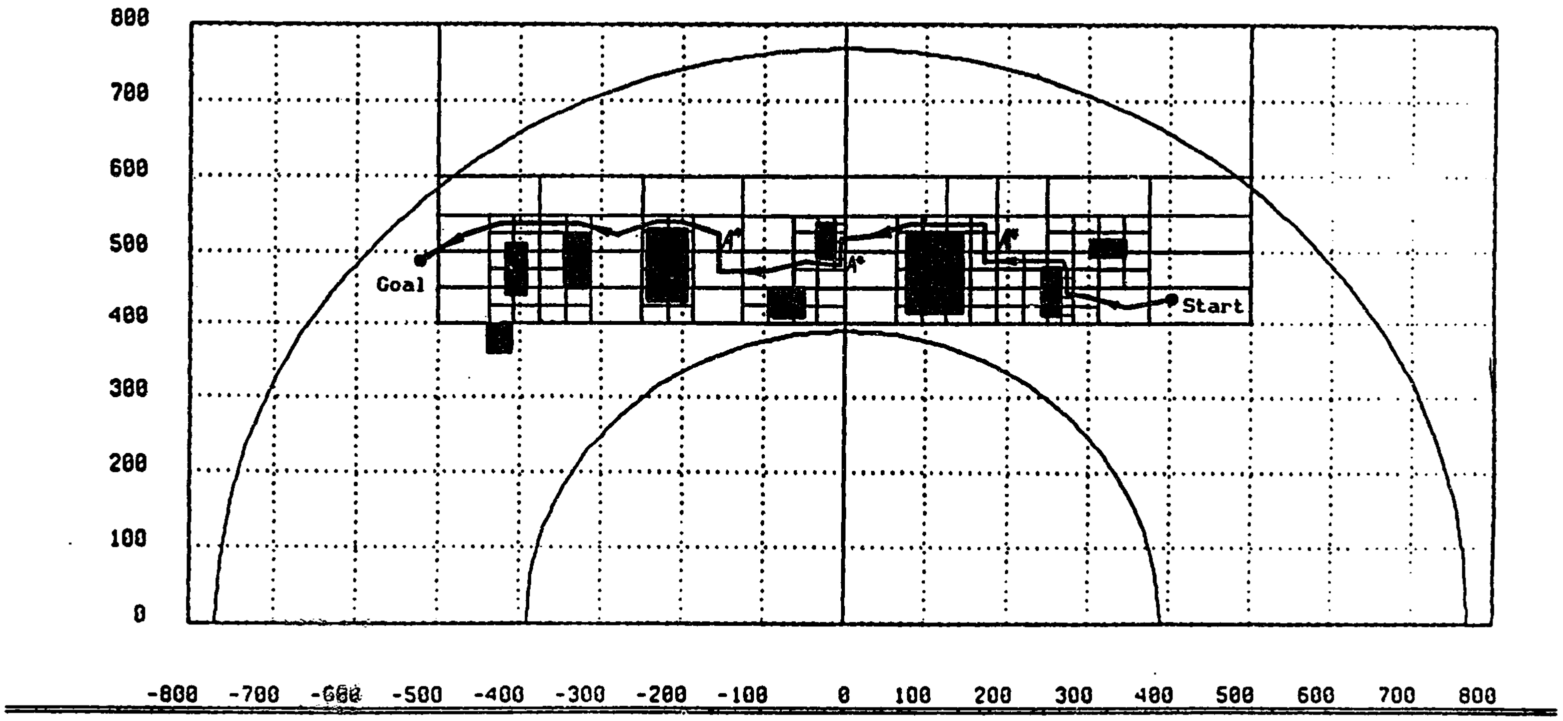
(a)

Figure 5.10. Examples of cases when mid_s and mid_g are required to help the robot find a safe path.



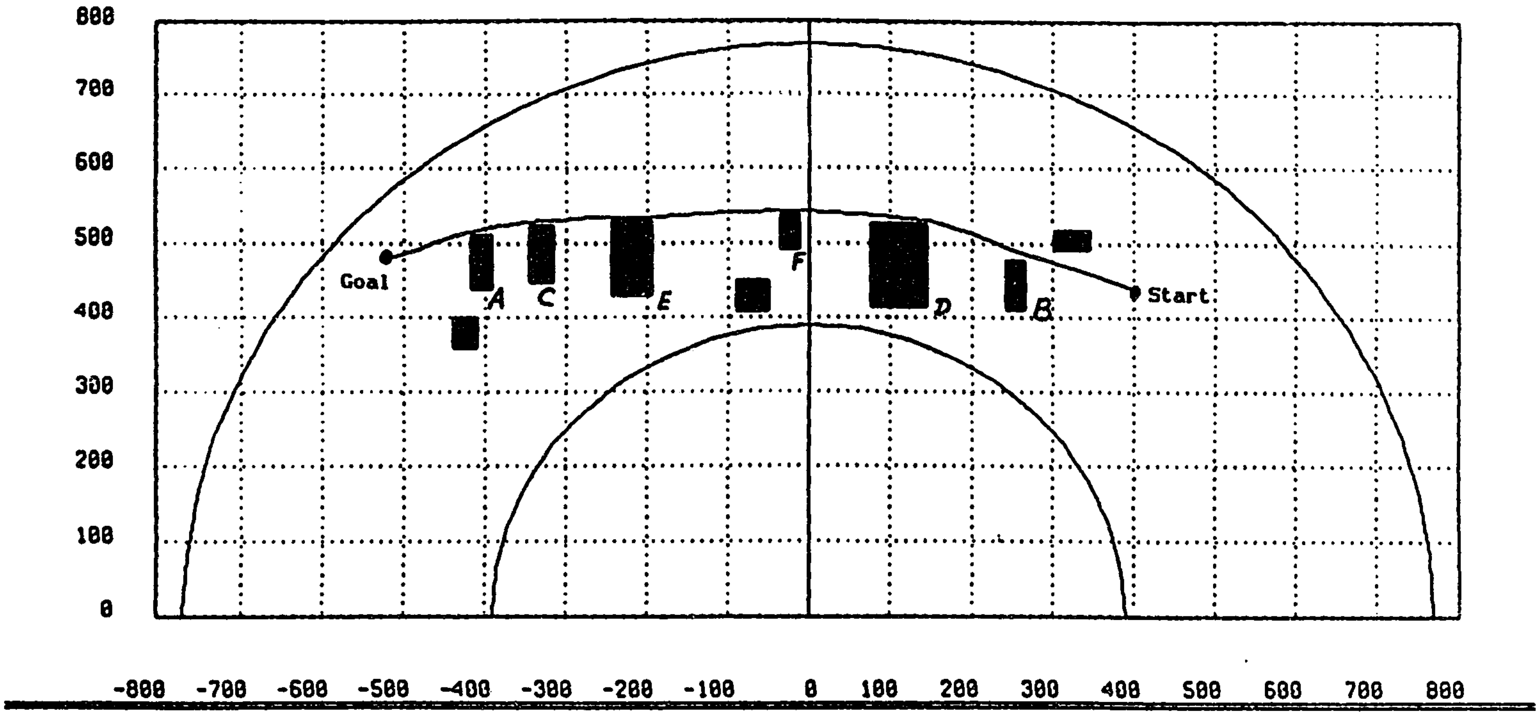
(b)

Figure 5.10. (Continued)



(a)

Figure 5.11. A Comparison with OFS approach: (a) a path generated by OFS, which uses three A^* searches, (b) a path generated by our algorithm with only three local searches.



(b)

Figure 5.11. (Continued)

CHAPTER 6. CONCLUSIONS AND SUGGESTIONS

We have presented in this thesis a new algorithm for collision-free motion planning. This algorithm can find a collision-free path that, in many cases, is the truly optimal path in a two-dimensional work environment. The algorithm is based on the well-known C-space approach and the V-graph search strategy. However, it simplifies the C-space obstacle computations by using rectangular hulls to enclose all workspace obstacles without losing much free space. We also suggest the construction of a simple search graph which has at most six valid nodes in a reduced search space. Thus a locally optimal path in each reduced space can be found easily and efficiently using Bellman's principle of optimality. The global path is a simple connection of all the local optimal paths. This algorithm has been implemented for motion planning of a PUMA 560 manipulator. In this application, we also introduced a method of avoiding the lower bound envelope of the workspace of the PUMA robot.

Our algorithm can be extended directly to three-dimensional path planning for Cartesian type of robots. The optimality of the collision-free paths will then be lost. As far as we know, there are still no general solutions to the find-optimal-path problem in three-dimensional space.

The communication between a supervisory computer and the PUMA robot is via an RS232 serial line at a speed of 9600 baud using DDCMP protocols. DDCMP requires some error checks (such as the CRC checks) and the message sequence formatting which may not be necessary for the simple uploadings of the generated path data to the PUMA. These checks slow down the communication slightly. An alternative way to speed up this process is to send the VAL II commands and data directly from our C program to the PUMA's terminal port as if they were typed from a keyboard. This technique has the disadvantage that no checking of errors is done but it is faster and perhaps suitable for planning and controlling the PUMA motions in a real-time environment from a remote supervisory computer. Details on this are discussed in Section 6.1.

So far we have only proposed an algorithm for collision-free motion planning without considering the manipulator's dynamics. The optimal collision-free paths are those which have the shortest Euclidean distance from start to goal configurations. A minimum-distance path is often not necessarily the minimum-time path. If we can have the control of the robot joint torques, the near minimum-time control strategy that we have discussed in Chapter 3 may be embedded in our algorithm to plan local time-optimal motions in joint space from one position point to the next position point on the collision-free path generated by our present algorithm. This will, however, require a unique inverse kinematics solution for each joint so that the transformation of a hand position in Cartesian space to the joint space could be straight forward. Traveling along a time-minimum path will mean more computations than that of traveling along a shortest distance path by ignoring the dynamics of the robot. More computations will result in a longer path planning time. Whether the time saved in traveling a time-optimal path could be justified by the long period of time wasted in the complicated computations for planning the path is open to some debate.

6.1. Speeding Up The Communication Between The PUMA And The SUN

The PUMA 560's terminal port is also an RS232. We can use this port to connect the PUMA to the SUN 3/160. If we generate all the necessary VAL II commands with carriage return signals as ASCII character strings from our C program and dump them to the terminal port, then the PUMA controller will be able to understand them just as if they were typed from a keyboard. Such a communication requires no DDCMP protocols and error checks and is therefore very fast.

The RS232 port (A or B) can be treated as a (device) file. It is "ttya" for port A or "ttyb" for port B and is stored in directory "robo:/dev" on the SUN. VAL II commands can be written to this file using function "fprintf()" in C. The following are some simple examples.

Suppose we use serial port A of the SUN and we want to send a VAL II command "CA" to the PUMA to calibrate the joint-position sensors in the PUMA and then send another VAL II command "DO READY" to the PUMA in order to move its hand to the home position. We could perform the above as follows:

```
char *ready="DO READY", *ca="CA", cr='\r';
FILE *port_a;
port_a = fopen("/dev/ttya", "w+");     /* Open port A */
fprintf(port_a, "%s%c", ca, cr);
fprintf(port_a, "y\r");        /* Answer YES to confirm */
fprintf(port_a, "%s%c", ready, cr);
fclose(port_a);
```

The path data can also be sent to the PUMA in the same way. For example, we use the VAL II location array variable move4[] (move4[0], move4[1], ···, move4[*k*-1]) to store, in transformation format, the *k* position points of the collision-free path generated by our algorithm. A VAL II command "POINT <location variable> {= <location values>}" can be sent to the PUMA from the C program to perform this. It is like the uploading of path data from the SUN to the PUMA via the supervisor port that we have done using a C-VAL II command "send_to_val()". Suppose the position points of the collision-free path have been generated and stored in a structure array "struct loc points[]" where "struct loc" is defined in (5-16), then we can use the following to transfer those points to the PUMA:

```
FILE *port_a;
char *p1="POINT move4[", *p2="]=", answer[80], cr= '\r';
int i;
port_a = fopen("/dev/ttya", "w+");      /* Open port A */
for (i=0; i<k; i++)
     fprintf(port_a, "%s%d%s%f,%f,%f,%f,%f,%f%c", p1, i,
                 p2, points[i].x, points[i].y, points[i].z,
                 points[i].o, points[i].a, points[i].t, cr);
```

Many of the VAL II commands require a response of answers from the programmer to confirm certain things. Take "POINT" command for example, after the values of X, Y, Z, O, A, T are assigned to a location variable the query "CHANGE?" will appear. Usually our data need not be changed, so we should answer "N" or simply send a carriage return signal. Thus we can rewrite the above loop as

```
for (i=0; i<k; i++)
     {
```

```
fprintf(port_a, "%s%d%s%f,%f,%f,%f,%f,%f%c", p1, i,
          p2, points[i].x, points[i].y, points[i].z,
          points[i].o, points[i].a, points[i].t, cr);
if ((fscanf(port_a, "%s", answer) > 0)
          fprintf(port_a, "%c", cr);
}
```

Once the path data have been transferred to the PUMA we are ready to execute the collision-free motions by running a small VAL II program called "motion" (see Table 5.3.) that has been stored in the VAL II memory before the execution. This is done by sending a VAL II command "EXECUTE" namely:

```
fprintf(port_a, "EXECUTE motion,1,1%c", cr);
```

which executes only once from step 1 of the program.

One thing we should not forget is that there might be risks for doing such a communication without any safety checks, because it may create fatal errors that will either confuse the PUMA controller or crash the whole system. This may be particularly significant in a noisy work environment that has large electric equipment.

REFERENCES

[1] Bellman, R., *Applied Dynamic Programming*, Princeton University Press, 1962

[2] Benson, R. V., *Euclidean Geometry and Convexity*, McGraw-Hill Book Company, 1966

[3] Brady, M., J. M. Hollerbach, T.J. Johnson, T.Lozano-Pérez, and M.T.Mason, Eds., *Robot Motion: Planning and Control*, The MIT Press, Cambridge, Mass., 1982

[4] Brady, M., *Robotics Science: System Development Foundation Benchmark Series*, The MIT Press, Cambridge, Mass., 1989

[5] Branicky, M. S., and W.S. Newman, "Rapid Computation of Configuration Space Obstacles," *IEEE Int. Conf. on Robotics and Automation*, pp.304-310, 1990

[6] Brooks, R. A., "Symbolic reasoning among 3-D models and 2-D images," *Artificial Intelligence*, Vol.17, pp.285-348, 1981

[7] Brooks, R. A., "Solving the Find-Path Problem by Good Representation of Free Space," *IEEE Trans. on Systems, Man and Cybernetics*, Vol.SMC-13, No.13, pp.190-197, 1983

[8] Brooks, R. A., "Planning Collision Free Motions for Pick and Place Operations," *1st Int.Symposium on Robotics Research*, M.Brady and R.Paul, Eds., The MIT Press, Cambridge, Mass., pp.5-37, 1984

[9] Brooks, R. A. and T. Lozano-Pérez, "A Subdivision Algorithm in Configuration Space for Findpath with Rotation," *IEEE Trans. on Systems, Man and Cybernetics*, Vol.15, No.2, 224-233, 1985

[10] Brost, R. C., "Computing Metric and Topological Properties of Configuration-Space Obstacles," *IEEE Int. Conf. on Robotics and Automation*, pp.170-176, 1989

[11] Canny, J. F., *The Complexity of Robot Motion Planning*, Ph.D. Dissertation, The MIT Press, Cambridge, Mass., 1988

[12] Canny, J.F.,"On the "Piano Movers'" Series by Schwartz, Sharir and Ariel-Sheffi," in *The Robotics Review 1*, O. Khatib, J. Craig, and T. Lozano-Pérez, Eds., The MIT Press, 33-40, 1989

[13] Chen, Y., and M. Vidyasagar, "Optimal Trajectory Planning for Planar *n*-Link Revolute Manipulators in the Presence of Obstacles," *IEEE Int. Conf. on Robotics and Automation*, pp.202-208, 1988

[14] Connell, I., *Modern Algebra: A Conceptive Introduction*, North Holland, New York, NY, 1982

[15] "DDCMP Specification, Version 4.0", Digital Equipment Corporation, March 1978

[16] Donald, B. R., "A Search Algorithm for Motion Planning with Six Degrees of Freedom," *Artificial Intelligence*, Vol.31, pp.295-353, 1987

[17] Elgazzar, S., D. Green and D. O'Hara, "A Vision-Based Robot System Using A Multiprocessing Controller," Tech. Report, Division of Elec. Eng., NRC Canada, June 1984

[18] Eltimsahy, A. H., and W.S. Yang, "Near Minimum-Time Control of Robotic Manipulator with Obstacles in the Workspace," *IEEE Int. Conf. on Robotics and Automation*, pp.358-363, 1988

[19] Faverjon, B., "Obstacle Avoidance Using An Octree in the Configuration Space of a Manipulator," *IEEE Int. Conf. on Robotics*, pp.504-512, 1984

[20] Faverjon, B., "Hierarchical Object Models for Efficient Anti-Collision Algorithms," *IEEE Int. Conf. on Robotics and Automation*, pp.333-340, 1989

[21] Fletcher, R. W., and A. A Goldenberg, "Collision Avoidance for Robot Manipulators: Application to CATIA/IBM 7565 Interface," *J. of Robotic Systems*, 5(2), pp.125-146, 1988

[22] Fu, K. S., R. C. Gonzale and C. S. G. Lee, *Robotics: Control, Sensing, Vision, and Intelligence*, McGraw-Hill Book Company, 1987

[23] Fujimura, K., and H. Samet, "Motion Planning in a Dynamic Domain," *IEEE Int. Conf. on Robotics and Automation*, pp.324-330, 1990

[24] Ge, Q. J., and J. M. McCarthy, "Equations for Boundaries of Joint Obstacles for Planar Robots," *IEEE Int. Conf. on Robotics and Automation*, pp.164-169, 1989

[25] Ge, Q. J., and J. M. McCarthy, "An Algebraic Formulation of Configuration-Space Obstacles for Spatial Robots," *IEEE Int. Conf. on Robotics and Automation*, pp.1542-1547, 1990

[26] Glavina, B., "Solving Findpath by Combination of Goal-Directed and Randomized Search," *IEEE Int. Conf. on Robotics and Automation*, pp.1718-1723, 1990

[27] Gondran, M., *Graphs and Algorithms*, John Wiley & Sons, New York, NY, 1984

[28] Gourishankar, V. G., "Lecture Note: Optimal Control Systems," Dept. of Elec. Eng., Univ. of Alberta, 1986

[29] Gupta, K. K., "Fast Collision Avoidance for Manipulator Arms: A Sequential Search Strategy," *IEEE Int. Conf. on Robotics and Automation*, pp.1724-1729, 1990

[30] Hasegawa, T., "Collision Avoidance using Characterized Description of Free Space," *Proc. of* '85 *ICAR*, pp.69-76, Tokyo, 1985

[31] Hayward, V., "Fast Collision Detection Scheme by Recursive Decomposition of A Manipulator Workspace," *IEEE Int. Conf. on Robotics and Automation*, pp.1044-1049, 6

[32] Herman, M., "Fast, Three-Dimensional, collision-Free Motion Planning," *IEEE Int. Conf. on Robotics and Automation*, pp.1056-1063, 1986

[33] Ilari, J., and J. L. Reyna, "Some Experimental Results Using Heuristics for Solving the Find-Path Problem in C-space," in *IFAC Theory of Robotics*, pp.337-342, Vienna, Austria, 1986

[34] Izaguirre, A., "Implementing Remote Control of A Robot Using the VAL II Language," Tech. Report, Univ. of Pennsylvania, Aug. 1984

*[35] Jones, J. L., and T. Lozano-Pérez, "Planning Two-Fingered Grasps for Pick-and Place Operations on Polyhedra," *IEEE Int. Conf. on Robotics and Automation*, pp.683-688, 1990

[36] Jun, S., and K. G. Shin, "A Probabilistic approach to Collision-Free Robot Path Planning," *IEEE Int. Conf. on Robotics and Automation*, pp.220-225, 1988

[37] Jun, S., and K.G. Shin, "Shortest Path Planning with Dominance Relation," *IEEE Int. Conf. on Robotics and Automation*, pp.138-143, 1990

[38] Kanayama, Y., "Least Cost Paths with Algebraic Cost Functions: Part I," *IEEE Int. Conf. on Robotics and Automation*, pp.75-80, 1988

[39] Kant, K., and S. Zucker, "Planning Collision-Free Trajectories in Time-Varying Environments: A Two-Level Hierarchy," *IEEE Int. Conf. on Robotics and Automation*, pp.1644-1649, 1988

[40] Khatib, O., "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots," *The Int. J. of Robotics Research*, Vol.5, No.1, pp.90-98, 1986

[41] Kheradpir, S., and J. S. Thorp, "Real-Time Control of Robot Manipulators in the Presence of Obstacles," *IEEE J. of Robotics and Automation*, Vol.4, No.6, pp.687-698, 1988

[42] Kim, B. K., and K. G. Shin, "Suboptimal Control of Industrial Manipulators with a Weighted Minimum Time-Fuel Criterion," *IEEE Trans. on Automatic Control*, Vol-30, No.1, pp.[illegible]-10, 1985

[43] Koivo, A. J., and G. A. Bekey, "Report of Workshop on Coordinated Multiple Robot Manipulators: Planning, Control, and Applications," *IEEE J. of Robotics and Automation*, Vol.4, No.1, pp.91-93, Feb. 1988

*[44] Kuan, D. T., J. C. Zamiska and R. A. Brooks, "[illegible] Decomposition of Free Space for Path Planning," *IEEE [illegible] Conf. on Robotics and Automation*, pp.168-173, 1985

[45] Kyriakopoulos, K. J., "Minimum Jerk Path Generation," *IEEE Int. Conf. on Robotics and Automation*, pp.364-369, 1988

[46] Lee, D. T., and R.L. Drysdale, III, "Generalization of Voronoi Diagrams in The Plane," *SIAM J. of Comput.*, Vol.10, No.1, pp.73-87, 1981

*[47] Lee, D.T., and F.P. Preparata, "Euclidean Shortest Path in the Presence of Rectilinear Barriers," *Networks*, Vol.14, pp.393-410, 1984

[48] Lozano-Pérez, T., and M. A. Wesley, "An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles," *Communications of the ACM*, Vol.22, No.10, pp.560-570, 1979

[49] Lozano-Pérez, T., "Automatic Planning of Manipulator Transfer Movements," *IEEE Trans. on Systems, Man and Cybernetics*, Vol.SMC-11, No.10, pp.681-698, 1981

[50] Lozano-Pérez, T., "Spatial Planning: A Configuration Space Approach," *IEEE Trans. on Computers*, Vol.C-32, No.2, pp.108-120, Feb. 1983

*[51] Lozano-Pérez, T., "A Simple Motion-Planning Algorithm for General Robot Manipulators," *IEEE J. of Robotics and Automation*, Vol.RA-3, No.3, pp.224-238, 1987

*[52] Lozano-Pérez, T., J. L. Jones, E. Mazer, and P.A. O'Donnell, "Task-Level Planning of Pick-and-Place Robot Motions," *Computers*, pp.21-29, March 1989

[53] Meagher, D., "Geometric Modeling Using Octree Encoding," *Computer Graphics and Image Processing*, Vol.19, 129-147,1982

[54] Meng, A. C., "Dynamic Motion Replanning for Unexpected Obstacles," *IEEE Int. Conf. on Robotics and Automation*, pp.1848-1849, 1988

[55] Meyer, W., "Path Planning and the Geometry of Joint Space Obstacles." *IEEE Int. Conf. on Robotics and Automation*, pp.215-219, 1988

[56] Muck, K. L., "Motion Planning in Constrain Space," *IEEE Int. Conf. on Robotics and Automation*, pp.633-635, 1988

[57] Nilsson, N. J., *Problem-Solving Methods in Artificial Intelligence*, McGraw-Hill Book Company, 1971

[58] Nilsson, N. J., *Principles of Artificial Intelligence*, Tioga Publishing Co., Palo Alto, CA, 1980

[59] Ó'Dúnlaing, C., M. Sharir, and C. K. Yap, "Retraction: A New Approach to Motion-Planning," *Proc. of the 15th Symposium on the Theory of Computing*, 1983

[60] Ó'Dúnlaing, C., and C. K. Yap, "A Retraction Method for Planning the Motion of a Disc," *J. of Algorithms*, March 1985

*[61] Ozaki, H., and A. Mohri, "Synthesis of a Minimum-time Manipulator Trajectories with Geometric Path Constraints using Time Scaling," *Robotica*, Vol.6, pp.41-46, 1988

[62] Paden, B., A. Mees and M. Fisher, "Path Planning Using a Jacobian-Based freespace Generation Algorithm," *IEEE Int. Conf. on Robotics and Automation*, pp.1732-1737, 1989

*[63] Papadimitriou, C. H., "An Algorithm for Shortest-Path Motion in Three Dimensions," *Information Processing Letters*, Vol.20, pp.259-263, 1985

[64] Paul, R. P., *Robot Manipulators: Mechanics, Programming, and Control*, The MIT Press, Cambridge, Mass., 1981

[65] Samet, H., "Neighbor Finding Techniques for Images Represented by Quadtrees," *Computer Graphics and Image Processing*, Vol.18, pp.37-57, 1982

[66] Schwartz, J., J. Hopcroft and M. Sharir, Eds., *Planning, Geometry and Complexity of Robot Motion*, Ablex Publishing Corp., Norwood, NJ, 1987

[67] Sharir, M., "Algorithmic Motion Planning in Robotics," *Computers*, pp.9-20, March 1989

[68] Shihari, Y., *Artificial Intelligence: Concepts, Techniques and Applications*, John Wiley & Sons, 1984

[69] Shiller, Z., and S. Dubowsky, "Global Time Optimal Motions of Robotic Manipulators in The Presence of Obstacles," *IEEE Int. Conf. on Robotics and Automation*, pp.370-375, 1988

[70] Shimano, B. E., C.C. Geschke and C.H. Spalding III, "VAL-II: A New Robot Control System for Automatic Manufacturing," *IEEE Int. Conf. on Robotics and Automation*, pp.278-292, 1984

[71] Singh, J. S., and M. D. Wagh, "Robot Path Planning using Intersecting Convex Shapes: Analysis and Simulation," *IEEE J. of Robotics and Automation*, Vol.RA-3, No.2, 101-108, 1987

[72] Srihari, S. N., "Representation of Three-Dimensional Digital Images," *Computing Surveys*, Vol.13, No.4, Dec. 1981

[73] Suh, S., and K. G. Shin, "A Variational Dynamic Programming Approach to Robot-Path Planning With a Distance-Safety Criterion," *IEEE J. of Robotics and Automation*, Vol.4, No.3, pp.334-349, 1988

[74] Suhand, S., and A. B. Bishop, "Collision-Avoidance Trajectory Planning Using Tube Concept: Analysis and Simulation," *J. of Robotic Systems*, 5(6), pp.497-525, 1988

[75] Takahashi, O., and R.J. Schilling, "Motion Planning in a Plane Using Generalized Voronoi Diagrams," *IEEE Trans. on Robotics and Automation*, Vol.5, No.2, pp.143-150, April 1989

[76] Udupa, S.M., "Collision Detection and Avoidance in Computer Controlled Manipulators," Ph.D. Dissertation, California Inst. of Tech., Pasadena, Calif., 1977

*[77] Udupa, S. M., "Collision Detection and Avoidance in Computer Controlled Manipulators," *Proc. 5th Int. Joint Conf. Artificial Intelligence*, MIT, 1977

[78] "Unimate PUMA Mark II Robot: 500 Series Equipment Manual", Unimation Inc., Danbury, Connecticut, 1985

[79] "Unimate Industrial Robot: VAL-II Programming Manual Version 2.0," Unimation Inc., Danbury, Connecticut, 1986

[80] Warren, C. W., "Global Path Planning Using Artificial Potential Fields," *IEEE Int. Conf. on Robotics and Automation*, pp.316-321, 1989

[81] Welzl, E., "Construction the Visibility Graph for n-Line Segments in $O(n^2)$ Time," *Information Processing Letters*, Vol.20, pp.167-171, 1985

[82] Wong, E. K., and K. S. Fu, "A Hierarchical-Orthogonal-Space Approach to Collision-Free Path Planning," *IEEE Int. Conf. on Robotics and Automation*, pp.506-511, 1985

[83] Wong, E. K., and K. S. Fu, "A Hierarchical Orthogonal Space Approach to Three-Dimensional Path Planning," *IEEE J. of Robotics and Automation*, Vol.RA-2, No.1, pp.42-53, 1986

[84] Yap, C. K., "Algorithmic Motion Planning," in *Advances in Robotics*, vol.1, J. T. Schwartz and C. K. Yap, Eds., Lawrence Erlbaum Associates, Hillsdale, NJ, pp.95-143, 1987

[85] Zhu, D., and J. Latombe, "Constraint Reformation in a Hierarchical Path Planner," *IEEE Int. Conf. on Robotics and Automation*, pp.1918-1923, 1990

[86] Fu, L., and D. Liu, "An Efficient Algorithm for Finding a Collision-free Path Among Polyhedral Obstacles," *J. of Robotic Systems*, 7(1), pp.129-137, 1990

[87] Gewali, L. P., S. Ntafos and I. G. Tollis, "Path Planning in the Presence of Vertical Obstacles," *IEEE Trans. on Robotics and Automation*, Vol.6, No.3, pp.331-341, June 1990

[88] Noborio, H., T. Naniwa and S. Arimoto, "A Quadtree-Based Path-Planning Algorithm for a Mobile Robot," *J. of Robotic Systems*, 7(4), pp.555-574, 1990

Note: "*" indicates that the publication is not specifically referred to in the thesis but it may be of some use for further readings.

APPENDICES

1. Direct Control Of The PUMA From The SUN

The program for direct control of the PUMA from the SUN is stored in "robo:/usr/ee/jill/ddcmp2/super.x". "super.x" is compiled from two C routines "super_6m.c" (contains subroutines of communication via the supervisor port with VAL II) and "super_6d.c" (contains subroutines of the DDCMP protocols and CRC error checks) with a small header file "super_4i.c" included in "super_6d.c". After executing this program there is a "menu" that permits the following operations:

(1) "start_DDCMP": Starts the DDCMP communication. In VAL II we should write "enable network" in order to start the communication using the protocol. A dot appears after this command if the communication works correctly. If a VAL II command "enable supervisor" is written at the terminal, then the message "prompt_VAL II=>" will appear on the SUN indicating that it is ready to send commands to VAL.

(2) "start_terminal": Is similar to "start_DDCMP" with the difference that both of the commands "enable network" and "enable supervisor" are written in VAL II by the terminal. Thus the SUN will work as a dumb terminal.

(3) "abort_DDCMP": Stops the DDCMP program.

(4) "short_status": Shows the actual status of the VAL II system. Such as program running state, arm calibration, etc.

(5) "abort_status": Stop the running of "short_status".

(6) "ask_state": Writes the state of the DDCMP (number of messages received and sent, last message received, etc.)

2. C Like Functions Of VAL II (C-VAL II)

C-VAL II functions are similar to the VAL II commands. They can be called from an archived library, "libpuma.a", which also contains DDCMP routines. There are already more than one hundred C-VAL II commands that are stored in "libpuma.a". Many user needed VAL II commands can also be written in C and appended to "libpuma.a". This is a very important library that we link (in the "makefile") with our application routines for planning collision-free motions for the PUMA 560. The following is a brief description of this library.

The messages received and sent to VAL II consist of a byte corresponding to the logical unit, a byte corresponding to the function code, two bytes corresponding to the function qualifier (error function received), and the message data. There are seven primitives functions of communication in "libpuma.a":

(1) "send_wait(char message[])": Sends a message and waits until it is actually sent.

(2) "send_no_wait(char message[], int *error)": Puts the message in a stack. If the number of messages is equal to or larger than the maximum number that can be stored in the stack then *error=-1 else *error=0.

(3) "receive_wait(int number_msg, char *tableau_msg[])": Waits until the number of messages received is equal to number_msg.

(4) "receive_no_wait(int number_msg, char *tableau_msg[], int *error)": Similar to "receive_wait". The difference is that it does not wait until it has received a certain number of messages. If the number of messages received is less than number_msg then *error=-1 and

the messages will not be read, else *error=0 and the messages will be read.

(5) "number_sent(int *number_msg)": Returns in *number_msg the number of messages sent.

(6) "number_received(int *number_msg)": Returns in *number_msg the number of messages received.

(7) "delete_msg(int number_msg, int *error)": Deletes the number of messages that is equal to number_msg if there are at least this number of messages received (*error=0), else the messages will be kept and *error=-1.

These primitives are frequently called by the many different C-VAL II functions that are used to control the PUMA motions.

In each C_VAL II command (or function) there are many different parameters that pass information. Among them are two very important ones. These two are used in almost all C-VAL II functions.

(a) char *quality: which indicates if the command is going to be sent by waiting the answer (*quality="wait"), or without waiting for an answer (*quality="nowait"). In the latter case the messages are put in a stack and sent sequentially, and the answers are automatically stored in another stack and the programmer has the responsibility to read them. From our experience, it is always better to wait for the answer, i.e. set *quality="wait" in each C-VAL II function, so that the next function can be executed properly.

(b) int *error: which indicates the error code that will be sent out. *error=0 means successful, *error=-1 means not successful.

Of all the C-VAL II functions, two are the most important for initializing and terminating communications and should be included in each application program.

(1) start_supervisor(error)

int *error;

Permits the initialization of the DDCMP communication with the VAL II controller so that our C-VAL II commands can then be accepted by the PUMA. If VAL II has not started the communication, i.e. Step 1—3 mentioned in Section 5.3.1 of Chapter 5. have not been done, or another person is using the supervisor port, then this function will be blocked and the value returned in "error" is -1. A successful communication returns an error code of 0.

(2) abort_supervisor(error)

int *error;

Terminates and released the DDCMP communication with the PUMA from a C-VAL II program. If it finds the line is not occupied then an error code of -1 will be returned. Otherwise "error" is 0 and the termination is successful.

Most of the VAL II (and hence the C-VAL II) commands are not quite useful to our motion planning tasks. The ones that are specially made for our application programs to implement the motion planning algorithm that we have discussed in Chapter 4 on the PUMA 560 robot are simply the following:

— start_supervisor(int *error): initializes DDCMP communication;

— ready_v(char *quality, int *error): moves the PUMA to its home position as described in (5-4);

— deletel_v(char *data, char *quality, int *error): deletes all location points in transformation format stored in the VAL II location variable name "data";

— send_to_val(struct location SUNpoints[], int number, char *data): sends "number" location points (in transformation format) generated in the SUN to the VAL II memory and stores them in the VAL II location variable name "data"; (This function does not exist in VAL II.)

— move_v(struct location *point, char *quality, int *error): moves the PUMA using joint interpolation to a location point, described by "*point", which should be defined in transformation format as in (5-5), i.e.,

point->quality="location";

— moves_v(struct location *point, char *quality, int *error): similar to move_v() but moves the robot in a straight line fashion to "*point";

— execute_v(char *prog, int loops, int step, char *quality, int *error): executes a VAL II program a number of times equal to the value of "loops" beginning at a step number "step";

— where_v(char *quality, int *error): displays the location and orientation of the PUMA hand in both transformation and precision point formats.

— abort_supervisor(int *error): terminates DDCMP communication;

Before we run any of the above functions, Step 1—3 in Section 5.3.1. must be first performed. Step 2—3 can also be included in an auto-start program in VAL II so that each time we start the PUMA controller this auto-start program will be immediately executed. This will ensure that the PUMA is ready to accept remote controls from the supervisor computer, the SUN workstation. A sample auto-start program called "auto" is given in Table A5.1. of Appendix 5.

3. List Of All Path Planning Routines For The PUMA 560 And The "makefile"

The application routines for planning the PUMA 560 motions are stored in directory "robo:/usr/ee/jill/avoid". The following is a list of all C routines that are compiled together to generate our application program "av".

int avoid_env(): avoids the lower bound of the PUMA 560 envelope R_{min} and returns the number of points generated for avoiding R_{min}.

void bubble_sort(): performs bubble sorts for rearranging valid obstacles in the reduced space $\Omega^i(CO_v^i)$.

double angle(), norm(), radius(): these are frequently used routines for calculating angles, norms and radii.

void find_mid(): finds mid_s, mid_g, mid_1 and mid_2 parameters.

int genloc(): generates an optimal path and returns the total number of points generated for the path.

int genloc2(): generates an alternate path for avoiding R_{min} and returns the number passed from avoid_env().

void genloc3(): calculates the local start and goal points, $start^i$ and $goal^i$ for the ith search layer.

int genloc6(), genloc8(), genloc9(), genloca(), genlocb(): these are subroutines of genloc().

int line_path(): generates a single line path when no obstacles are found between start and goal, and returns the number of points generated.

int line_path2(): same as line_path() but is only used when R_{min} is found between start and goal points and has to be avoided.

int line_seg(): generates line-segments for each portion of a local optimal path and returns the number of points generated for the local path These line-segments are connected together to form the final suboptimal (global) path.

int line_seg2(), line_seg3(),line_seg6(),line_seg8(),line_seg9(), line_sega(): these are subroutines of line_seg().

void listall(): lists all generated position points on screen.

void listdata(): lists or displays the input data.

void main(): this is the main routine of the program.

int ne_sw(): determines all valid vertices *VERT*v+ for NE-SW-ward or SW-NE-ward traveling direction, and performs a graph search among these vertices using Bellman's principle of optimality.

int nw_se(): determines all valid vertices *VERT*v- for NW-SE-ward or SE-NW-ward traveling direction, and performs a graph search among these vertices using Bellman's principle of optimality.

int obs_test(): tests if an obstacle is within the reduced space Ω^0 defined by global start and goal points, $start^0$ and $goal^0$, and returns the number of obstacles in Ω^0.

void one_obs_test(): tests if an obstacle is between two valid vertices that should be connected.

void path(), pathmain(): these two routines are actually the "execute_motion()" in Table 5.3.

int readp(): reads an already generated path form file "path.rec" on the hard disk, and returns the number of position points read.

void send_to_val(): uploads the generated position points in transformation format to the VAL II memory.

void plotpath(): plots the generated path and the environment on a suntool window using SunCGI graphics functions.

void plotmain(), rddata(), scale(), form_arr(), prt_vscle(), prt_hscle(), get_scale(), draw_x_axis(), draw_y_axis(): these are subroutines of plotpath().

void rdata(): reads input data file from hard disk.

int start_goal_nonobs(), start_goal_nonobs2, start_goal_nonobs3(): these routines are used to test if there are any obstacles between local start and goal points, $start^i$ and $goal^i$ ($i > 0$). If not then a single line path will be generated to connect $start^i$ and $goal^i$.

void start_goal_test(): tests if start and/or goal points are outside of the PUMA workspace. If true then it sends out an error message that no safe path exists and terminates the program.

int slope_test(): decides the traveling direction of the path in order to determine if $VERT_v = VERT_{v+}$ or $VERT_v = VERT_{v-}$.

void sort_avo(), sort_avo2(): these are routines of sorting the valid obstacles for graph searching purposes.

void store_obs(), store_obs2(): transfer a valid obstacle variable to another place also for graph searching purposes.

int testloc(): tests if all the position points generated for the collision-free path are within the PUMA 560's physical reach.

void wfile(): writes the generated path to file "path.rec".

The above routines are stored in different C subprograms. These can be found in the header file "multi.h" which also contains many definitions of variables and constants.

In order to make use of the MC68881 floating-point coprocessor, the "makefile" should include the option "-m68881" if it uses the GCC compiler or "-f68881" if it uses the CC compiler. In the linking stage of the compilation in the "makefile", the archived library "robo:/usr/ee/jill/lib/libpuma.a" that contains C-VAL II functions and DDCMP routines should also be included.

4. The Format of *input* and *output* Data File

The input data file contains information of the number of obstacles (*num*), the resolution (*reso*) of the path (i.e. how many points should be between every 50mm length of the path), start (X_s,Y_s) and goal (X_g,Y_g) locations, obstacle indices (*ind*), obstacle locations (X_{obs},Y_{obs}), obstacle heights (H_{obs}) and sizes (WID_X_{obs},WID_Y_{obs}). The format of the input data file is defined as follows:

X_s	Y_s				
X_g	Y_g				
reso	*num*				
ind	X_{obs}	Y_{obs}	H_{obs}	WID_X_{obs}	WID_Y_{obs}

Table A4.1. is an example of the input data file which results a path shown in Figure 5.5.(a).

Table A4.1.

Example of The Input Data File

400	500				
-300	700				
1	9				
1	310	520	0	40	40
2	-220	615	0	40	40
3	-40	550	0	40	40
4	-95	635	0	50	50
5	85	485	0	25	45
6	245	635	0	25	45
7	115	720	0	30	40
8	35	655	0	25	70
9	155	550	0	25	85

The generated path is stored in an output data file "path.rec". It uses the transformation format defined in (5-1), that is [X, Y, Z, O, A, T] for each point of the path. Therefore, the *i*th point is at the *i*th line of file "path.rec" in the form:

X_i Y_i Z_i O_i A_i T_i

In our applications, we set $Z_i = -400$, $O_i = 0$, $A_i = 90$ and $T_i = 0$. Thus only values of X_i and Y_i are not constant.

5. A Sample Auto-Start Program

Before C-VAL II commands can be sent to the PUMA controller via the supervisor port, the communication should be at a ready state by running a small auto-start program in the VAL II memory following the procedure below:

(a) Before turning on power to the PUMA controller, press and hold down AUTO START pushbutton on the front panel.

(b) Turn on power. Continue holding down AUTO START pushbutton until AUTO START indicator light on and floppy drive goes on and off.

(c) Turn on ARM power and set the controller to COMP mode from the teach pendant.

Once the above procedure is done the PUMA will only accept commands from the supervisor computer. Figure 5.3. shows the details of our PUMA 560 and supervisor computer (SUN 3/160) system.

Table A5.1 is a sample auto-start program called "auto" which we use for our applications.

Table A5.1.

A Sample Auto-Start Program Stored in The VAL II Memory.

```
.PROGRAM auto
MC CA              ;calibrate all joint-position sensors
MC DO RIGHTY       ;keep righty arm configuration
MC DO ABOVE        ;keep elbow above arm configuration
MC DO FLIP         ;keep wrist flipped
MC DO READY        ;move hand to home position as in (5-4)
MC SPEED 30        ;set PUMA to 30% of its maximum speed
MC ENABLE NETWORK,SUPERVISOR,DISK.NET,REMOTE.PIN
.END
```