

A Study of Mobile Robot Motion Planning



**DUBLIN CITY
UNIVERSITY**

Ollscoil Chathair Bhaile Átha Cliath

By

Bang Wang B. Sc., M. Sc.

A Dissertation Presented in Fulfilment
of the Requirements for the Ph. D. Degree

Supervisors

Prof. Michael Ryan

Mr. Charlie Daly

School of Computer Applications

December 1994

Declaration

I hereby certify that this material, which I submit for assessment on the programme of study leading to the award of Ph. D. degree in Computer Applications, is entirely my own work and has not been taken from the work of others save and to the extent that such work is cited and acknowledged within the text of my work.

Signed: *Bang Wang*
Bang Wang

Date: *1. Dec. 1994*

Acknowledgements

I would like to express my heartfelt gratitude to my supervisors: Prof. Michael Ryan and Mr. Charlie Daly whose help, supervision and guidance were invaluable during my period of study.

I would also like to express my gratitude to Dr. Michael Scott for his proof reading of this thesis and for his invaluable comments and suggestions.

I would also like to thank Dr. Alan Smeaton, Dr. Heather Ruskin, Dr. James Power for their help whilst I was studying in the School of Computer Applications. Sincere thanks are also expressed to the technicians of the School, Mr. Jim Doyle and Mr. Eamonn McGonigle, for all their invaluable help they gave me.

I would also like to thank all my fellow postgraduate students at the School, but I would like to especially mention Gary Leeson, Barry O'Connell, Micheal Padden, Colmán McSweeney, Donal Murray, who helped me understand Ireland, improve my English, and provided much needed support over the last couple of years.

I would also like to thank my good friend, Dr. Jun Yan, for his guidance to this research. Without his suggestions and ideas, I would not have been able to complete this thesis.

Finally, I would like to extend my gratitude to the School of Computer Applications for its financial and other support given to me, and which has given me the opportunity to learn and experience Western (especially Irish) culture.

A Study of Mobile Robot Motion Planning

Bang Wang B. Sc., M. Sc.

Abstracts

This thesis studies motion planning for mobile robots in various environments. The basic tools for the research are the configuration space and the visibility graph. A new approach is developed which generates a smoothed minimum time path. The difference between this and the Minimum Time Path at Visibility Node (MTPVN) is that there is more clearance between the robot and the obstacles, and so it is safer.

The accessibility graph plays an important role in motion planning for a massless mobile robot in dynamic environments. It can generate a minimum time motion in $O(n^2 \bullet \log(n))$ computation time, where n is the number of vertices of all the polygonal obstacles. If the robot is not considered to be massless (that is, it requires time to accelerate), the space time approach becomes a 3D problem which requires exponential time and memory. A new approach is presented here based on the improved accessibility polygon and improved accessibility graph, which generates a minimum time motion for a mobile robot with mass in $O((n+k)^2 \bullet \log(n+k))$ time, where n is the number of vertices of the obstacles and k is the number of obstacles. Since k is much less than n , so the computation time for this approach is almost the same as the accessibility graph approach.

The accessibility graph approach is extended to solve motion planning for robots in three dimensional environments. The three dimensional accessibility graph is constructed based on the concept of the accessibility polyhedron. Based on the properties of minimum time motion, an approach is proposed to search the three dimensional accessibility graph to generate the minimum time motion.

Motion planning in binary image representation environment is also studied. Fuzzy logic based digital image processing has been studied. The concept of Fuzzy Principal Index Of Area Coverage (PIOAC) is proposed to recognise and match objects in consecutive images. Experiments show that PIOAC is useful in recognising objects. The visibility graph of a binary image representation environment is very inefficient, so the approach usually used to plan the motion for such an environment is the quadtree approach. In this research, polygonizing an obstacle is proposed. The approaches developed for various environments can be used to solve the motion planning problem without any modification.

A simulation system is designed to simulate the approaches.

Table of Contents

1. INTRODUCTION.....	1
1.1 The Basic Problem of Motion Planning	2
1.2 Motion Planning Environments	4
1.3 Classification of Motion Planning	5
1.4 Scope of the Thesis.....	7
2. LITERATURE REVIEW	9
2.1 Introduction	9
2.2 Geometrical Planning.....	10
2.2.1 Configuration Space.....	11
2.2.2 Visibility Graph	12
2.2.3 Voronoi Diagram Approach.....	16
2.2.4 Freeway Approach	17
2.2.5 Potential Field Function Approach	19
2.2.6 Exact Cell Decomposition Approach.....	23
2.2.7 Approximate Cell Decomposition Approach	25
2.3 Dynamic Planning	27
2.4 Motion Planning in Dynamic Environments	28
2.4.1 Hardness Results	28
2.4.2 Space Time Approach.....	29
2.4.3 Divide and Conquer Strategy	38
2.4.4 Accessibility graph.....	39
2.4.5 Collision Avoidance with Moving Obstacles	41
2.4.6 Collision Detection Among Moving Objects.....	42
2.5 Fuzzy Logic Methods.....	43
2.5.1 Fuzzy Logic Methods for Mobile Robots Navigation	43

2.5.2 Fuzzy Logic Methods for Digital Image Processing	45
2.6 Neural Networks for Mobile Robots Navigation	45
2.7 Summary	47
3. SHORTEST PATH PLANNING	50
3.1 Introduction	50
3.2 Configuration Space.....	51
3.2.1 Definition of Configuration Space.....	51
3.2.2 Obstacles in Configuration Space	53
3.3 Visibility Graph	58
3.3.1 Definition of the Visibility Graph.....	58
3.3.2 Graph Search Algorithms	60
3.3.3 Acceleration of Graph Search.....	61
3.3.4 Smoothing the Path.....	66
4. MINIMUM TIME PATH PLANNING	73
4.1 Introduction	73
4.2 Dynamic Planning	74
4.2.1 Mathematical Model of Dynamic Planning	75
4.2.2 Minimum Time Path at Visibility Nodes (MTPVN)	79
4.3 Smoothed Minimum Time Path	95
4.4 Summary and Discussion	103
5. MOTION PLANNING IN DYNAMIC ENVIRONMENTS.....	105
5.1 Introduction	105
5.2 Accessibility Graph.....	107
5.2.1 Definition	107
5.2.2 Minimum Time Motion Planning Algorithm	112
5.3 Accessibility Polygon and Its Applications in Motion Planning.....	115
5.4 Improved Accessibility Graph	119
5.4.1 Improved Accessibility Polygon	119
5.4.2 Improved Accessibility Graph	125
5.4.3 Motion Planning Algorithm for Robots with Mass	127
5.5 Applications	131
5.5.1 Applications of Theory of Sectors to Accelerate the Search	131
5.5.2 Circular Obstacles and Convex Curved Obstacles.....	132
5.5.3 Growing and Shrinking Obstacles	136
5.5.4 Piecewise Linear Motion of the Obstacles.....	139

5.5.5 Motion Planning in Environments with Accelerating Obstacles	141
5.5.6 Motion Planning in Environments with Rotating Obstacles.....	143
5.5.7 Motion Planning in Environments with Transient Obstacles	145
5.6 Summary and Discussion	148
6. MOTION PLANNING IN THREE DIMENSIONAL ENVIRONMENTS.....	152
6.1 Introduction	152
6.2 Obstacles in a Three Dimensional Environment.....	153
6.2.1 Definition of Polyhedra	153
6.2.2 Obstacles in Configuration Space	155
6.3 Shortest Paths in a Three Dimensional Static Environment.....	157
6.3.1 Properties of a Shortest Path.....	157
6.3.2 Three Dimensional Visibility Graph.....	159
6.4 Motion Planning in a Three Dimensional Dynamic Environment.....	160
6.4.1 Properties of a Minimum Time Motion	161
6.4.2 Three Dimensional Accessibility Graph	166
6.5 Summary and Discussion	175
7. MOTION PLANNING IN BINARY IMAGE REPRESENTATION ENVIRONMENTS	176
7.1 Introduction	176
7.2 Fuzzy Logic Methods for Digital Image Processing.....	178
7.2.1 Introduction	178
7.2.2 Fuzzy Measures	179
7.2.3 Fuzzy Operators for Digital Images.....	186
7.2.4 Fuzzy Logic based Image Motion Detection and Analysis	192
7.3 Motion Planning in Binary Image Representation Environments	201
7.3.1 Quadtree Approach	201
7.3.2 Approximating an Obstacle with Its Principal Rectangle	203
7.3.3 Polygonizing an Obstacle	205
7.4 Summary and Discussion	209
8. SIMULATION PROGRAMMING.....	211
8.1 Introduction	211
8.2 Simulation Programming for Static Environments.....	212
8.2.1 A Special Window System for DOS Applications	212
8.2.2 Simulation System Design	216

8.3 Simulation Programming for Dynamic Environments.....	218
8.3.1 The Moving Object Class “CSprite”	218
8.3.2 Obstacle Object Class “Polygon”	220
8.3.3 Creating the Bitmap of a Polygonal Obstacle.....	222
8.3.4 Simulation System Design	224
8.4 Digital Image Processing Programming.....	227
9. CONCLUSIONS.....	228
9.1 Conclusions	228
9.2 Recommendations for Further Research	231
REFERENCES.....	233
APPENDIX A. SOME HEADER FILES	258
A.1 Header File of class Rect and Rso.....	258
A.2 Header File for Screen Style Definitions.....	259
A.3 Header File of Class TxBuff.....	260
A.4 Header File of Class Trso	261
A.5 Header File of Class Grso	261
A.6 Header File of Class Fso	262
A.7 Header File of Class Tfso	263
A.8 Header File of Class Gfso.....	263
A.9 Header File of Class Iso and Class IsoMgr.....	264
A.10 Header File of Text Window Object Class Wso	266
A.11 Header File of Graphic Window Object Class Wso	266
A.12 Header File of the Menu System	267
A.13 Header File of Text Button Class and Icon Button Class	268
APPENDIX B. SOURCE CODES OF CLASS CSPRITE	270
B.1 Original Source Codes	270
B.2 Source Codes after Revision	275
APPENDIX C. RESULTS OF FUZZY FEATURES OF SOME OBJECTS.....	283

Chapter

1. Introduction

One of the ultimate goals in Robotics is to create autonomous robots. Such robots will accept high-level descriptions of tasks and will execute them without further human intervention. The input descriptions will specify what the users want the robot to do rather than how to do it. The robots will be any kind of versatile mechanical device equipped with actuators and sensors under the control of a computing system.

Making progress towards autonomous robots is of major practical interest in a wide variety of application domains such as manufacturing, construction, waste management, space exploration, undersea work, assistance for disabled and medical surgery. It is also of great technical interest for computer science. It raises challenging and rich computational issues from which new concepts of broad usefulness are likely to emerge.

Developing the technologies necessary for autonomous robots is a formidable undertaking which spans areas such as automated reasoning, perception and control. It raises many important problems. One of these is motion planning which this thesis covers.

The problem of motion planning can be loosely stated as follows: How can a robot decide what motions to perform in order to achieve goal arrangements of physical objects ? This capability is eminently necessary since, by definition, a robot accomplishes tasks by moving in the real world. The minimum one would expect from an autonomous robot is to plan its own motions.

Planning is a part of everybody's activities in daily life. We need to plan our actions from the moment we get up in the morning. We plan what to do for the day, what to eat for lunch, whom to speak to, how to get to the party in time in the evening, etc. At first glance, motion planning looks relatively simple, since humans deal with the problem with no apparent difficulties in their everyday lives. In fact, the elementary operative intelligence which people use unconsciously to interact with their environment, e.g. preparing and serving coffee, assembling a device and moving in a building, turns out to be extremely difficult to duplicate using a computer controlled robot.

1.1 The Basic Problem of Motion Planning

The motion planning problem is that of finding a motion for an autonomous mobile robot which must move from a given starting position to a given destination position in an environment that contains a pre-defined set of obstacles so that the robot does not collide with any of the obstacles. An obstacle can be a solid object that the robot can not physically pass through, a slippery area where the robot has to go around, a dangerous area on the ground on which the robot can not function well for some reason or for safety, an area that needs to be avoided because other robots are at work, etc. Solving this problem is crucial for many tasks such as navigation and factory automation.

The goal of defining a basic motion planning problem is to isolate some central issues and investigate them in depth before considering additional difficulties. In the basic

problem, it is assumed that the robot is a moving object in the workspace and the dynamic properties of the robot will be ignored. The motion is constrained to non-contact motion so that the issues related to the mechanical interaction between two physical objects in contact can be ignored. These assumptions essentially transform the “physical” motion planning problem into a purely geometrical path planning problem. The robot is further assumed as a rigid object, that is, an object whose points are fixed with respect to each other. So the motions of the robot are only constrained by the obstacles.

The basic motion planning problem resulting from the above simplifications can be defined as follows:

Let $\mathbf{A}$ be a single rigid object, that is, the robot, moving in Euclidean space $\mathbf{W}$, called workspace, represented as $\mathbf{R}^N$, with $N=2$ or 3 .

Let $\mathbf{B}_1, \mathbf{B}_2, \dots, \mathbf{B}_q$ be fixed rigid objects distributed in $\mathbf{W}$. They are called obstacles.

Assume that both the geometry of $\mathbf{A}, \mathbf{B}_1, \mathbf{B}_2, \dots, \mathbf{B}_q$ and the locations of $\mathbf{B}_1, \mathbf{B}_2, \dots, \mathbf{B}_q$ in $\mathbf{W}$ are accurately known. Assume further that no kinematic constraints limit the motions of $\mathbf{A}$.

The problem is: Given an initial position and orientation and a goal position and orientation of $\mathbf{A}$ in $\mathbf{W}$, generate a path τ specifying a continuous sequence of positions and orientations of $\mathbf{A}$ avoiding contact with $\mathbf{B}_1, \mathbf{B}_2, \dots, \mathbf{B}_q$, starting at the initial position and orientation, and terminating at the goal position and orientation. Report failure if no such path exists.

It is obvious that the basic problem is somewhat oversimplified. But solving it is of practical interest. Furthermore, it can be easily extended to other practical motion

planning problems and a lot of practical motion planning problems can be realistically expressed as instances of the basic motion planning problem by some simple assumptions.

1.2 Motion Planning Environments

A motion planning environment is the working space where the robot will perform its tasks. In the basic motion planning problem definition, the workspace is the Euclidean space where the geometry and locations of all the obstacles are accurately known and all the obstacles are stationary. Practically, this is not always the case. Obstacles in the environment may not always be static. Obstacles which are stationary in the environment, that is, fixed permanently at their initial positions are called static obstacles, while obstacles which can move, change their shapes and size, and even disappear and reappear over time are called dynamic obstacles. An environment in which there are only static obstacles is called static environment while an environment in which there is any dynamic obstacle is called a dynamic environment.

Another issue which is relevant to obstacles in the environment is whether or not the obstacles are known ahead of planning. When all the information regarding the obstacles, that is, sizes, positions and motions etc., of the obstacles is completely known a priori, the obstacles are said to be completely known, or the obstacles are in a known environment (this is the case in the basic motion planning problem). But, in practical applications, the obstacles in the environment are detected by the sensors on the robot. Since the working range and resolution of the sensors on the robot are limited, the geometry and locations of the obstacles in the environment may not have been known accurately by the robot. In this case, complete information about the obstacles is not available at the time of planning, and the environment is said to be partially known.

Motion planning environments are characterised as either static or dynamic, and completely known or partially known. In terms of this classification, the environment can be categorised into four groups as shown in Table 1.1, that is, (1) completely known static environment, (2) partially known static environment, (3) completely known dynamic environment, (4) partially known dynamic environment. In the basic motion planning problem definition, the environment is the completely known static environment.

	Static Obstacles	Dynamic Obstacles
Known	Completely Known Static Environment	Completely Known Dynamic Environment
Partially Known	Partially Known Static Environment	Partially Known Dynamic Environment

Table 1.1: Classification of Environments

1.3 Classification of Motion Planning

In the basic motion planning problem definition, it is assumed that only geometrical factors will be taken into consideration. If the dynamics of the robot are not considered, that is, only geometry of the environment is concerned, the motion planning is called geometrical planning. The geometrical planning only determines the shortest path for the robot from the starting position to the destination position. A number of problems linked with the inertia and velocity dependent factors makes geometrical planning insufficient for mobile robot motion planning. So motion planning in which the inertia and velocity dependent factors are taken into consideration is needed and is called dynamic navigation. Dynamic navigation can determine the minimum time path for the mobile robot from the starting position to the destination position.

If the classification of the environment is taken into account, the motion planning of mobile robots can be classified as Table 1.2, that is, (1) geometrical planning in completely known static environment, (2) geometrical planning in partially known static environment, (3) geometrical planning in completely known dynamic environment, (4) geometrical planning in partially known dynamic environment, (5) dynamic planning in completely known static environment, (6) dynamic planning in partially known static environment, (7) dynamic planning in completely known dynamic environment, (8) dynamic planning in partially known dynamic environment.

	Geometrical Planning	Dynamic Planning
Completely Known Static Environment	Geometrical Planning in Completely Known Static Environment	Dynamic Planning in Completely Known Static Environment
Partially Known Static Environment	Geometrical Planning in Partially Known Static Environment	Dynamic Planning in Partially Known Static Environment
Completely Known Dynamic Environment	Geometrical Planning in Completely Known Dynamic Environment	Dynamic Planning in Completely Known Dynamic Environment
Partially Known Dynamic Environment	Geometrical Planning in Partially Known Dynamic Environment	Dynamic Planning in Partially Known Dynamic Environment

Table 1.2: Classification of Motion Planning

Case (1) in fact is the basic motion planning problem. Now it is obvious that only solving the basic motion planning problem is not enough for practical applications. The basic motion planning problem must be extended to the other more practical problems. This thesis will focus on the basic problem and its extension problems listed in table 1.2.

Much of the prior research on motion planning is concerned with case (1) and (2), that is, geometrical planning in completely known and partially known static environment. There is also research on case (5) and (6), that is, dynamic planning in completely known and partially known static environment. Recently, cases (7) and (8) are attracting many researchers. There is little research on cases (3) and (4), as shortest path planning is meaningless in a dynamic environment.

1.4 Scope of the Thesis

The rest of this thesis is organised as follows. Chapter 2 contains detailed review of the literature. The approaches to various problems of motion planning are reviewed.

In chapter 3, the configuration space and shortest path problem are discussed in detail. The visibility graph method which can lead to the solution of shortest path in completely and partially known environments is described. Simulation results of shortest path problem are presented. Configuration space and visibility graph form the bases for the other part of the thesis.

Chapter 4 is about dynamic planning of a mobile robot in completely known and partially known environments. First the visibility graph method to solve the dynamic planning problem is discussed. Then a new method which is more practical is introduced. Simulation results are also presented.

Motion planning in dynamic environment is researched in chapter 5. First the accessibility graph method is introduced and proven to lead to the solution of minimum time path for massless mobile robot. The space time method which solves the motion planning of massive mobile robots is also introduced. Then, based on the concepts of improved accessibility polygon and improved accessibility graph, the accessibility graph method is extended to solve the motion planning problem of

massive mobile robots, that is, the improved accessibility graph method. Some special cases about dynamic environments are also discussed.

In Chapter 6, motion planning in 3 dimensional environments is discussed. First, the properties of a motion in static environments and dynamic environments are researched. The concepts of the accessibility polyhedron and the three dimensional accessibility graph are introduced. A new approach based on three dimensional accessibility graph is proposed to generate the minimum time motion for 3 dimensional dynamic environments.

Motion planning in binary image representation environments is discussed in chapter 7. Fuzzy logic methods for digital image processing are first discussed. The concept of fuzzy principal index of area coverage is introduced and its applications in digital image motion detection and analysis is discussed. A new approach based on polygonizing an obstacle for the motion planning of a mobile robot is introduced when the planning environment is given as a two dimensional binary image. Two methods of polygonizing an obstacle are implemented. One is approximating an obstacle by its principal rectangle. Another is approximating an obstacle by an eight vertices polygon.

In chapter 8, simulation programming is discussed. A simulation system using C++ in a PC is introduced in full details.

Chapter 9 is the conclusion of the thesis. Further research on the subject is also recommended.

Chapter

2. Literature Review

2.1 Introduction

There has been a wealth of research on mobile robot motion planning in static and dynamic environments. This chapter will review the approaches which are used to deal with various motion planning problems. First, approaches to geometrical planning in static environments are briefly reviewed, especially, the concepts of the visibility graph and space decomposition which play fundamental roles in path planning in two dimensional environments. The visibility graph is important since it is used to generate the shortest path between the starting and destination points in a two dimensional environment cluttered with polygonal obstacles. The idea of space decomposition has led to a number of methods to generate reasonably good paths between the starting and destination points in a static environment. The artificial potential field function approach is fast but a local minimum can cause it to fail.

The concept of dynamic planning in a two dimensional static environment is then introduced. Approaches to deal with dynamic planning in a static environment are

reviewed. The concept of the visibility graph also plays an important role in dynamic planning .

In this chapter, the previous approaches to solve the problems of dynamic planning in dynamic environments are reviewed. Some research results have shown from a computational theoretical point of view that planning motion in dynamic environments is harder than planning in static environments. It has been proven that even for a simple case in a two dimensional dynamic environment the problem falls into the category of NP-hard. This means that the problem of motion planning in dynamic environments belongs to the class of problems to which no polynomial time algorithm is known to date. However, there is variety of approaches at solving the problem. Some approaches make use of heuristics or the concept of improved visibility, while some others try to solve the subclass of the problem in polynomial time.

Finally, fuzzy logic and neural network approaches are reviewed.

2.2 Geometrical Planning

In geometrical planning only the geometrical factors of the robot and the environment are taken into consideration. The inertia and velocity linked factors are not considered. Motion planning problems in this domain are usually solved in the following two steps. First, a graph is defined to represent a geometric structure of the environment. Next graph search is performed to find a connected component between the node containing the starting position and the node containing the destination position. The meaning of geometric structure embedded in the graph varies from one approach to another. For example, a node of the graph in a particular approach (e.g. space decomposition) can represent a convex safe region (that is an area that is not occupied by any of the obstacles), while an edge of the graph can represent an adjacency relationship between safe regions. A connected component in

the graph in such formulations represents a series of safe regions that are adjacent to each other, from which a collision-free path is obtained.

Note that the graph does not have to be constructed in the first step. It is often a better strategy not to build the entire graph, especially when the number of nodes in the graph is potentially very large. In such a case, the graph is just defined on the environment and the second step incrementally constructs the graph starting from the start node, while searching for a path. Construction of the graph terminates as soon as the node containing the destination position is found. This saves having to build the entire graph.

When the number of nodes in the graph is relatively small, Dijkstra's shortest path algorithm is often used to find a path after the graph has been constructed. When the number of nodes in the graph is large, heuristic search algorithms such as A^* are often used to accelerate the search process.

2.2.1 Configuration Space

Configuration space is a very important concept in motion planning of a mobile robot [Udapa, S. 1977] [Lozano-Preze, T. 1979] [Ilari, J. 1990] [Bajai, C. 1990]. The configuration space is a transformation from the physical space in which the robot is of finite size into another space in which the robot is treated as a point. This will simplify the problem. Formally, configuration space is a concept in mechanics and is described as follows. The position and orientation of an object A in two dimensional space can be specified by a 3 tuple (x, y, θ) in three dimensional space, called the configuration space of the object, and is denoted by $Cspace_A$. Here (x, y) represents the position of a reference point of A in the plane, and θ represents the angle between the reference axis of A and x axis of the reference coordinate. If the orientation of A is fixed its configuration space is two dimensional, since the pair (x, y) is sufficient to specify the position of A .

In $Cspace_A$ some points correspond to placements of A in which A overlaps other objects in physical space. Such points are called illegal. While other points that do not overlap with other objects are called legal. More specifically, the sets of points in $Cspace_A$ that correspond to the placements of A where A overlaps with object B in physical space is called a Cspace obstacle for A due to B and denoted by $CO_A(B)$, that is, an obstacle in physical space is transformed into an obstacle in configuration space. The problem of motion planning for A in physical space is transformed into the problem of finding a path for the point in the configuration space such that every point on the path is legal. As a result of the transformation, obstacles which do not overlap with each other in physical space may overlap in configuration space. The concept of configuration space will be discussed in more details in the next chapter.

2.2.2 Visibility Graph

The visibility graph [Nilsson, N. 1969] [Lozano-Perez, T. 1979] [Ghosh, S. 1987] is one of the earliest path planning methods. It can be used to plan a motion for a mobile robot in completely known and partially known static environments [Meystel, A. 1990] [Latombe, J. 1990]. It mainly applies to two dimensional configuration space with polygonal obstacles. The visibility graph is the non-directed graph whose nodes are the starting point and destination point and all the vertices of obstacles in the environment. The links of the visibility graph are all the straight line segments connecting two nodes that do not intersect the interior of any obstacle in the environment. Fig. 2.1 is an example of a visibility graph. The three shaded polygons represent obstacles in configuration space. Points S and G are the starting point and destination point respectively. The shortest path from S to G is a finite sequence of edges of the visibility graph. A sequence of thick line segments in Fig. 2.1 represents a shortest path from S to G.

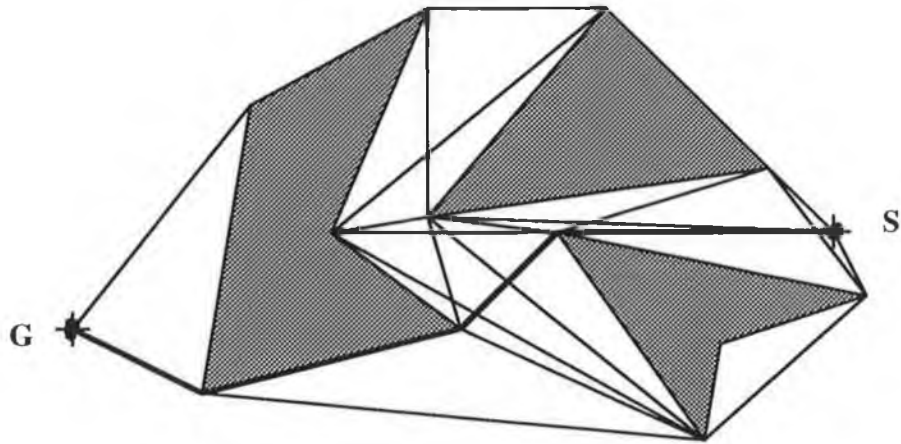


Fig. 2.1: Visibility Graph

In shortest path computations making use of visibility graph, it is necessary to determine which obstacle vertices are visible from a given point. It takes $O(n \bullet \log(n))$ time to determine visible vertices from a given point, where n is the total number of vertices in all the polygons [Fujimura, J. 1992]. By applying the process of finding the visible vertices to every vertex of polygons in the environment, it takes $O(n^2 \bullet \log(n))$ time to construct the whole visibility graph.

For partially known environments, the robot does not know exactly the positions and numbers of the polygons in the environment because of the working range of sensors on the robot. The Wandering Standpoint Algorithm (WSA) [Meystel, A. 1990] is used in this case. The visibility graph of the known vertices is constructed. The next standpoint of the process of wandering is determined using the A^* -like cost function $g=f+h$. When the robot arrives the next standpoint, the visibility graph from that standpoint is constructed again based on new information from the robot sensors.

In [Meystel, A. 1990], sector theory is used to speed up the construction of the visibility graph. By using sector theory, at most two successors on every obstacle are needed to add to the visibility graph for finding the shortest path. This will greatly reduce the size of the visibility graph. In [Liu, Y. 1990] [Liu, Y. 1991] and [Liu, Y. 1992], the concepts of local shortest path and tangent graph (or reachability graph)

are used to speed up the search. In fact, construction of tangent graph is almost the same as construction of the visibility graph using sector theory.

Some other methods which speed up the search are reported. For an environment containing m stationary polygonal obstacles with a total of n vertices, Reif and Storer [Reif, J. 1985] have shown an algorithm that runs in $O(n \cdot m + n \cdot \log(n))$ time. Ghosh and Mount [Ghosh, S. 1987] have demonstrated an algorithm to construct the visibility graph in $O(E + n^2)$ time, where E is the numbers of edges in the visibility graph. Kapoor and Maheshwari [Kapoor, S. 1988] have observed that a shortest path can be computed in $O(E_s + n^2)$ time, where E_s is the numbers of edges in the visibility graph that are tangents at both endpoints. Here the quantities E and E_s are bounded by n^2 and m^2 , respectively. Mitchell [Mitchell, J. 1990a] proposes yet another method that runs in $O(k \cdot n \cdot \log^2(n))$ time, where k is a parameter bounded by n . This method runs asymptotically faster than other methods when k is much smaller than n . Recently, Mitchell [Mitchell, J. 1990b] has obtained an optimal algorithm for finding a shortest path in two dimensional environments in $O(n \cdot \log(n))$ time.

In [Latombe, J. 1990] [Liu, Y. 1991] and [Liu, Y. 1992], the concepts of visibility graph and tangent graph are extended to motion planning in environments with generalised polygons, that is, the generalised visibility graph. In [Latombe, J. 1990], the generalised polygon is defined as a region which is enclosed by line or arc in two dimensional space. In [Liu, Y. 1991] and [Liu, Y. 1992], the generalised polygon is a region which is enclosed by any convex curve and line, though the words “generalised polygon” do not appear there.

For some special cases, faster computation of the shortest path is possible. The case that the obstacles are line segments that are parallel to each other has been solved in $O(n \cdot \log(n))$ time, which is optimal [Lee, B. 1984] [Fujimura, K. 1992]. For three dimensional environments which contain polyhedral obstacles, a shortest path is known to be a polygonal path that bends at some internal points on obstacle edges as well as at vertices of the obstacles. The problem of motion planning in three dimensional environments has proven to be NP-hard. When the visibility graph

method is applied in a polyhedral configuration space, an approximate shortest path is obtained by adding fictitious vertices in the edges of the obstacles [Latombe, J. 1990] [Fujimura, K. 1992].

A related motion planning problem called the weighted region motion planning has also been studied [Mitchell, J. 1990c] [Mobasser, B. 1990] [Rowe, N. 1990] [Fujimura, K. 1992]. The environment plane is subdivided into regions of different weights which represent costs per unit distance in the corresponding regions. The problem is to find a least cost path between a given starting position and destination. The conventional shortest path problem is a special case of the weighted region problem in which a point on the plane is assigned a weight of ∞ if it is inside an obstacle, and 1 otherwise.

Another related problem in motion planning is the safest problem [Kanayama, Y. 1988]. The cost function based on the clearance from obstacles is considered. A cost per unit along the path is defined as $w^{-1/k}$, where w is the clearance from the obstacles in the environment and k is a positive integer called the safety index. The total cost of the path, which is obtained by integrating the above cost over the entire path,

$$Cost = \int w^{-1/k} ds \quad (2.1)$$

is to be minimised. This minimisation problem is solved by using the calculus of variations. In the above formulation, a small k results in a path which is safer but is longer. Kanayama also considers the use of cost functions of the form $w^{1/k}$ along the path. In contrast with the path obtained in the previous case, the path obtained by using this cost function prefers to pass close to the obstacles in the environment. In [Suh, S. 1988], an alternative approach by dynamic programming to path planning using cost functions is reported.

2.2.3 Voronoi Diagram Approach

Voronoi diagrams were introduced by Voronoi more than 80 years ago. They have been studied since then by many researchers, and have found applications in many areas such as crystallography, physics, molecular chemistry, geology, pattern analysis, image processing, statics and so on [Leven, D. 1987]. Many algorithms have been proposed for the construction of Voronoi diagrams. In [Leven, D. 1987], a more efficient method has been proposed for construction of Voronoi diagrams and the diagrams are used to check the intersection between convex polygons.

The Voronoi diagram approach for a mobile robot motion planning is also known as the retraction approach [Latombe, J. 1990]. It consists of a continuous mapping (also called retraction) of the robot's free space (where the robot does not intersect with any obstacle) onto a one dimensional network of curves in the free space. It can be stated as follows:

1. Compute the Voronoi diagram.
2. Compute the points $\rho(S)$ and $\rho(D)$ and identify the arcs which contain these two points in the Voronoi diagram, where ρ represents the retraction (mapping), S and D are the starting point and destination point respectively.
3. Search in the Voronoi diagram for a sequence of arcs $A_1, A_2, \dots, A_p$ such that $\rho(S) \in A_1$, $\rho(D) \in A_p$, and, for all $i \in [1, p-1]$, A_i and A_{i+1} share an endpoint.
4. If the search terminates successfully return $\rho(S)$, $\rho(D)$, and the sequence of arcs connecting them; otherwise return failure.

The overall time complexity of the above motion planning approach by Voronoi diagram is $O(n \log n)$. In addition it is also applicable when the obstacles in the environment are generalised polygons [Latombe, J. 1990].

Other retraction approaches like Voronoi diagram have been proposed to solve more involved instances of motion planning problems. C. O'Dunlaing, M. Sharir and C. K. Yap [O'Dunlaing, C. 1983] [O'Dunlaing, C. 1984a] [O'Dunlaing, C. 1984b] proposed a retraction method for planning the motion of a segment ("ladder") among polygonal obstacles. In the method, three dimensional free space is first retracted onto a two dimensional variant of the Voronoi diagram. In the second step, the variant Voronoi diagram itself is retracted onto a network of one dimensional curves in the free space. The whole process takes $O(n^2 \cdot \log(n) \cdot \log(\log(n)))$ time. J. F. Canny and B. R. Donald proposed a "simplified Voronoi diagram" which is easier to extend to higher dimensional space than the classical generalised Voronoi diagram [Canny, J. 1988] [Latombe, J. 1990].

2.2.4 Freeway Approach

The freeway approach specifically applies to planning the motion of a polygonal object moving both in translation and rotation in a two dimensional static environment. The intuition behind it is similar to the intuition behind retracting free space onto its Voronoi diagram, that is, keeping the robot as far as possible from the obstacles.

The freeway approach consists of extracting geometric figures called freeways from the environment, connecting them into a graph called the freeway net, and searching the graph. A freeway is a straight linear generalised cylinder. Fig. 2.2 illustrates the geometry of a freeway. Fig. 2.3 shows four freeways in a bounded environment, where the shaded polygons are obstacles.

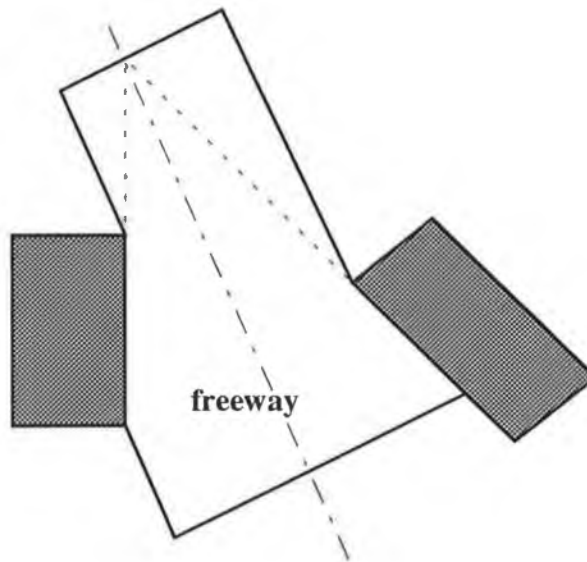


Fig. 2.2: A freeway in two polygonal obstacles

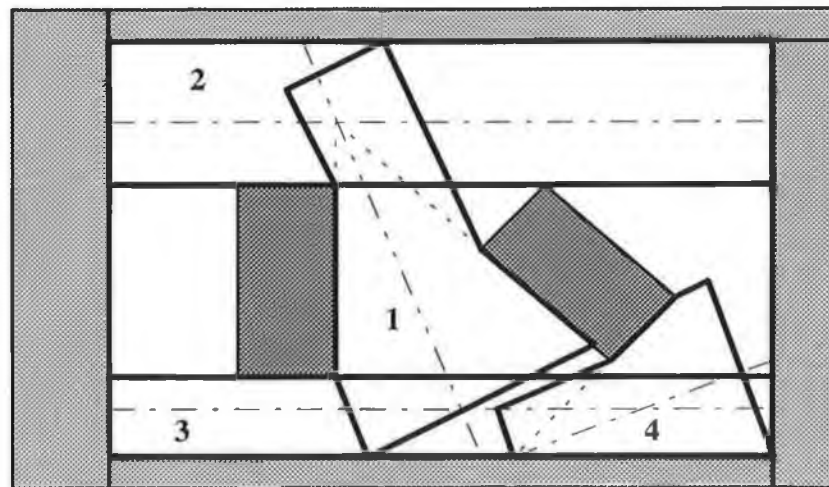


Fig. 2.3: Multiple Overlapping Freeways in a Bounded Environment

Every freeway in the environment is a node of the graph. A special method is used to construct the graph of freeways [Latombe, J. 1990]. The constructed freeway graph is searched for a path between the starting position and the destination position.

Experiments with the freeway approach have shown that it works fast in a “relatively uncluttered” environment [Latombe, J. 1990]. However, the approach is not complete, that is, if a path exists, it may fail to find it.

In [Tang, K. 1992], a similar approach is proposed for the motion planning of mobile robots where orientation is considered. The freeways in this approach are triangles. The free space of the environment is triangulated. A special graph is constructed with the triangles as nodes and is searched for a path between a starting position and the destination position. Simulation results show that it is efficient.

2.2.5 Potential Field Function Approach

Potential field function approach treats the robot represented as a point in configuration space as a particle under the influence of an artificial field U whose local variations are expected to reflect the “structure” of the free space in the environment. The potential field function is typically defined over free space as the sum of an attractive potential pulling the robot towards the goal configuration and a repulsive potential pushing the robot away from obstacles in the environments. Motion planning is performed in an iterative fashion. At each iteration, the artificial force $\mathbf{F} = -\nabla U$ induced by the potential function at the current position is regarded as the most promising direction of motion, and path generation proceeds along this direction by some increment.

Potential field function was originally developed as an on-line collision avoidance approach, applicable when the robot does not have a prior model of the obstacles in the environment, but senses them during motion execution. Emphasis was put on real time efficiency, rather than on guaranteeing the attainment of the goal. In particular, since an on-line potential field approach essentially acts as a fastest descent optimisation procedure, it may get stuck at a position where the potential field function reaches the local minimum other than the destination position. However, the idea underlying potential field can be combined with graph searching techniques. Then, using a prior model of the environment, it can be turned into a systematic motion planning approach.

Even when graph searching techniques are used, local minima remain an important cause of inefficiency for the potential field function approach. Hence, dealing with local minima is the major issue the potential field function approach has to cope with. The issue can be solved in two aspects: (1) in the definition of the potential field function, by attempting to specify a function with no or few local minima, (2) in the designing of the search algorithm, by including appropriate techniques for escaping from local minima.

Most motion planning methods based on potential field function approaches have a strong empirical flavour. They are usually incomplete, that is, they may fail to find a path between the starting position and the destination position even if one exists. But some of them are particularly fast in a wide range of situations. The most important thing is that it makes it possible to design some motion planning methods which are both quite efficient and reasonably reliable, and especially suitable for real time applications. So the potential field function approach is increasingly popular for implementing practical motion planners.

In [Latombe, J. 1990], several methods based on potential field function approach are introduced. The first one is a depth first planning method. It constructs a path as the product of successive path segments starting at the initial position. Each segment is oriented along the negated gradient of the potential field function computed at the position attained by the previous segment. The amplitude of the segment is chosen so that the segment lies in free space. Because this method simply follows the steepest descent of the potential field function until the destination position is reached, it may get stuck at a local minimum position of the potential field other than the destination position. Dealing with the local minima of the depth first planning method is not easy. First the fact that the local minimum has been reached must be recognised by the robot and this is difficult, since motions are discretized, the robot does not stop exactly at the zero force position. Second, the robot must escape from the local minimum and this is not a simple problem. However, the depth first planning method is fast. If the problem of local minimum is solved then leads to the best first planning method.

In best first planning method, the configuration space is first discretized with a fine regular grid. It consists of interactively constructing a tree T whose nodes are nodes in the configuration space grid. The root of T is the starting position of the robot. At every iteration, the algorithm examines the neighbours of the leaf of T that has the smallest potential field function value, retain the neighbours not already in T at which the potential field function value is less than some large threshold M , and installs the retained neighbours in T as successors of the currently considered leaf. The algorithm terminates when the destination position has been reached (success) or when the free subset of the grid accessible from the starting position has been fully explored (failure). Each node in T has a pointer towards its parent. If the search is successful, a path can be generated by tracing the pointers from the destination position to the starting position.

This method follows a discrete approximation of the negated gradient of the potential function until it reaches a local minimum. When this happens, it operates in a way that corresponds to filling the well of this local minimum until a saddle point is reached. So this algorithm is guaranteed a return of a path in the grided configuration space whenever there exists one in that grided space and otherwise reports failure.

For low dimensional configuration space (such as two or three) the best first planning method is efficient, fast and reliable. It may even be made faster by using a pyramid of grids at various resolution. However when the dimensions of the configuration space become larger, filling up the local minimum will take a long time and it will become very inefficient.

Another method of path planning using a potential field function consists of constructing a functional J of a path p and optimising J over all possible paths. For example, if U is a potential field function that is defined over the whole configuration space, with large values in the obstacles region and small values in free space, a possible definition of J is:

$$J(p) = \int U(p(s))ds + \int \left\| \frac{dp}{ds} \right\| ds \quad (2.2)$$

The purpose of the first term in $\mathbf{J}$ is to make the optimisation produce a free path while the second term is optional and is aimed at producing shorter paths. This is similar to the weighted problems in visibility graph. In fact, other objectives can be encoded in this form. The optimisation of $\mathbf{J}$ can be solved by using standard variational calculus or dynamic programming methods [Gilbert, E. 1985] [Hwang, Y. 1988] [Suh, S. 1988].

For larger dimensions of configuration space, the variational path planning method using potential field function will also become very inefficient. Since it essentially consists of minimising a functional $\mathbf{J}$ by following its negated gradient, it may get stuck at a local minimum of $\mathbf{J}$ that does not correspond to a free path. In addition, the optimisation of $\mathbf{J}$ is conducted over a space of much larger dimensions and can be very costly. However, the advantage of variational path planning method is that it allows additional objective criteria to be included in $\mathbf{J}$.

In [Hwang, Y. 1992], a method similar to Voronoi diagram approach by using potential field function is introduced. First a one dimensional network which is similar to the Voronoi diagram is created by searching the valleys of the potential field function over the environment. Then the path is planned by the global and local planners. The definition of potential field function over an environment with more than one polygonal obstacles is different from the definition in [Latombe, J. 1990]. In [Latombe, J. 1990], the value of the potential field function at any point is the sum of the values of the potential field function due to every single polygonal obstacle, while in [Hwang, Y. 1992] the value at any point is given by the maximum value among the values of the potential field function due to every individual polygonal obstacle.

It is possible to make the potential field function approach more efficient by defining the potential field function in a special manner so that no local minima exist. But it is difficult to do that, especially when the shapes of the obstacles in the environment are

complex or the dimensions of the configuration space are relatively big. Combining the techniques of special definition of the potential field function and the best first planning method, a new powerful planning method based on the potential field function approach - randomised planning method has been implemented and experimented with in difficult motion planning problems [Barraquand, J. 1990] [Latombe, J. 1990].

2.2.6 Exact Cell Decomposition Approach

The principle of the exact cell decomposition approach is to first decompose the robot's free space into a collection of non-overlapping regions, called cells, whose union is exactly the free space. Next, the connectivity graph which represents the adjacency relationship among the cells is constructed and searched. If the search is successful, the outcome of the search is a sequence of cells, called a channel, connecting the cell containing the initial position to the cell containing the destination position.

The most simple case in the application of exact cell decomposition is to plan a path in a two dimensional configuration space, that is, the orientation of the robot is not taken into consideration. The free space in environment is decomposed into small cells using a method called trapezoidal decomposition. Two cells which have a border in common are called adjacent. The connectivity graph is constructed based on adjacency and searched. The path generated by the exact cell decomposition is shown in Fig. 2.4.

Schwartz and Sharir proposed an exact cell decomposition method which can plan the motion of a line segment (also called a ladder) among an environment with polygonal obstacles. Because the orientation of the line segment is taken into consideration, the configuration space becomes a three dimensional space. Based on the concept of critical region, the free space is decomposed into special three dimensional cells. The connectivity graph is constructed with these special cells as

nodes. The starting and destination positions are identified into their corresponding cells. The path between the starting position and destination position can be successfully found by searching the graph [Schwartz, J. 1983a].

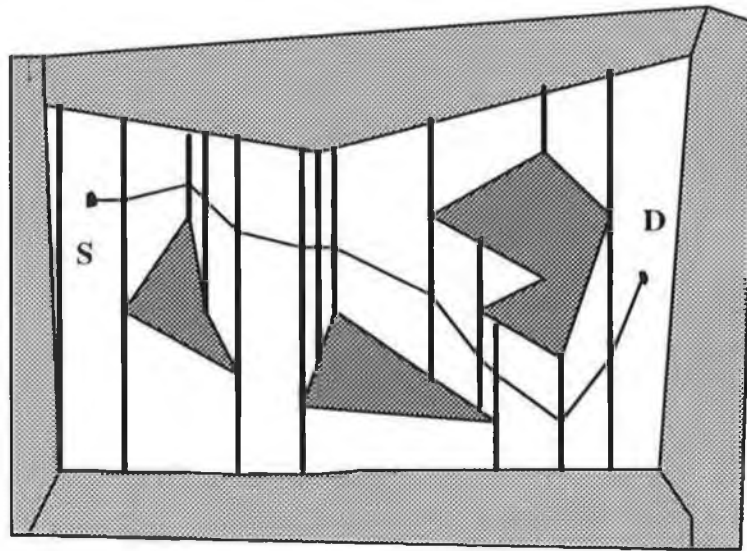


Fig. 2.4: Exact Cell Decomposition and Path

Schwartz and Sharir also proposed another general path planning method based on the exact cell decomposition approach [Schwartz, J. 1983b]. It is based on the Collins decomposition of the free space. It is a general path planning method, that is it can solve many kind of motion planning problems. However, it is very inefficient. It merely serves as a proof of existence of a general path planning method.

Avnaim, Boissonnat and Faverjon proposed a method which can plan the motion of a polygonal object that can both translate and rotate among polygonal obstacles based on a variant of the exact cell decomposition approach [Avnaim F. 1988]. Rather than decomposing free space into three dimensional cells, the method only decomposes the free space's boundary and some well chosen subsets of the free space into two dimensional cells. The method is easy to implement and experiments show it is also efficient.

2.2.7 Approximate Cell Decomposition Approach

The concepts of quadtrees and octrees have led to the approximate cell decomposition approach for motion planning [Kambhampati, S. 1986] [Fryxell, R. 1987] [Noborio, M. 1990] [Zhu, D. 1991]. Quadtrees can also be used to represent the shape of an object in digital image processing [Jahne, B. 1992]. The idea of approximate cell decomposition approach is to keep subdividing the environment into subspaces of equal size recursively until each subspace is either completely occupied by some obstacle, or completely outside of any of the obstacles, or the pre-specified resolution limit is reached. As a result of the dividing, the obstacles and free space in the environment are represented as a collection of blocks of various sizes. A hierarchical data structure called the region quadtree is often used to store these blocks. After the decomposition, a collision free path is obtained as a sequence of blocks which are adjacent to each other. The method is particularly attractive when the environment is given in the form of a two dimensional array whose pixels indicate whether or not corresponding locations in the environment are occupied by obstacles. This form of environment is typically obtained by taking an image of the environment by a camera fixed in the environment or fixed on the robot. And since using cameras as sensors for a mobile robot is very popular, the method is useful and practical. When the environment is given as a two dimensional binary image, the visibility graph approach has the disadvantage that because of digitising, there can be many vertices in the image, which make it very time consuming to construct the visibility graph. Even with the use of sector theory, it still takes time for the visibility graph approach to find the vertices for constructing the visibility graph, since there are so many vertices in the obstacle.

The idea of using the quadtree representation for motion planning is to ignore such details that have little to do with generation of the path. First, the input binary image is converted to the quadtree representation. After that the environment is represented as a collection of blocks of various sizes. In this way, large blocks representing free space are quickly identified, resulting in a fast computation of the path. Then some heuristic search method such as A^* is used to search for a path. Although the path

obtained in this representation is usually not shortest in terms of Euclidean distance, experimental results show that path planning based on the quadtree representation takes less computation than the method using visibility graph when the environment is given as a binary image [Noborio, H. 1990].

A variant of the quadtree representation of the environment is called the PM-quadtree [Samet, H. 1985]. The idea is to subdivide space into subspaces recursively until the contents of each subspace become simple enough (e. g., each subspace contains only one vertex of a polygon of each subspace is intersected by only one edge of a polygon). This representation can store polygonal shapes exactly with less storage than the region quadtree representation. This representation is attractive for storing a large number of polygonal shapes as required in geographical information systems.

The idea of decomposition of a three dimensional environment in a similar way leads to the concept of octree. A three dimensional binary image can be represented by using regular decomposition by means of the x, y, z axes, which results in an octal tree called octree representation [Meagher, D. 1982]. The octree representation of the environment has been used to detect collisions between two three dimensional objects and planning a path among stationary obstacles in three dimensional environments [Herman, M. 1986] [Faverjon, B. 1984] [Wong, E. 1986]. The octree representation is also used to represent the space time of a two dimensional dynamic environment and solve the constrained motion planning of mobile robots [Fujimura, K. 1992].

Another method for geometrical planning is the Silhouette Method [Canny, J. 1988] [Latombe, J. 1990]. This is a general planning method. It involves tools from differential geometry and elimination theory. It is a very theoretical method and very inefficient. It is only useful in establishing the existence of a path.

2.3 Dynamic Planning

Geometrical planning approaches can generate a lengthy optimal path for a mobile robot in a planning environment. But a lengthy optimal path may not be a time optimal path. In some cases, time minimum is more important than length minimum. So the problem of dynamic planning, that is, minimum time planning is proposed [Meystel, A. 1990] [Meystel, A. 1986] [Meystel, A. 1986].

Dynamic planning is very similar to geometrical planning and the visibility graph plays a very important role in solving the dynamic planning problems. In [Meystel, A. 1990] [Meystel, A. 1986] [Meystel, A. 1986], “slalom” situations are analysed, then the topological passageways of the environment are calculated. The values of acceleration and deceleration are constrained in certain ranges, that is, the change of speed in unit time is limited, and the mass of the robot will not influence the planning, instead it is reflected in the acceleration and deceleration. The maximum velocity of the robot is also limited. An A^* like algorithm is used to search the visibility graph for the minimum time path. When the robot reaches a turning point, the velocity of the robot at that turning point is determined by a non-linear equation. The bigger the angle the robot is about to turn, the less the velocity of the robot. When the angle is 180 degrees, the velocity of the robot will be 0. If the angle is 0 degree, the velocity of the robot at the turning point will be the maximum velocity. The time the robot takes to turn is proportional to the angle and inversely proportional to the velocity at that turning point. Because the successors of the point are not known when the robot reaches it, the calculation of the time for the robot to turn will lead to depth first search. This will be inefficient. So the concept of Point Of Invariant is introduced. The velocity curve can be found by using production rules. The minimum time path can be generated by searching the visibility graph.

In [Donald, B. 1989], an approximate algorithm which can solve the minimum time path planning problem more quickly is proposed and implemented. It is possible to bound the approximation by an error term ϵ . The algorithm can run in polynomial time. If the geometrical obstacles, the dynamic bounds and the error term ϵ are

supplied, the algorithm can generate the solution that is ϵ close to optimal and spends only a polynomial (in $(1/\epsilon)$) amount of time computing the answer.

2.4 Motion Planning in Dynamic Environments

Research results indicate that planning motion of a mobile robot in an environment with moving obstacles is intrinsically harder than planning motion in the environment with stationary obstacles. Most of the problems dealing with stationary obstacles in two dimensional environments are solvable in polynomial time, while even some basic problems dealing with moving obstacles in two dimensional environments do not seem to be solvable in polynomial time. Nevertheless, there are number of algorithms that have been proposed to solve motion planning problems in dynamic environments. Many of these algorithms work in limited domains and generate a path in polynomial time.

2.4.1 Hardness Results

Reif and Sharir [Reif, J. 1985] show that the problem of planning motions of a three dimensional rigid robot in an environment containing stationary and moving obstacles is PSPACE-hard when the robot's velocity is bounded, and NP-hard without such a bound.

Canny and Reif [Canny, J. 1987] show the following lower bound on the motion planning problem in dynamic environments: motion planning for a point in the plane with bounded velocity is NP-hard, even when the moving obstacles are convex polygons moving with constant linear velocities without rotation. The problem is shown in PSPACE [Canny, J. 1988]. The hardness of the general problem of avoiding moving obstacles in two dimensional environments has also been researched by Sutner and Maass [Sutner, K. 1988]. A time dependent graph has been used to

formulate the problem. It is proved that the path existence problem for time dependent graphs is PSPACE-complete.

There is a similar but different category of problems that are usually called coordinated motion problems. They involve planning coordinated motions for multiple robots. This problem is also motion planning in dynamic environments in the sense that the approach must consider the motions of several robots at the same time. Results show that such problems in two dimensional environments are PSPACE-hard. But there are some approaches which run in polynomial time for some special cases [Hopcroft, J. 1984] [Yap, C. 1984].

2.4.2 Space Time Approach

Despite the difficulty of motion planning in dynamic environments, various approaches have been proposed to solve the problem. Sutner and Maass [Sutner, K. 1988] have shown that the problem of avoiding moving obstacles in one dimensional environments is tractable by applying a plane sweep technique to space time polygonal obstacles. Fig. 2.5 contains an example of two dimensional space time diagram in which time dependent obstacles are represented as polygons. The point robot as well as the obstacles exist in a one dimensional environment, that is, a line, which is represented by the vertical axis in the diagram. The horizontal axis represents time. In the diagram, rectangle A represents a line segment in the line that exists only for a certain interval of time $[t_1, t_2]$. Polygons B and C represents a line segment in the environment whose length varies as a function of time. Object D represents a line segment that moves at a constant speed in a fixed direction in the environment.

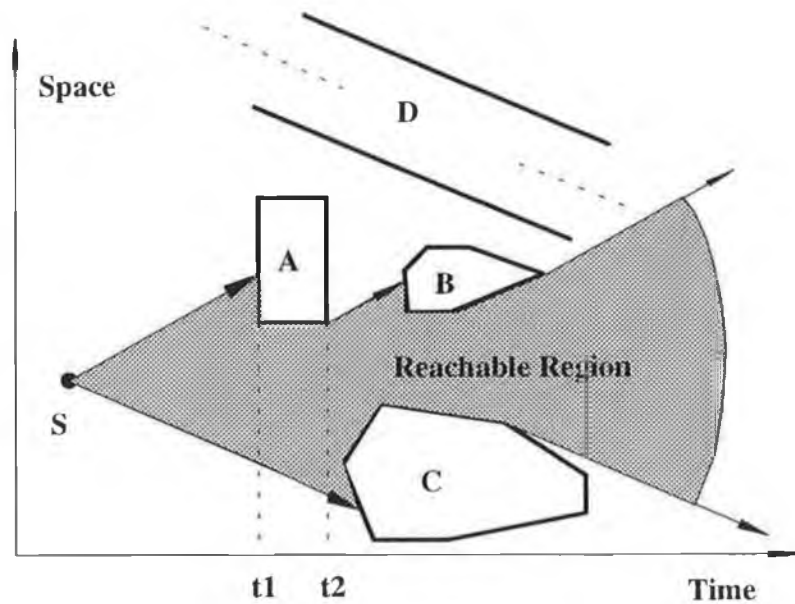


Fig. 2.5: Space Time Diagram for One Dimensional Dynamic Objects

Given a starting point in space time (a location and a time), the set of points in space time that can be reached from the starting point is called a reachable region. When the point robot is subject only to a speed bound, the robot can only reach a region bounded by two rays emanating from the starting point in space time (that is, a cone in space time whose vertex is at the starting point). However, due to dynamic obstacles in space time, the reachable region is a part of the cone as illustrated in Fig. 2.5. The shaded area in Fig. 2.5 represents the set of points in space time that are reachable from starting point S . In two dimensional space time, the reachable region can be efficiently computed in $O((n+s) \cdot \log(n))$ time, where n is the total number of vertices of the polygonal obstacles and s is the number of intersections between obstacle edges.

Once the reachable region has been identified, a time minimal motion to any point can be easily computed. Likewise, objects in two dimensional dynamic environments can be represented as three dimensional space time objects and the reachable region can similarly be defined as a volume in three dimensional space. However, construction of a reachable region in three dimensional space time is substantially more complicated than the case of two dimensional space time. Sutner and Maase [Sutner, K. 1988]

have shown that it takes $O(n^7)$ to construct the reachable region for a single polyhedral dynamic object in three dimensional space time, where n is the number of edges of the polygon. From this result, it seems very difficult to deal with more than a single polyhedral object in three dimensional space time.

O'Dunlaing [O'Dunlaing, C. 1987] discusses a similar problem with an acceleration bound. The problem of moving a particle (a point robot) placed on a line from one location on the line to another location without colliding with two particles that are moving at both ends is considered. The motions of the two obstacle particles are defined by piecewise parabolic functions. It is shown that the problem can be solvable in $O(n^2)$ time, where n is the number of pieces that define the two obstacles motion.

Kant and Zucker [Kant, K. 1986] have applied the idea of avoiding one dimensional moving obstacles to the problem of avoiding moving obstacles in two dimensional environments. The problem is divided into two subproblems: the Path Planning Problem (PPP) among stationary obstacles and Velocity Planning Problem (VPP) along a fixed path. In the first problem, that is, PPP, a path is planned among all the stationary obstacles while ignoring all the moving obstacles. For example, in Fig. 2.6, obstacles A, B, C and D are stationary obstacles, while obstacles E and F are moving in the marked direction. Path SMNOPG is a collision free path with respect to stationary obstacles A, B, C and D.

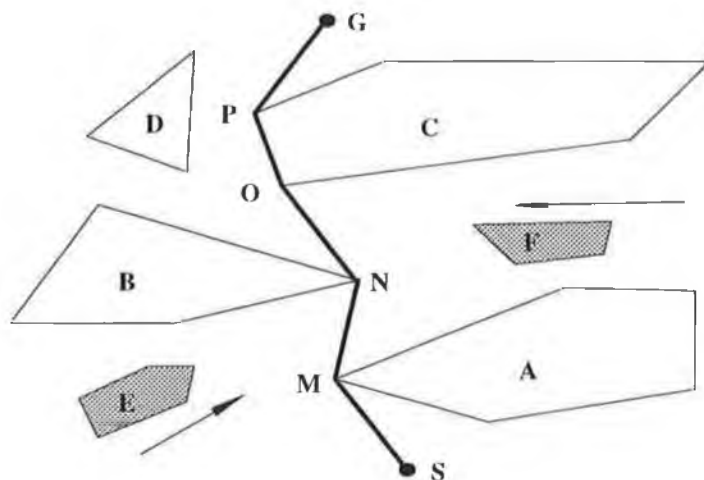


Fig. 2.6: Plan a path among stationary obstacles, while ignoring moving obstacles.

In the second part (VPP), moving obstacles are mapped in space time, where the horizontal axis represents the path specified in the first part (PPP). As shown in Fig. 2.7, shaded areas represent regions through which the robot may not pass when following the path computed in the first part. The positions of these regions influence the choice of the speed. The shapes of these regions may be complicated. Therefore, they are approximated by enclosing rectangles to facilitate the second part (VPP). A space time path that satisfies the maximum speed constraint of the robot is adopted as a final solution. A graph search method instead of plane sweep may be used to find a feasible path. The robot (here, the robot is considered as a point) makes a series of vertex to vertex transitions, as illustrated in Fig. 2.7. Here a vertex means a vertex of a space time polygon. A space time path SXYZ determines speed profile along path SMNOPG.

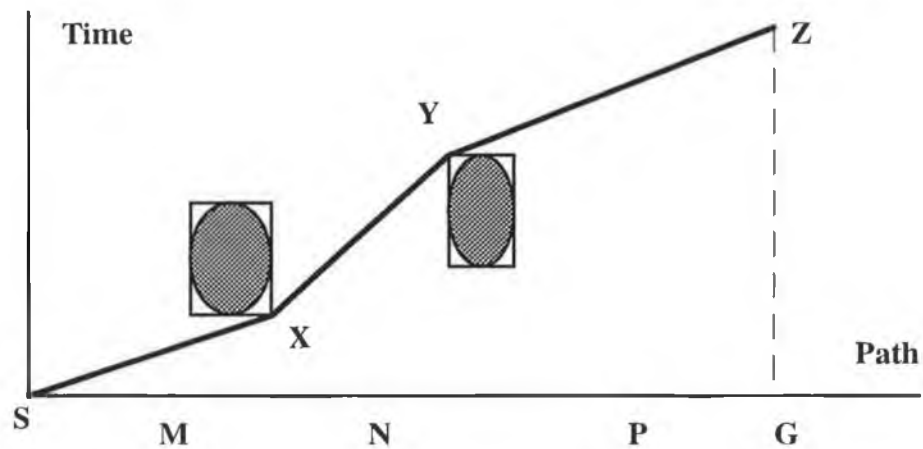


Fig. 2.7: Plan a motion in space time

As Kant and Zucker pointed out, there are cases in which this approach fails to generate a motion even when one exists. This happens, for example, when a part of a moving obstacle is always coincident with the path determined in the first stage. This is because the path is fixed in the second stage, thus the robot is not allowed to circumnavigate the moving obstacles. Also, since the path is fixed in the first stage, the method cannot easily incorporate a moving destination point.

Some studies have been made based on the path velocity decomposition. Lee [Lee, B. 1987] uses a similar idea of path velocity decomposition to plan motions of two robots. Tournassoud [Tournassoud, P. 1988] solves a one dimensional moving obstacle avoidance problem by using visibility graph similar to the graph used by Kant and Zucker [Kant, K. 1986]. Griswold and Eem [Griswold, N. 1990] takes uncertainty of the moving obstacles into account while using the same principle for path planning. Pan and Luo [Pan, T. 1990] also apply the idea of path velocity decomposition to deal with moving obstacles in two dimensional environments. The idea of traversability vectors is used to determine if a given motion segment is free of collision with moving obstacles. Kyriakopoulos and Saridis [Kyriakopoulos, K. 1990] have extended the idea to deal with unexpected moving obstacles.

Erdmann and Lozano-Perez [Erdmann, M. 1987] represent the motions of moving obstacles using a set of slices which embody space time. The slices represent configuration spaces at particular times. The times are those at which some moving obstacles change velocity. A motion consists of a series of straight motion segments each of which starts at a vertex of an obstacle in one slice and terminates at a vertex of an obstacle in the next slice. Between two vertices, the moving obstacle moves in a straight line with a constant speed. As a result, along the final path, the robot changes velocity only at some of the vertices of the obstacles when some obstacle's velocity changes. The approach is complete (that is, the approach can always generate a path if one exists) when the topology of the free space does not change (that is, the obstacles do not merge or split) and runs in $O(r \cdot n^3)$ time, where n is the total number of edges of the obstacles in the environment and r is the number of slices constructed. When the obstacles do not change their velocities at all, the robot is to move at a constant speed from the starting position to the destination position. For the problem of co-ordinating multiple mobile robots as is done by Erdmann and Lozano-Perez, this does not seem to cause a problem, since all obstacles will eventually come to stop.

Reif and Sharir [Reif, J. 1985] show that the problem of deciding whether a motion exists in the presence of obstacles moving with fixed velocities without rotation in a

two dimensional environment is solvable in polynomial time in the number of vertices of all the obstacles and exponential time in the total number of obstacles in the environment. They use the property that whenever there is a path in space time, there is a polygonal path that bends only at edges of space time obstacles. For each vertex of each obstacle, they keep track of all time intervals in which the vertex is reachable.

Shih, Lee and Gruver [Shih, C. 1990] represent free space in space time by a collection of polytopes. A linear programming method is used to identify non-empty polytopes defined by the boundaries of space time obstacles. Then, a graph is constructed to represent the adjacency relationship between the non-empty polytopes. Finally, graph search is performed to generate a piecewise linear motion in free space. Their method has advantages over other approaches, it can be generalised to higher dimensions as well as handle overlaps of moving obstacles in a uniform manner. In [Warren, C. 1990], space time is also used to represent moving obstacles. The artificial potential field function in space time is used to search for a path.

In [Fujimura, K. 1992] [Fujimura, K. 1989], the space time is used to solve the constrained motion planning problem in two dimensional dynamic environments, that is, the robot has mass, it takes time for the robot to accelerate from zero velocity to its maximum velocity. A variant of octree is used to represent the three dimensional space time. The space time is subdivided into eight subspaces of equal size called cells, until every cell satisfies one of the following conditions:

1. It contains part of the trajectory of a vertex of an obstacle.
2. It does not contain any part of the trajectory of a vertex, but contains part of the trajectory of an edge of an obstacle.
3. It does not contain any part of the trajectory.
4. It is entirely contained in the trajectory.

Such a variant of an octree can store the shapes of all the space time obstacles exactly and requires less memory than the ordinary octree and hence the graph search is fast.

The concept of C-points (control points) is defined in space time. A sequence of C-points forms a skeleton of the final path. A C-point is an ordered pair consisting of an L-point and a T-point as shown in Fig. 2.8 and Fig. 2.9.

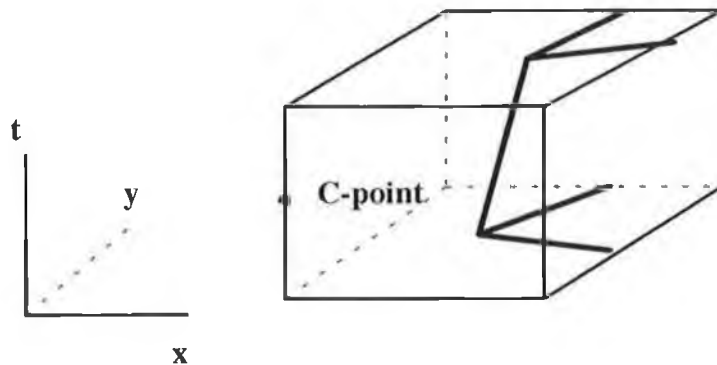


Fig. 2.8: C-point in a cell

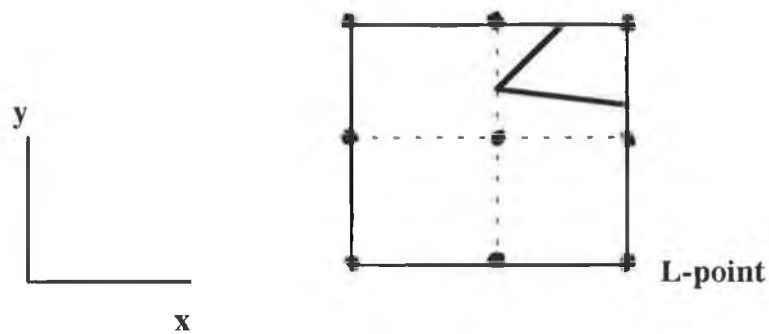


Fig. 2.9: L-point in a cell

The L-point represents the two dimensional projection (x, y) of the C-point, and the T-point represents the time at which the object passes the L-point. The x and y values of an L-point take on discrete values. In every cell, the L-points can only lie at nine locations, as shown in Fig. 2.9. The value of the T-point is assigned during the search stage. The search procedure first chooses the next location to go to from the nine L-

points shown in Fig. 2.9. Next, it determines the appropriate speed. This in turn determines the T-point. Two identical sequence of L-points with different set of T-points represent two different motions. The T-point is determined by the T-point component of the previous C-point on the path and by the acceleration chosen at the L-point component of the previous C-point.

In order to make the search feasible, two restrictions on the choice of acceleration are imposed. The first restriction is that robot can only change its acceleration and direction of motion at L-points. The second is that the robot retains the same acceleration between successive L-points. It is also assumed that the acceleration takes on discrete values.

The graph is constructed by using starting point and destination point and the L-points as nodes. Search is performed on the graph. The approach is not complete, that is, it may fail to generate a path even one exists.

In spite of all the measures that have been applied to make the approach efficient, the approach still involves a large search space and is very inefficient, the set of possible L-points tends to grow rapidly as the number of obstacles increase. For example, suppose that the world consists of $2^n \bullet 2^n \bullet 2^n$ octants of the same size. Then the number of L-points can be as large as $O(2^{3n})$. At each L-point, the next location and the next acceleration are selected using the path conditions. Since this choice must be made at every L-point in the worst case, the execution time of the worst case of the approach is exponential in the number of nodes in the input octree. The closer two obstacles are, the deeper the level of subdivision becomes, resulting in more nodes in the tree. For this reason, the approach is faster in an uncluttered environment [Fujimura, K. 1992].

Recently, Th. Fraichard and C. Laugier [Fraichard, Th. 1993a] [Fraichard, Th. 1993b] [Fraichard, Th. 1993c] extend the space time approach to solve the motion planning problem of a car-like mobile robot which is subjected to holonomic and non-holonomic constraints in dynamic environments. The approach used is an extension

of Kant and Zucker's approach and is called state-time space approach. The problem is divided into two subproblems: path planning and trajectory planning (in fact it is almost the same as Kant and Zucker's division, trajectory planning is velocity planning). The path for the robot is planned among the stationary obstacles without considering the moving obstacles. Next, a three dimensional state-time space is constructed, that is, the length s along the preplanned path is the X axis, the speed along the preplanned path is the Y axis, time t is the Z axis, as shown in Fig. 2.10.

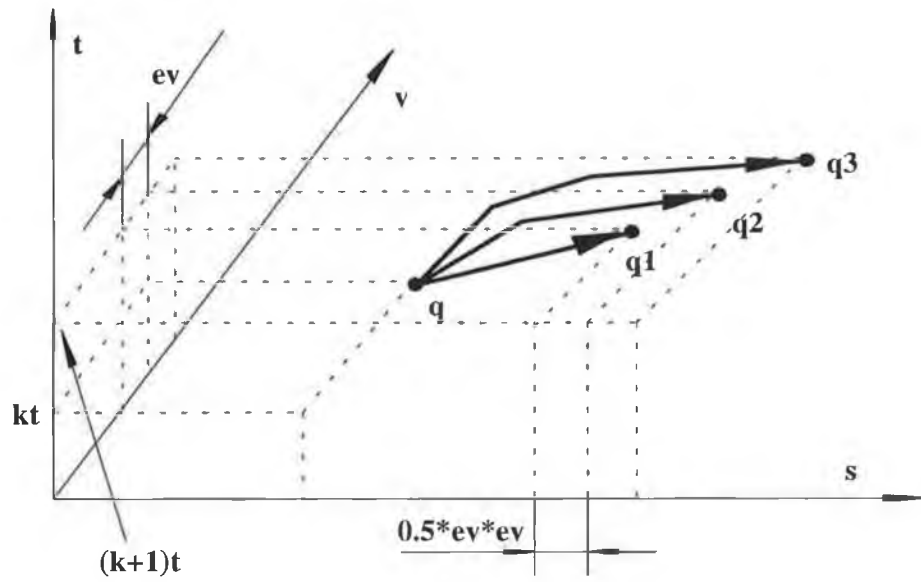


Fig. 2.10: Three dimensional state time space

To make the search efficient, the space time is discretized with time interval t . Let $q(kt)=(s(kt), v(kt))$ be a state of the robot at time kt , the states the robot can reach at next time interval $k(t+1)$ can be calculated by assuming that the acceleration applied to it at time interval kt is either the minimum, null or maximum, that is, three states q_1, q_2, q_3 , as shown in Fig. 2.10. They can be calculated as follows.

$$s_1((k+1)t) = s(kt) + v(kt) * t - \frac{1}{2} A_{\max d} \bullet t^2 \quad (2.3)$$

$$v_1((k+1)t) = v(kt) - A_{\max d} \bullet t \quad (2.4)$$

$$s2((k + 1)t) = s(kt) + v(kt) * t \quad (2.5)$$

$$v2((k + 1)) = v(kt) \quad (2.6)$$

$$s3((k + 1)t) = s(kt) + v(kt) * t + \frac{1}{2} A_{\max a} \bullet t^2 \quad (2.7)$$

$$v3((k + 1)t) = v(kt) + A_{\max a} \bullet t \quad (2.8)$$

where $A_{\max d}$ is the maximum deceleration and $A_{\max a}$ is the maximum acceleration. For every state, there will be 3 successor states, a graph is thus constructed. Searching the graph will generate a trajectory for the robot.

Like Kant and Zucker's approach, this approach may not generate a motion even one exists, that is, it is not complete. To solve this problem, the concept of an adjacent path to a free path is introduced. If the path generated at the first stage is always occupied by some moving obstacles, it will fail to generate a motion. In this case, the robot will move to the adjacent path to avoid the moving obstacles which are on the planned path. By definition, there are one left adjacent path and one right adjacent path for every path. So for every state q , there will be five states into which the robot can move from q : the three states defined above and the two adjacent path states.

2.4.3 Divide and Conquer Strategy

Suzuki and Arimoto [Suzuki, H. 1990] consider using the divide and conquer paradigm for motion planning in the presence of time varying obstacles in two dimensional dynamic environments. First, they encode time varying obstacles in space time using the grid representation. Then, a subgoal is selected in space time such that the subgoal is reachable from the starting position and that the destination position can be reachable from the subgoal. Thus, the original problem is decomposed into two subproblems of smaller size. Each subproblem is further decomposed into two

subproblems in a similar manner. This process is repeated until each subproblem becomes simple enough so that a solution is easily found. A final solution is obtained by concatenating the solutions for all subproblems.

Lamadrid [Lamadrid, J. 1986] has also adopted a recursive strategy for avoidance of moving obstacles in two dimensional dynamic environments. The robot is assumed to move at a constant velocity, while the obstacles' motions are defined by quadratic functions of time. First, a straight line motion from the starting position to the destination position is considered. This motion is checked to see if the robot collides with any of the moving obstacles while moving along the path. If no collision is detected, the path is accepted. Otherwise, an appropriate subgoal is selected based on some criteria. Then, the same procedure is applied recursively with the subgoal as a new destination position to determine a motion to the subgoal.

2.4.4 Accessibility graph

The accessibility graph is proposed by K. Fujimura [Fujimura, K. 1988] [Fujimura, K. 1992]. It is a time varying graph. The approach uses a similar idea to the visibility graph to solve motion planning problems in dynamic environments. Making use of the concept of accessibility, the accessibility graph is defined to represent moving objects in dynamic environments. The accessibility graph is shown to be a generalisation of the visibility graph in the sense that paths to the destination position are found as sequence of edges of the graph. In fact, when all the obstacles have zero velocity, the accessibility graph becomes the visibility graph of these polygonal obstacles. More importantly, if the robot is able to move faster than any of the obstacles, then the graph exhibits a property: a time minimum motion is represented as a sequence of edges of accessibility graph. It will be discussed in Chapter 5 in detail.

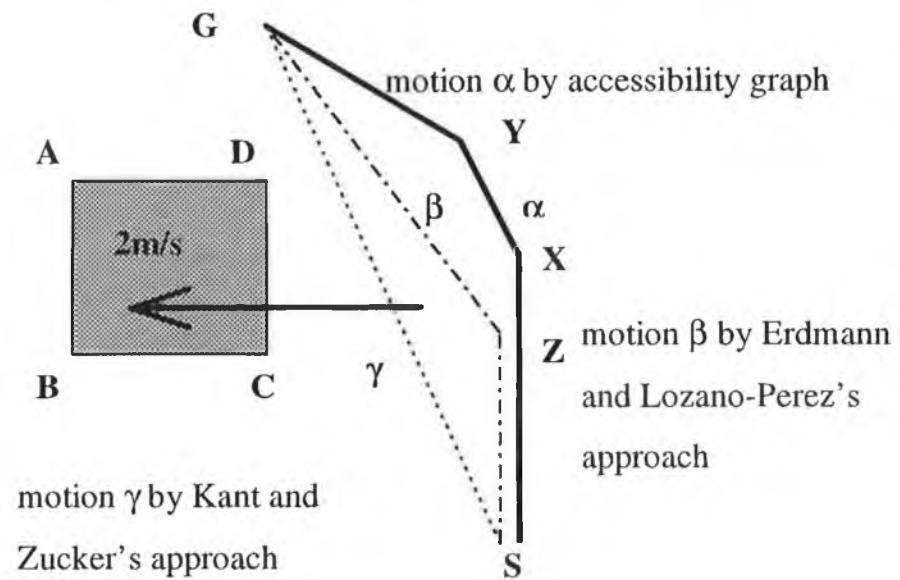


Fig. 2.11: Three motions by different approaches

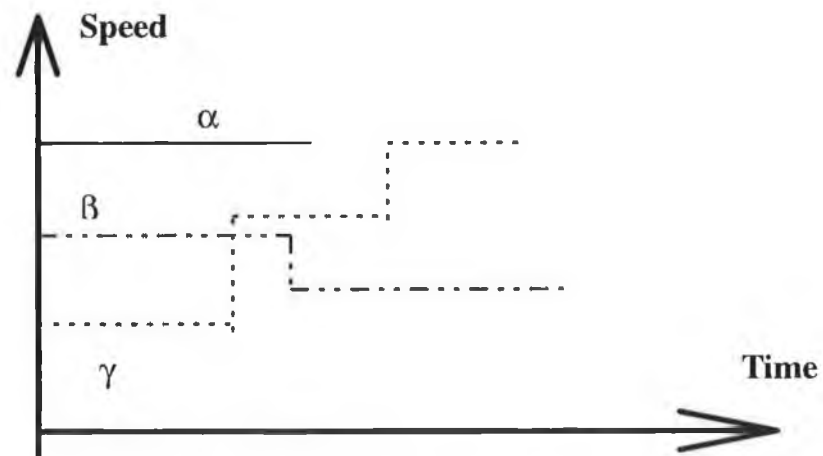


Fig. 2.12: Time Speed Profiles of the Three Motions

The advantages of accessibility graph over Kant and Zucker's space time approach are that the later tries to solve the problem of "where" and "when" in different stages, while the former solves the "where" and "when" problem at the same time. So the accessibility graph is complete, that is, it can always generate a motion if one exists, and it is a minimum time solution. Fig. 2.11 shows the results generated by accessibility graph, Kant and Zucker's approach and Erdmann and Lozano-Perez's approach with one moving obstacle in the environment. Fig. 2.12 shows the velocity

and time curves for the robot with different approaches. It is obvious that the accessibility graph can generate the fastest path, that is, a time minimum path. For Kant and Zucker's approach, the path is a line since there is no stationary obstacles in the environment. It is also obvious that the motions generated by Kant and Zucker's approach and Erdmann and Lozano-Perez's approach are not time minimal.

2.4.5 Collision Avoidance with Moving Obstacles

Sometimes, the motions of moving obstacles are not known precisely ahead of time. Even worse, the robot may encounter moving obstacles unexpectedly while in motion. In such a case, it is important to be able to act quickly so as to avoid imminent collision. Optimisation of the motion is of secondary consideration in such a situation. Reactive behaviour is more important than long term planning of robot motion.

This reactive approach leads to an alternative strategy to dealing with moving obstacles, that is, to keep avoiding obstacles near at hand, hoping that the robot eventually reaches its destination position. The robot avoids obstacles based on a certain set of predetermined rules. One such rule could be "if an obstacle is approaching from the left and there is no obstacle to the right, move toward the right." The motion of the robot is determined by a collection of such heuristic rules with respect to the location of the destination position and locations of nearby obstacles relative to the robot.

In [Meystel, A. 1990], the control system of the mobile robot is a hierarchical system, avoiding the unexpected obstacles will be the pilot's responsibility. Maeda and Takegaki [Maeda, Y. 1988] use a reactive obstacle avoidance scheme based on production rules encoded in fuzzy logic. Koyama et al [Koyama, H. 1991] also consider dynamic obstacles avoidance using fuzzy production rules in the presence of multiple obstacles moving at different speeds. Takeuchi [Takeuchi, T. 1988] uses

fuzzy logic method for the navigation of an autonomous mobile robot among stationary obstacles.

Some other approaches to handling moving obstacles include the following. Tournassoud [Tournassoud, P. 1986] considers a local collision avoidance problem using separating hyperplanes. Kehtarnavaz and Li [Kehtarnavaz, N. 1988] consider a collision free navigation scheme in the presence of moving obstacles with unknown trajectories. An autoregressive model is used to predict the future positions of the obstacles. Kehtarnavaz and Griswold [Kehtarnavaz, N. 1990] use a collision zone for a high collision likelihood space for obstacle avoidance. Yagi, Kawato and Tsuji [Yagi, Y. 1991] has used a conic mirror which enables an omnidirectional view around the mobile robot for moving obstacle avoidance. Basu has introduced a probability model for avoidance of moving obstacles [Fujimura, K. 1992].

Parker represents an obstacle by a circle. A tangent point is a point on the circle at which the robot and the obstacle meet tangentially. The robot moves in the direction of the tangent point at a constant speed. The A^* algorithm is used to search for a motion to the goal position [Fujimura, K. 1992].

2.4.6 Collision Detection Among Moving Objects

There is another related but different category of problems usually called collision detection. This is to answer the question of the form: "Is there a collision between two moving objects?" Although the answer to this problem may not provide a direct solution to collision avoidance problems, it can serve as a useful component of collision avoidance problems as it can report a potential collision between, say a mobile robot and its surroundings.

Cameron [Cameron, S. 1985] discusses three approaches for collision detection for moving objects. The first one is to perform static collision detection repetitively over some period of time. The major difficulty in this approach is to choose appropriate

time steps; if time steps are large, collision may not be detected even when it occurs. If time steps are small, then much computation time is wasted. The second approach is to perform collision detection in space time. Although this method is complete, it may require complex computation to process four dimensional space time. The third approach is to represent moving objects by their sweeping volumes.

Samet and Tamminen [Samet, H. 1985] describe a way to add the time dimension to a CSG tree. By converting a CSG tree to the bintree representation, dynamic collision detection can be efficiently performed. The problem of collision detection is also discussed by Esterling and Van Rosendale [Esterling, D. 1983]. They divide the time dimension as well as the other dimensions recursively to quickly locate the collision position between two moving objects. This idea is further developed by Cameron [Cameron, S. 1990] who reports a collision detection method based on four dimensional space time. He extends a CSG representation into four dimensions and adopts a divide and conquer strategy to locate collision between two moving objects.

2.5 Fuzzy Logic Methods

Fuzzy logic was developed in 1965 by Professor L. Zadeh. It is an extension to traditional logic theory. It makes some problems much easier to solve by allowing a more natural representation of the situation being dealt with. Fuzzy logic has been widely applied in many areas such as process control, pattern recognition, image processing, etc. Only fuzzy logic methods for mobile robots navigation and digital image processing will be reviewed in this part.

2.5.1 Fuzzy Logic Methods for Mobile Robots Navigation

Fuzzy logic methods are usually used for low level navigation of a mobile robot. In [Meystel, A. 1990], the mobile robot control system is a hierarchical system usually with three levels. The highest, most abstract (and least time critical, precise or

detailed) level is the planner, which operates on a global mission to determine the connected subgoals or tasks to achieve an assigned objective. The medium level is the navigator, which utilises a detailed plan or map to evaluate an obstacle free local path that optimises a performance criterion (say, shortest path, minimum time etc.) to produce motion and velocity trajectory. The lowest level is the pilot, it executes the commands from the navigator and avoids the unexpected obstacles which are unknown by the planning and navigation stages. Fuzzy logic plays the most important role at pilot level. Fuzzy logic is used to generate the velocity and acceleration for the robot with rules like: if the velocity is big and the distance from an obstacle is near then deceleration is big, etc. Fuzzy logic is also used to avoid unexpected obstacles at the pilot level. In [Kemal, M. 1989], a similar method using fuzzy logic is also introduced.

R. Garcia Rosa, D. Kumpel and M. C. Garcia-Alegre [Rosa, R. 1989] [Rosa, R. 1990] reports a local navigation system based on fuzzy logic control. They use fuzzy rules to navigate a mobile robot along a curved wall. A similar system is also reported by J. Gasos, et al [Gasos, J. 1990]. The rules used are similar to the rules used in [Meystel, A. 1990].

C. J. Harris and C. G. Moore report a self organised fuzzy control system for an autonomous guided vehicle [Harris, C. 1989]. The control system is used to navigate the autonomous guided vehicle to park in a given place. The input variables of the system are the distance and velocity of the vehicle. The output of the system is the steering angle of the guided vehicle. The parking is divided into three types: forward parking, reverse parking and reverse garage or slot parking. Different control rules are used for different types of parking. Intelligent identification (fuzzy identification) of the model is performed to organise the control rules.

Maeda and Takegaki [Maeda, Y. 1988] report a reactive moving obstacle avoidance scheme based on production rules encoded in fuzzy logic. In their simulation, relative speeds of moving obstacles are represented by fuzzy numbers. These fuzzy numbers, together with distances to the obstacles between the robot, indicate how dangerous

the obstacles are and are then used to determine the direction of motion of the robot. 112 rules are used to achieve flexible obstacle avoidance. They assume that the robot encounters one moving obstacle at a time and that the motion of the robot is controlled only by steering angles. Koyama et al [Koyama, H. 1991] also consider dynamic obstacle avoidance using fuzzy production rules in the presence of multiple obstacles moving at different speeds.

2.5.2 Fuzzy Logic Methods for Digital Image Processing

Fuzzy logic based methods have been used in digital image processing [Pal, S. 1990] [Pal, S. 1990] [Goetcherian, V. 1980] [Pal, S. 1981] [Pal, S. 1988] [Pal, S. 1989] [Feng, J. 1991] [Li, H. 1990] [Yan, J. 1992] [Yan, J. 1992]. Fuzzy logic based methods for image smoothing, image enhancement, image edge detection and image motion detection and analysis are very effective.

All the fuzzy logic methods for digital image processing are very similar. In [Pal, S. 1988], the digital image is first fuzzified, the concept of fuzzy compactness of the image is defined. Several algorithms for digital image thresholding by optimisation of fuzzy compactness are discussed. In [Pal, S. 1981], image enhancement by fuzzy logic is discussed. J. Yan and M. Ryan report all the fuzzy logic methods for image processing [Yan, J. 1992]. They report using the fuzzy index of compactness to recognise the objects in an image and to detect the motion of the objects in two sequence images in another paper [Yan, J. 1992]. The method can detect the translation of objects. However, it can not be used to detect the rotation of the objects. Details will be discussed later in chapter 7.

2.6 Neural Networks for Mobile Robots Navigation

Neural networks have recently been used in mobile robot navigation [Pourboghrat, F. 1990] [Fukuda, T. 1990] [Ho, A. 1990] [Sorouchyari, E. 1989] [Nagata, S. 1990]

[Porto, V. 1991] [DeMuth, G. 1990] [Dodds, D. 1990]. Neural networks have advantages over the traditional computer systems in the way that they work in parallel manner and are suitable for real time applications. In [Pourboghra, F. 1990], an approach for generating shortest path in static environments by neural networks is reported. The task of finding the shortest path in a static environment is formulated into an optimization problem for which a recurrent neural network, similar to Hopfield's, is used. The output of the units in the network correspond to the position of some intermediate points on the path which would be connected by straight line segments. The steady-state values of the outputs of the units in the network correspond to the coordinates of the position of the intermediate points on the shortest collision free path to the target. To derive the dynamics of the network, first an energy function corresponding to the distance from the target, the total length of the path and the degree of closeness to the obstacles is derived. The dynamics of the network are then derived according to a gradient algorithm. Experiments show that the outputs of such a network would always reach a set of steady-state values, and these values minimize the defined energy function, that is, the shortest path is always generated by the network.

In [DeMuth, G. 1990], neural networks are used to navigate an underwater autonomous mobile robot to avoid stationary and moving obstacles. The inputs of the neural network consists of beam outputs from a forward-looking sonar, and difference between current and desired values of the autonomous mobile robot course and speed are inputs to normal navigation and control. The neural network outputs are the robot's rudder angle and propulsion power and basic maneuvering characteristics of the robot are incorporated in the model. When a distant obstacle is first detected, the robot propulsion is not changed (that is, the robot's speed is not changed) and the robot rudder angle is changed slightly. If the robot comes closer to the obstacle, a larger rudder angle correction plus speed change are applied. When a stationary obstacle is detected by the forward looking sonar, the neural network places a lower limit on the cross range distance of the obstacle with respect to the robot current heading. The neural network outputs a correction command if $R \sin \theta$ becomes less than a threshold value (where R is the range to the navigation obstacle

and θ is the angle of the obstacle with respect to the robot heading). Fast closing rate of a moving obstacle also initiates corrective change of the robot rudder angle and speed. A second neural network is used for this and a Boolean combiner uses class vectors from both the static and dynamic obstacle classifiers to reconcile the avoidance maneuver to be taken. Obstacle avoidance is accomplished using a proximity detector to avoid stationary obstacles and a rate detector to avoid dynamic obstacles. The detections are made using two dimensional masked binary filters implemented as multilayer neural networks in the classification mode. Simulation results show that the robot successfully avoided collision with all the obstacles in the environment. In [Porto, V. 1991], a similar neural network approach is reported for obstacle avoidance for underwater mobile robots.

In [Ho, A. 1990], an approach to plan a near optimal collision free path for a mobile robot on binary or varied terrains by neural networks is reported. The motion planning problem is formulated as a classification problem in which class labels are uniquely mapped onto the set of maneuverable robot motions. The neural network planner is an implementation of the popular adaptive error back-propagation model. The motion planner learns to plan a “good”, if not optimal, collision free path from supervision in the form of training samples. A multi-scale representational scheme, as a consequence of a vision-based terrain sampling strategy, maps physical problem domains onto an arbitrarily chosen fixed size input layer of an error back-propagation network. The mapping not only reduces the size of the computation domain, but also ensures applicability of a trained network over a wide range of problem sizes. Simulation results of binary terrain navigation show that the neural network planner performs effectively in unknown environments.

2.7 Summary

The above review shows that the main problems in geometrical navigation in two dimensional environments have been effectively solved, while the problems in three dimensional environments still need researching. The approximate approach has been

proposed to attack the problem [Latombe, J. 1990]. Because of the computation complexity, simple and easy approaches are needed to solve the problems.

For dynamic navigation, the methods used in [Meystel, A. 1990] [Meystel, A. 1986] are effective and practical. But the time for the mobile robot to turn around a turning point is determined by the angle the robot is about to turn. This is not very accurate in a real situation. And it makes estimating the time for the rest of the motion from the present position very difficult in some cases (a fictitious goal is needed). Furthermore, the velocity of the robot at a turning point is decided by the angle that the robot is going to turn. To find this angle, the future point of the path must be known. This will lead to depth first search which is not efficient for real applications. A more practical method like smoothing the path at a turning point is possible. The velocity of the robot at the smooth part can be decided by the curvature of the path and the minimum distance between the robot and any of the obstacles.

As for motion planning in dynamic environments, the review shows that the main problem in one dimension has been efficiently solved, while the problems in higher dimensional environments are still a subject of active research. A number of approaches have been proposed to solve the problems. Some of them can not generate optimal time paths, because they plan the motion of the mobile robot in two stages. For the accessibility graph approach, it plans the motion of the mobile robot in one stage, so it can generate the time optimal path. But it can only generate a path for a massless mobile robot, that is, it takes no time for the robot to accelerate from zero speed to its maximum speed. And such an assumption is not practical. Although a space time approach is proposed by K. Fujimura [Fujimura, K. 1992] [Fujimura, K. 1989], the memory taken by the space time approach is of exponential order and it is very inefficient. To make the problem worse, for problems in three dimensional dynamic environments, the space time approach will become a four dimensional problem and it is very difficult to represent and will become too abstract. So a new approach like the accessibility graph approach which solves the motion planning problems of a mobile robot with mass will make the problem easier.

When the planning environment is given as a two dimensional binary image, the effective approach to date is the approximate cell decomposition approach and it is effective and efficient in stationary environments. For dynamic environments given as a two dimensional binary image, the approximate cell decomposition approach will not work. So approaches which can solve such problem in dynamic environments are needed.

All the above areas will be researched in later parts of this thesis.

Chapter

3. Shortest Path Planning

3.1 Introduction

In chapter 1, the basic motion planning problem is defined and its extensions to more practical problems are discussed. In this chapter, we will focus on solving the basic motion planning problem, that is, planning a path along which the robot can move from the starting position to the goal position without any collision with the obstacles in the environment. There may exist many different paths which can satisfy the above conditions. Some criterion must be used to select the best one. Since only the geometrical factors of the robot and the environment are taken into consideration, we would like to find the path with minimum total length, that is, the shortest path.

The problem of finding a collision free path of minimum total length in two and three dimensional environments has attracted much attention. This chapter will focus on the problem of finding the shortest path in two dimensional environments. To simplify the problem, the size of the robot will not be taken into consideration, that is, the problem will be solved in configuration space. In this chapter, the concept of

configuration space is discussed in detail. It is the basic tool to solve the shortest path problem and is also required to solve other motion planning problems which we come across later. Then the visibility graph approach which can generate the shortest path will be discussed in detail. The tangent graph approach which can generate the smoothed shortest path is also discussed. The visibility graph and tangent graph play an important role in solving other problems in the rest of this thesis. Simulation results by computer will also be given in this chapter.

3.2 Configuration Space

The underlying idea of configuration space is to represent the mobile robot as a point in an appropriate space, the robot's configuration space, and to map the obstacles in the planning environment in this space. This mapping transforms the problem of planning the motion of a dimensional robot into the problem of planning the motion of a point. It makes the constraints on the motions of the robot more explicit and simplifies the motion planning problem.

3.2.1 Definition of Configuration Space

Configuration space was first introduced in mechanics. We consider a rigid object $\mathbf{A}$, the mobile robot, moving in a physical workspace (environment) $\mathbf{W}$ (see Fig. 3.1). We represent $\mathbf{W}$ as the N -dimensional Euclidean space $\mathbf{R}^N$, where $N=2$ or 3 , equipped with a fixed Cartesian coordinate system, or frame, denoted by $\mathbf{F}_w$. We represent $\mathbf{A}$ at a reference position and orientation as a compact subset of $\mathbf{R}^N$. A moving frame, $\mathbf{F}_A$, is attached to $\mathbf{A}$ so that each point in the robot has fixed coordinates in $\mathbf{F}_A$. The origins of $\mathbf{F}_w$ and $\mathbf{F}_A$ are denoted by $\mathbf{O}_w$ and $\mathbf{O}_A$ respectively. $\mathbf{O}_A$ is called the reference point of $\mathbf{A}$. We assume that $\mathbf{A}$ has no symmetry, in particular, it is not a single point.

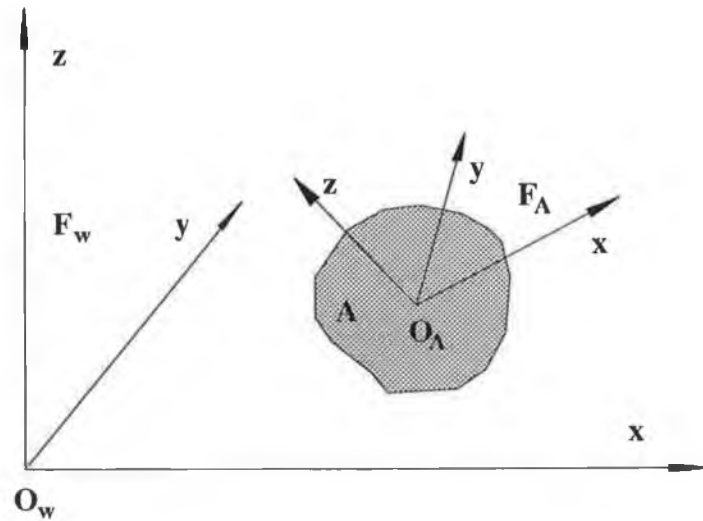


Fig. 3.1: Configuration of A specifies the position and orientation of F_A with respect to F_w

The configuration space of A is defined as:

A configuration q of A is a specification of the position and orientation of F_A with respect to F_w . The configuration space of A is the space C of all the possible configurations of A . A unique configuration of C is arbitrarily selected and is called the reference configuration of A . It is denoted by 0 .

The configuration space C is intrinsically independent of the choice of the frames F_A and F_w . Only the representation of C depends on these frames.

The subset of W occupied by A at configuration q is denoted by $A(q)$. A point a of A is denoted by $a(q)$ in W when A is at configuration q . Thus for any configuration q and q_1 , $a(q)$ and $a(q_1)$ are the same point in A , but in general do not necessarily coincide in W . More generally, if x is a geometric feature (e.g. a point, a vector, a set of points) in A , $x(q)$ denotes the same feature in W when A is at configuration q .

It is sometimes convenient to interpret a configuration as a rigid body transformation, that is, a transformation that preserves distance and orientation. If N of W is 2, that

is, the space is a two dimensional space, it is obvious that a tuple (x, y, θ) can denote the position and orientation of a rigid object in $\mathbf{W}$. So the configuration space of a rigid object in two dimensional space is a three dimensional Euclidean space. In the similar way, if the Euler representation is used in three dimensional space, the configuration space of a rigid object in three dimensional space will be a six dimensional Euclidean space.

3.2.2 Obstacles in Configuration Space

As mentioned in the definition of configuration space, a configuration can be interpreted as a rigid body transformation. When a robot $\mathbf{A}$ in workspace $\mathbf{W}$ is denoted as a point in its configuration space $\mathbf{C}$, the robot is said to be transformed from a rigid object into a point. To plan the motion of the robot in $\mathbf{W}$, it is necessary to transform the obstacles in $\mathbf{W}$ into $\mathbf{A}$'s configuration space. Suppose that there is an obstacle $\mathbf{B}$ in $\mathbf{W}$. In $\mathbf{C}$, some points correspond to placements of $\mathbf{A}$ in which $\mathbf{A}$ overlaps with $\mathbf{B}$ in $\mathbf{W}$. These points are called illegal. While other points in $\mathbf{C}$ corresponding to placements of $\mathbf{A}$ in which $\mathbf{A}$ does not overlap with $\mathbf{B}$ are called legal points. More specifically, the set of points in $\mathbf{A}$'s configuration space $\mathbf{C}$ that correspond to the placements of $\mathbf{A}$ where $\mathbf{A}$ overlaps with $\mathbf{B}$ in $\mathbf{W}$ is called a configuration space obstacle for $\mathbf{A}$ due to $\mathbf{B}$ and denoted by $\mathbf{CB}$. An illegal point in $\mathbf{C}$ is also said to be inside $\mathbf{CB}$. In this way, an obstacle $\mathbf{B}$ in the workspace $\mathbf{W}$ is transformed into $\mathbf{A}$'s configuration space $\mathbf{C}$. The problem of motion planning for $\mathbf{A}$ in workspace $\mathbf{W}$ is transformed into the problem of finding a path for the point in $\mathbf{A}$'s configuration space $\mathbf{C}$ such that every point on the path is legal.

The basic motion planning problem defined in chapter one can be redefined in configuration space as follows:

Let $\mathbf{A}$ be a single rigid object, that is, the robot, moving in Euclidean space $\mathbf{W}$, called workspace, represented as $\mathbf{R}^N$, with $N=2$ or 3 .

Let $\mathbf{B}_1, \mathbf{B}_2, \dots, \mathbf{B}_q$ be fixed rigid objects distributed in $\mathbf{W}$. They are called obstacles.

Assume that both the geometry of $\mathbf{A}, \mathbf{B}_1, \mathbf{B}_2, \dots, \mathbf{B}_q$ and the locations of $\mathbf{B}_1, \mathbf{B}_2, \dots, \mathbf{B}_q$ in $\mathbf{W}$ are accurately known. Assume further that no kinematics constraints limit the motions of $\mathbf{A}$.

The problem is: Given an initial configuration and a goal configuration of $\mathbf{A}$ in $\mathbf{A}$'s configuration space $\mathbf{C}$, generate a path τ specifying a continuous sequence of points in $\mathbf{C}$ which are not inside $\mathbf{CB}_1, \mathbf{CB}_2, \dots, \mathbf{CB}_q$, starting at the initial point, and terminating at the goal point. Report failure if no such path exists.

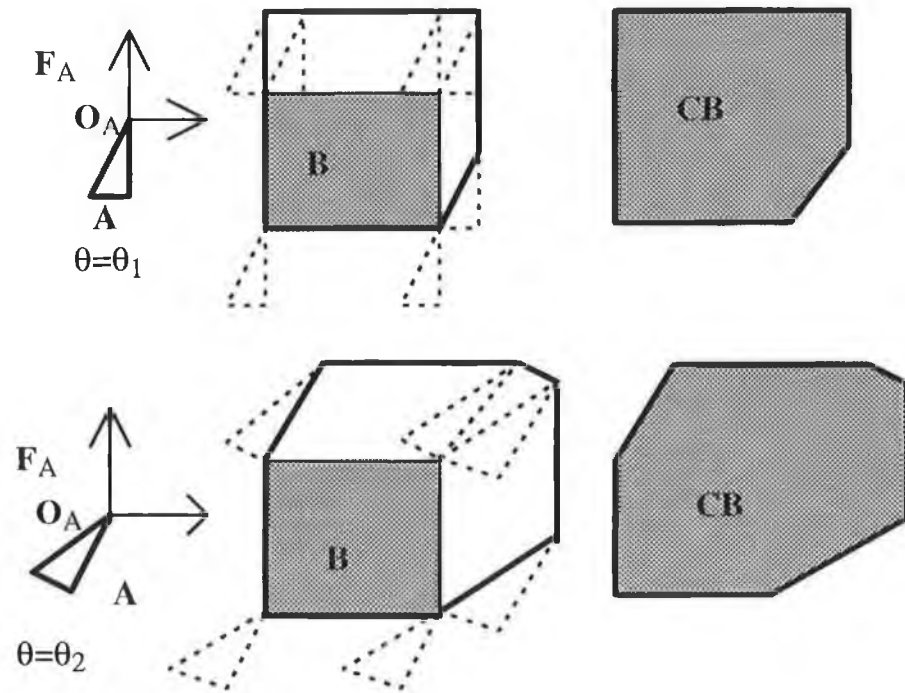


Fig. 3.2: Obstacles in two dimensional configuration space

In two dimensional planning environments, if the orientation of the robot is not considered, the configuration space will become a two dimensional Euclidean space.

Fig. 3.2 illustrates the case where both the robot and obstacle are all convex polygons. Fig. 3.3 illustrates the case where the robot is a disc and the obstacle is a polygon.

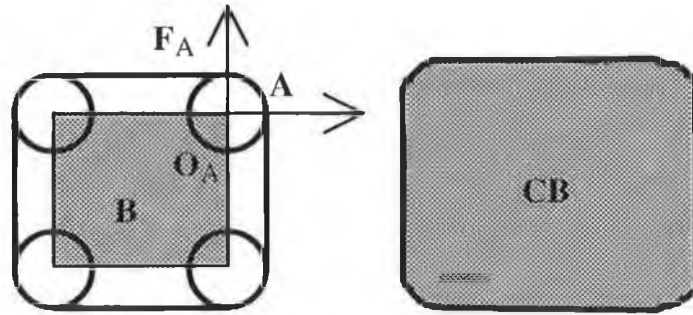


Fig. 3.3: Obstacle in configuration space of a disc

From Fig. 3.2 and Fig. 3.3, it is obvious that if the robot is considered as a polygon, the obstacles in configuration space are related to the orientation of the robot; while if the robot is considered as a circle (a disc), the obstacles in configuration space are not related to the orientation of the robot. To simplify the problem, if there is no special specification, the robot is considered as a circle. This assumption is practical, since a circle with minimum radius which can contain the robot inside it can be drawn to represent the robot. The path along which that circle can pass without any collision with any of the obstacles must be the path along which the robot can pass without collision with the obstacles.

There is still a problem when the robot is considered as a circle. As shown in Fig. 3.3, a polygon obstacle in physical space is transformed into a general polygon in configuration space, that is, a polygon with lines and arcs as its edges. It is not convenient to construct the visibility graph in a configuration space with such obstacles. A polygon can be used to replace that general polygon as shown in Fig. 3.4. The polygonal obstacle is expanded by the size of the robot's radius R . This method of expanding a polygonal obstacle has an advantage that the expanded size

can be selected bigger than the real size of the robot so that the robot can move in a path which has some clearance from the obstacles for the sake of safety. This convention will be used in this thesis.

If the coordinate values of X and Y of the vertices of A, B, C, D and E in Fig. 3.4 are all known, it is easy to calculate the coordinate values of X and Y of vertices EA, EB, EC, ED and EE . As shown in Fig. 3.5, suppose $(X_A, Y_A), (X_B, Y_B)$ and (X_E, Y_E) are the coordinate values of A, B and E respectively, coordinate values X_{EA} and Y_{EA} can be calculated as follows.

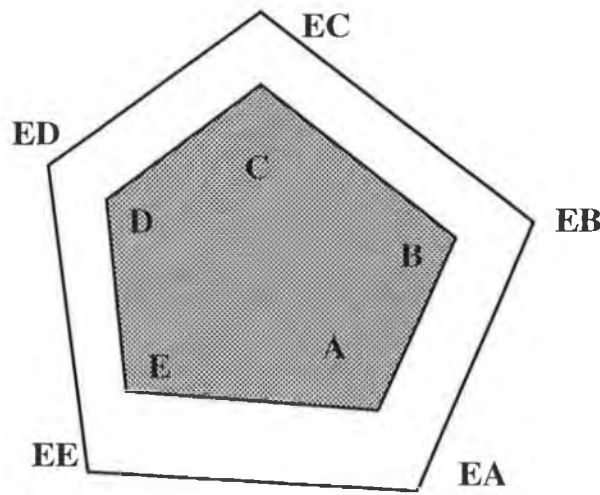


Fig. 3.4: Expansion of a polygonal obstacle

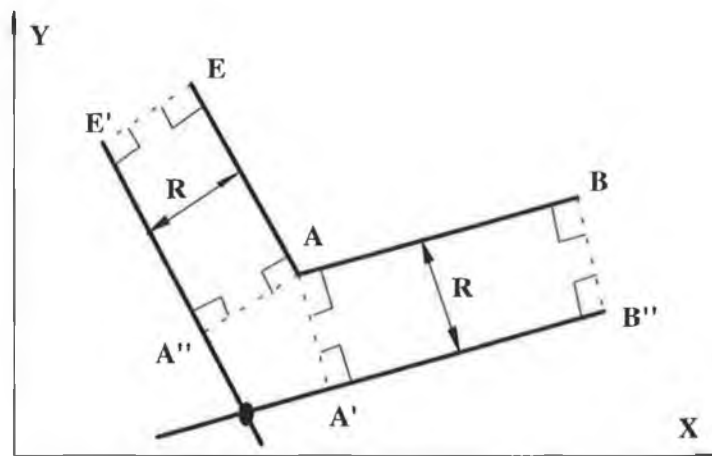


Fig. 3.5: Calculation of polygonal obstacles in configuration space

Line segments E'A'' and A'B'' are parallel to polygon edges EA and AB with distance R away from EA and AB respectively. Coordinate values of E', A', A'' and B' can be calculated by using the following equations:

$$X_{E'} = X_E + \frac{Y_A - Y_E}{\sqrt{(Y_A - Y_E)^2 + (X_E - X_A)^2}} R \quad (3.1)$$

$$Y_{E'} = Y_E + \frac{X_E - X_A}{\sqrt{(Y_A - Y_E)^2 + (X_E - X_A)^2}} R \quad (3.2)$$

$$X_{A''} = X_A + \frac{Y_A - Y_E}{\sqrt{(Y_A - Y_E)^2 + (X_E - X_A)^2}} R \quad (3.3)$$

$$Y_{A''} = Y_A + \frac{X_E - X_A}{\sqrt{(Y_A - Y_E)^2 + (X_E - X_A)^2}} R \quad (3.4)$$

$$X_{A'} = X_A + \frac{Y_B - Y_A}{\sqrt{(Y_B - Y_A)^2 + (X_A - X_B)^2}} R \quad (3.5)$$

$$Y_{A'} = Y_A + \frac{X_A - X_B}{\sqrt{(Y_B - Y_A)^2 + (X_A - X_B)^2}} R \quad (3.6)$$

$$X_{B''} = X_B + \frac{Y_B - Y_A}{\sqrt{(Y_B - Y_A)^2 + (X_A - X_B)^2}} R \quad (3.7)$$

$$Y_{B''} = Y_B + \frac{X_A - X_B}{\sqrt{(Y_B - Y_A)^2 + (X_A - X_B)^2}} R \quad (3.8)$$

With $(X_{E'}, Y_{E'})$, $(X_{A''}, Y_{A''})$, $(X_{A'}, Y_{A'})$ and $(X_{B''}, Y_{B''})$ calculated by the above equations, the coordinate values of EA can be calculated by finding the coordinate values of the intersecting point of line $E'A''$ and line $A'B''$.

$$\begin{cases} Y - Y_{E'} = \frac{Y_{A''} - Y_{E'}}{X_{A''} - X_{E'}} (X - X_{E'}) \\ Y - Y_{A'} = \frac{Y_{B''} - Y_{A'}}{X_{B''} - X_{A'}} (X - X_{A'}) \end{cases} \quad (3.9)$$

Coordinate values of EB, EC, ED and EE can be calculated in a similar way. The problem of motion planning for a circular robot in an environment with polygonal obstacles is now becoming the problem of motion planning for a point in an environment with expanded polygonal obstacles.

3.3 Visibility Graph

3.3.1 Definition of the Visibility Graph

As mentioned above, the motion planning for a mobile robot in a two dimensional environment becomes the motion planning for a point in a two dimensional configuration space. The visibility graph plays a very important role in planning the path for a point robot in two dimensional configuration space. Given an environment with polygonal obstacles $B_1, B_2, \dots, B_k$, the mobile robot A is required to move from the starting position S to the destination position D without colliding with any of the obstacles. The polygonal obstacles in the environment are first expanded into

obstacles $\mathbf{CB}_1, \mathbf{CB}_2, \dots, \mathbf{CB}_k$ in the configuration space of the robot. The starting position and destination position are also converted into configuration space as the starting point S and destination point D .

The visibility graph is the non-directed graph $\mathbf{G}$ specified as follows:

- $\mathbf{G}$'s nodes are S, D and the vertices of obstacles $\mathbf{CB}_1, \mathbf{CB}_2, \dots, \mathbf{CB}_k$ (at finite distance).
- Two nodes of $\mathbf{G}$ are connected by a line segment if and only if either the line segment joining them is an edge of $\mathbf{CB}_1, \mathbf{CB}_2, \dots, \mathbf{CB}_k$, or it does not intersect with the interior of any of $\mathbf{CB}_1, \mathbf{CB}_2, \dots, \mathbf{CB}_k$.

Fig. 3.6 shows the visibility graph of a simple configuration space.

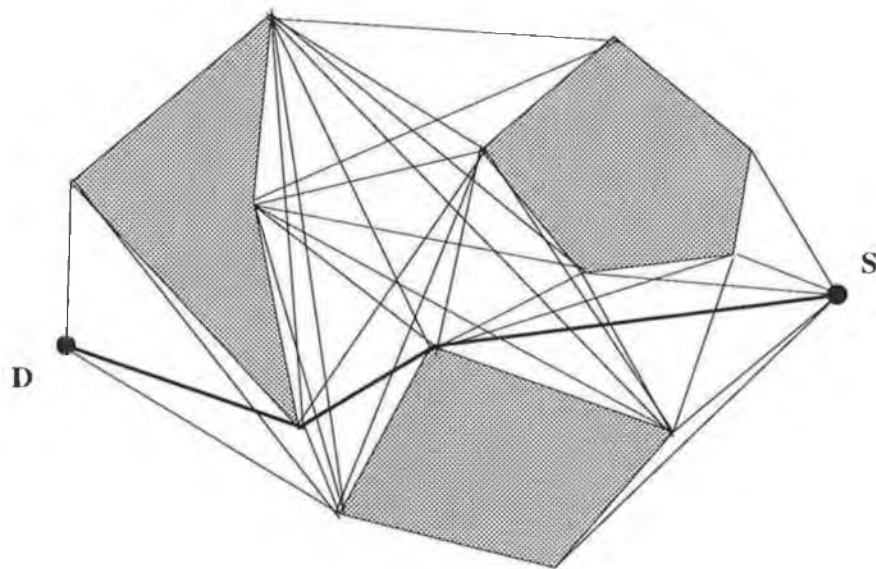


Fig. 3.6: Visibility Graph

3.3.2 Graph Search Algorithms

Once the visibility graph is constructed, graph search technique can be used to find a path between the starting point and the destination point. Various search algorithms such as A^* and Dijkstra have been proposed for the graph search. A^* is the most effective one. Here, an A^* -like algorithm is used to search for the shortest path. This algorithm constructs and searches the visibility graph at the same time. It can be described as follows:

1. Determine if there is a straight line path with no intersection with any obstacle from the starting point S to the destination point D. If such a path exists then exit and report that line as the path, else put the starting point S into the list OPEN (OPEN is initially empty) and make S the root node of the graph GRAPH.
2. If list OPEN is empty exit with failure, else choose and delete from list OPEN the node H with lowest associated cost and make H the current SP (standing point).
3. If SP=D then exit successfully with the path obtained in GRAPH by tracing parent pointers from the destination point D to the starting point S.
4. Expand SP by finding all the visible nodes from SP and make them members of SP's successors set M. Establish pointers from SP to its successors and pointers from each successor back to SP (parent pointers) and put all the successors into GRAPH.
5. Assign heuristic cost $f=g+h$ to each successor nodes. Here, for shortest path, g is the length of the path from the starting point S to the successor node, h is the estimated length from that successor node to the destination point.
6. If some node m in M already exists in GRAPH but does not exist in list OPEN, then delete m from M. Compare the cost of m with the cost of the old one

already existing in GRAPH. If the cost of m is less than the old one, then redirect all its successors to m and recalculate all the associated costs of the successors of m .

7. If some node m in M already exists in GRAPH and also exists in list OPEN then delete m from M . Compare the cost of m with the cost of the old one already existing in OPEN. If the cost of m is less than the old one, then replace the old one in list OPEN with m .
8. Place the remaining nodes of M with their associated costs into list OPEN.
9. Go to step 2.

The visibility graph is constructed in step 4 of the algorithm in which a cost of the form $f=g+h$ is assigned to each node, where g is the known cost from the starting point S to the node and h is the heuristic estimation of the cost from the node to the destination point D . The heuristic estimation, h , reduces the amount of search required on the visibility graph.

3.3.3 Acceleration of Graph Search

The most time consuming step in the proceeding algorithm is that of expanding the node SP in step 4 by generating its successors set M . The search space can be reduced by reducing the successor set. The criterion of expanding SP 's successors is that a node is visible from SP . For shortest path planning, it is not necessary to generate all the successors of SP . One approach to reduce the amount of successors generated in step 4 is the sector approach based on the theory of sectors. The sector approach goes beyond that simple criterion and generates at most two successors for each obstacle in the configuration space with the effect that the visibility graph size is greatly reduced.

A sector is defined in the following manner:

Let $\mathbf{CB} = \{\mathbf{CB}_1, \mathbf{CB}_2, \dots, \mathbf{CB}_n\}$ be the set of polygonal obstacles in the configuration space $\mathbf{C}$.

Let $SP, D \in \mathbf{C}$ be the starting point and destination point respectively, subject to the constraints that SP and D are not elements of any $\mathbf{CB}_i \in \mathbf{CB}$.

Let $\mathbf{CB}_j \in \mathbf{CB}$.

Let B_j be the boundary of $\mathbf{CB}_j$ defined by the ordered set of vertices v_{jk} and their corresponding line segments.

Let $T_j = \{p \in B_j: p \text{ be visible from } SP \text{ with respect to } \mathbf{CB}_j\}$. Note that $p \in T_j$ is not necessarily visible from SP with respect to other obstacles in $\mathbf{CB}$ as shown in Fig. 3.7, $T_j = AB \cup BC \cup HE$, which are thick line segments in Fig. 3.7.

Definition: Let $p_1, p_2 \in T_j$. A sector, SE , is the region defined by the cone of SP over (p_1, p_2) unioned with B_j (see Fig. 3.8).

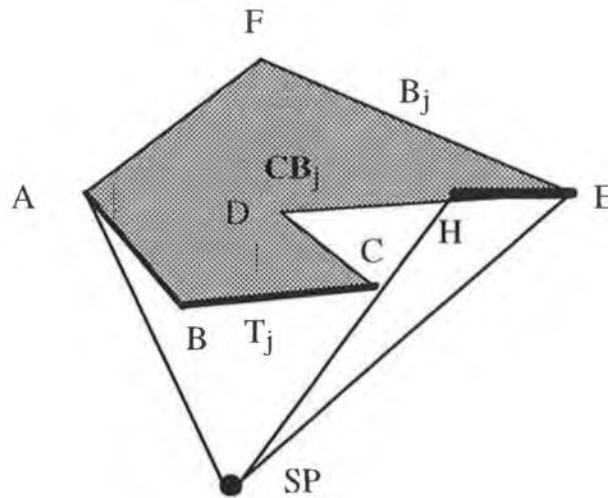


Fig. 3.7: Definition of T_j

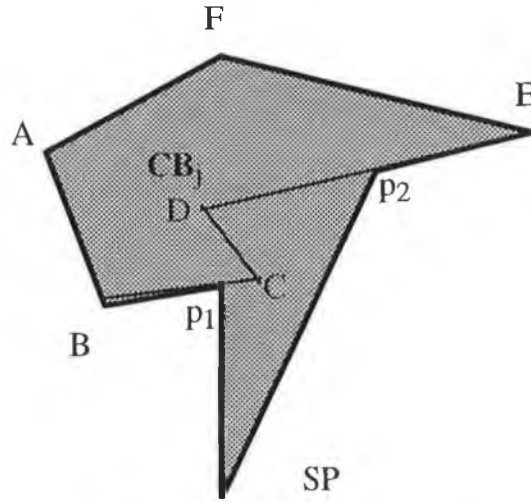


Fig. 3.8: General sector

For the sake of generality, if SE is a sector then the complement of C (here C is the configuration space), $(C-SE)$, is also a sector. For each obstacle CB_j in configuration space, a family of sectors (SE_{ij}) is defined by all the distinct pairs from T_j .

Definition: The primary sector, PS , for an obstacle CB_j is defined as follows:

- If the destination point $D \notin SE$ for all $SE \in (SE_{ij})$ then $PS = SE_{im} \in (SE_{ij})$ subject to SE_{im} has maximum area, as shown in Fig. 3.9.
- If the destination point $D \in SE$ for some $SE \in (SE_{ij})$ then $PS = SE_{im} \in (SE_{ij})$ subject to $D \in SE_{im}$ and SE_{im} minimises the area, as shown in Fig. 3.10.

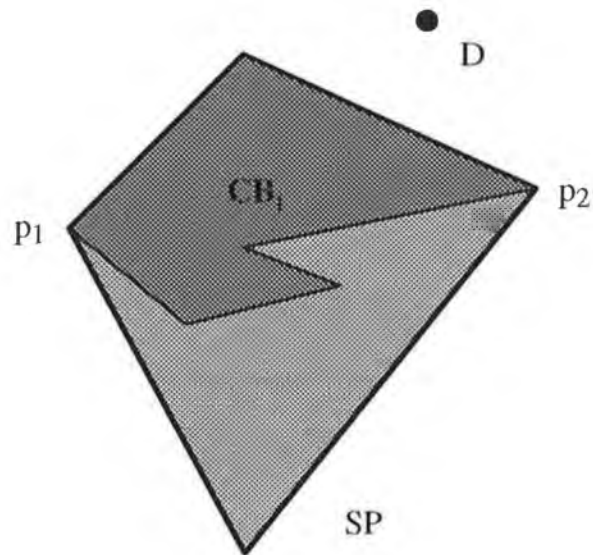


Fig. 3.9: Primary sector with goal outside it

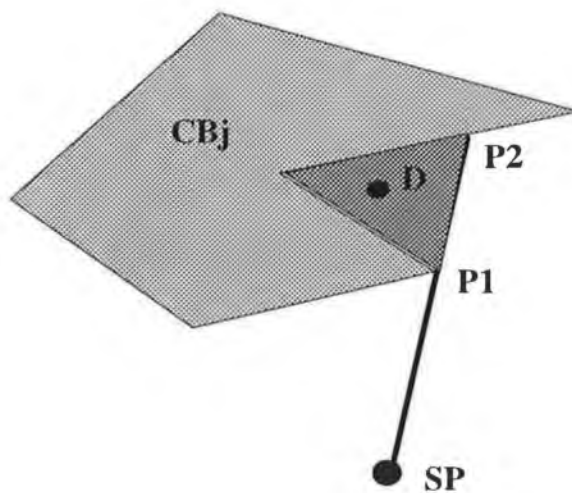


Fig. 3.10: Primary sector with goal inside it

Using the concept of primary sector, a point p in the configuration space is a successor to SP in step 4 of the proceeding algorithm if p satisfies the following conditions:

1. $p \in T_j$ for some CB_j and defines a primary sector.
2. p is a vertex of some CB_j .

3. p is visible from SP with respect to all obstacles CB in the configuration space.

A proof of the fact that a graph constructed using the above concept of successors contains the shortest path is presented in [Nakamura, K. 1982], that is, only part of the whole visibility graph which contains the shortest path is constructed. It must be noted that in the case of a configuration space with polygonal obstacles, the procedure of finding the primary sector is somewhat faster than considering all the visible vertices and expanding them. Fig. 3.11 is one of the simulation results of this algorithm by using a PC.

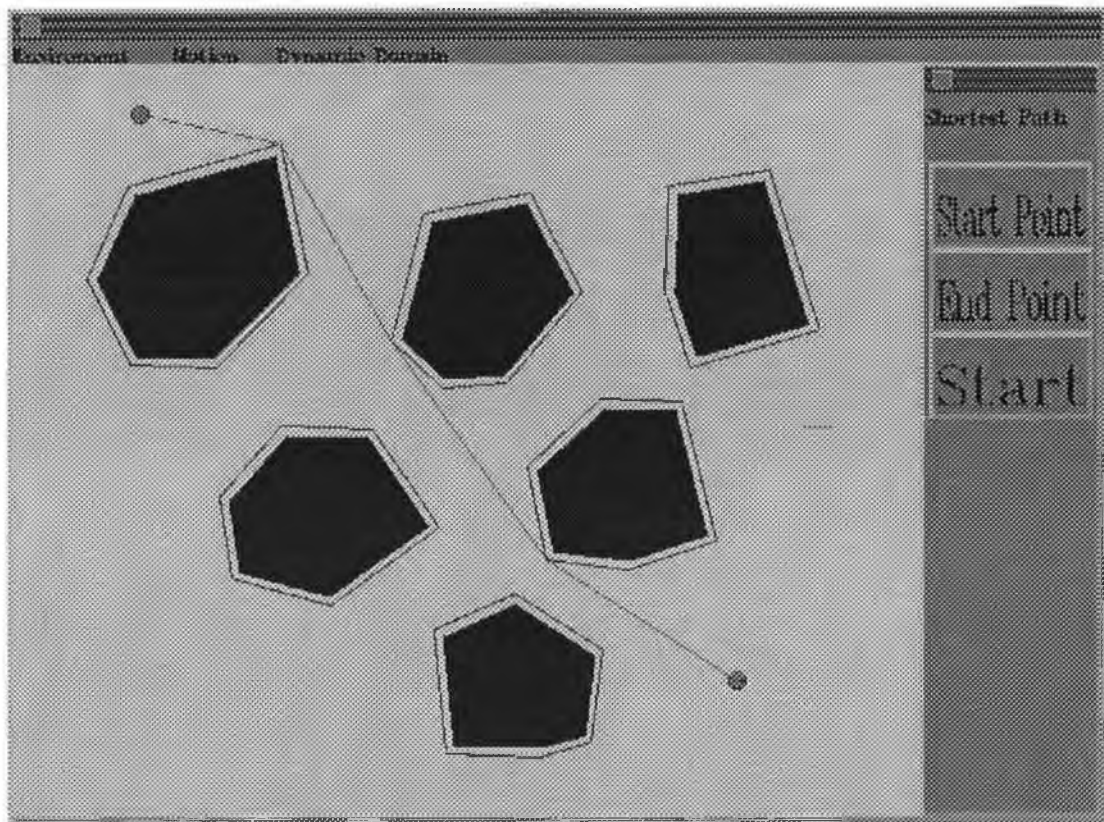


Fig. 3.11: Simulation result of the shortest path

3.3.4 Smoothing the Path

The shortest path planned by the above algorithm consists of the line segments which link the vertices and starting point and destination point. At each vertex of any obstacle, the path is not smooth. For a car-like mobile robot, it is difficult for the robot to follow that path, since the robot must change its orientation smoothly. There are two ways to smooth the path, the first is to plan the path as above and then smooth that path at each vertex. The second is to plan the smoothed path directly and this method will be discussed next.

The concept of tangent graph (also called general visibility graph) plays an important role in directly planning the smoothed path in a two dimensional environment. The polygonal obstacles in the environment are first expanded into general polygonal obstacles as shown in Fig. 3.3. The expanded size R can be selected to be bigger than the real size of the robot to keep enough clearance between the robot and obstacles. The tangent graph is constructed based on the concept of the locally shortest path.

3.3.4.1 Locally Shortest Path

Locally shortest path is the shortest path in a local sense. All locally shortest paths compose a subset of collision-free paths. It is defined based on the concept of the neighbouring region of a curve in a two dimensional configuration space.

Definition: Suppose that a curve C in a two dimensional space D and a small positive real ϵ are given. The region

$$W = \{w: w \in D \text{ and } \text{Distance}(w, c) < \epsilon, \exists \text{ point } c \in C\}$$

is called an ϵ -neighbouring region of curve C , where $\text{Distance}(w, c)$ is a function for computing the Euclidean distance between points w and c .

On the basis of the ϵ -neighbouring region, the locally shortest path is defined as follows:

Definition: A path is called locally shortest if there exists such a small ϵ that it is impossible to find a shorter collision-free path in its ϵ -neighbouring region.

Fig. 3.12 shows the locally shortest path and the shortest path in an environment with one obstacle. Path 1 SAFEDG is the locally shortest path, but path 2 SHMG is not, the shortest path SBCG is also one of the locally shortest paths. Obviously, if no locally shortest path exists, no collision-free path exists in an environment and vice versa.

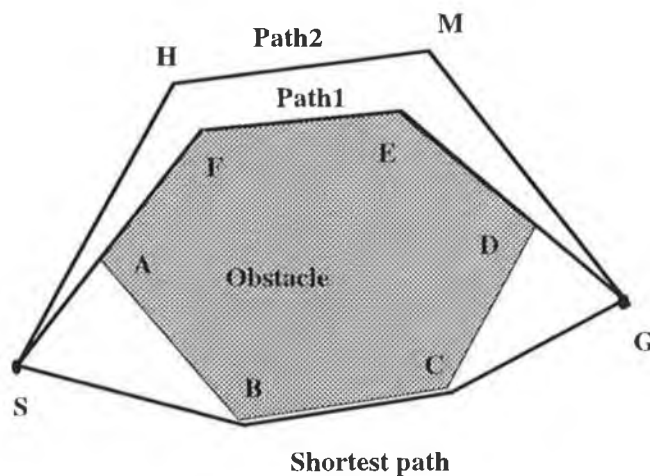


Fig. 3.12: Locally shortest path

3.3.4.2 Tangent Graph

The term “tangent” has close relationship to locally shortest paths. It is defined as:

Definition: If a line contacts a point P on the boundary of a polygonal obstacle but does not intersect any internal point in a small

neighbouring obstacle region of point P , that line is said to be tangent to the obstacle on point P , and point P is called the tangent point. If a line is simultaneously tangent to two different polygonal obstacles, the line is called a common tangent of the obstacles. Furthermore, if small obstacle regions near the two tangent points lie on the same side of the line, the line is called an external common tangent; otherwise it is called an internal common tangent.

The tangent graph has the same structure as the visibility graph. However, its nodes and edges are defined on the basis of locally shortest paths. A node of the tangent graph represents a vertex of a polygonal obstacle on a locally shortest path, and an edge of the tangent graph corresponds to a collision-free line segment, connecting two vertices, that is, on a locally shortest path. The name tangent graph follows from the fact that its nodes and edges are tangent points and tangent segments of polygonal obstacles, respectively. It is obvious that if the line segment connecting two vertices of polygonal obstacles is on a locally shortest path, the line segment (or its extension) must be tangent to the obstacles on the vertices. It is also obvious that the tangent graph is a subset of the visibility graph. Fig. 3.13 is an example of the tangent graph. The thick lines are the edges of the tangent graph and the vertices which the thick lines pass plus the starting and goal points are the nodes of the tangent graph. All the vertices of the polygonal obstacles plus the starting and goal points are the nodes of the visibility graph, and all the thick lines plus the dotted lines in Fig. 3.13 are the edges of the visibility graph.

Obviously, the tangent graph is similar to the visibility graph constructed by using sector theory. An algorithm similar to the one introduced in the above section is used to search the tangent graph to generate the shortest path. The result is the same as the result generated by the visibility graph algorithm.

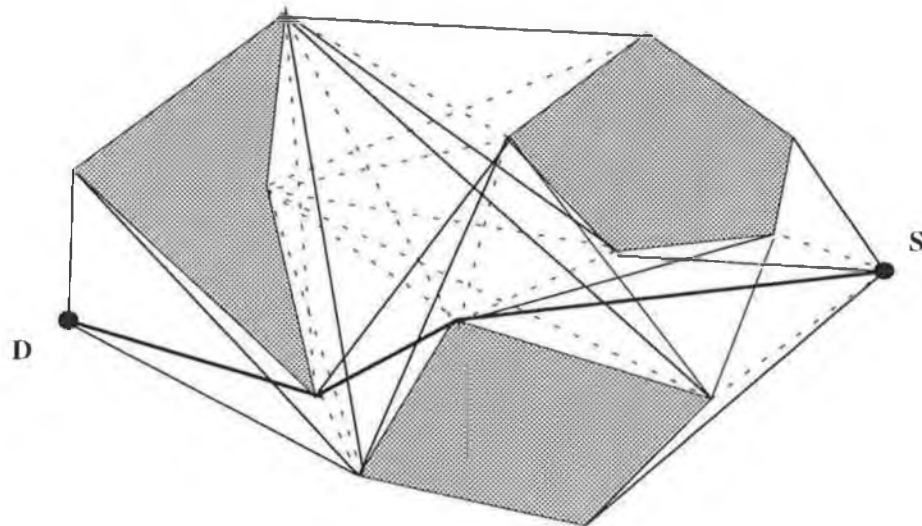


Fig. 3.13: Tangent graph and visibility graph

3.3.4.3 Tangent Graph of Generalised Polygons

There are four common tangent lines between two polygons as shown in Fig. 3.14.

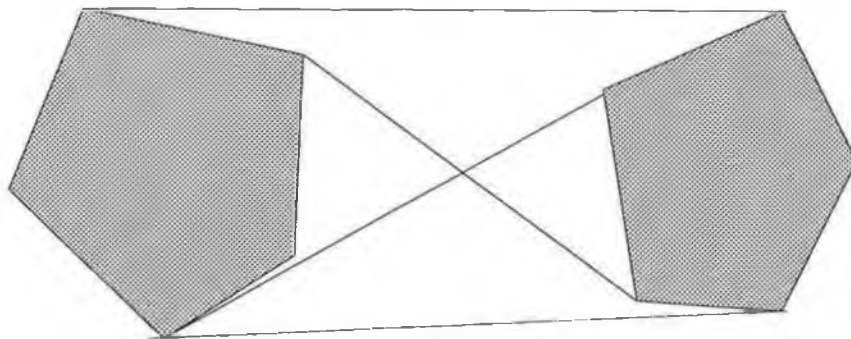


Fig. 3.14: Common tangent lines of two polygons

If the other obstacles are considered in the environment, there are at most four line edges between two obstacles in the tangent graph. The tangent graph can be extended to an environment with generalised polygonal obstacles. A generalised polygonal obstacle is defined as a region which is enclosed by line segments and arc segments as shown in Fig. 3.3 (in fact the edges of a generalised polygon can be any type of curve segments). The common tangent lines between any two generalised

polygonal obstacles, which consist the locally shortest path, are edges of the tangent graph of generalised polygonal obstacles. The nodes of that tangent graph are the tangent points of all the edges on the obstacles. Fig. 3.15 is an example of the tangent graph of generalised polygonal obstacles. Searching the tangent graph of the generalised polygonal obstacles will generate a smoothed shortest path. The thick line segments in Fig. 3.15 show the smoothed shortest path. The searching algorithm can be stated as follows:

1. Expand the polygonal obstacles in the environment into generalised polygons by the size of the robot R.
2. Determine if there is a straight line path with no intersection with any obstacle from the starting point S to the destination point D. If such a path exists then exit and report that line as the path, else put the starting point S into the list OPEN (OPEN is initially empty) and make S the root node of the graph GRAPH.
3. If list OPEN is empty exit with failure, else choose and delete from list OPEN the node H with lowest associated cost and make H the current SP (standing point).
4. If SP=D then exit successfully with the path obtained in GRAPH by tracing the parent pointers from the destination point to the starting point S.
5. Expand SP by finding all the visible tangent points of the generalised polygonal obstacles from the generalised polygon which SP is on and make them members of SP's successors set M. Establish pointers from SP to its successors and pointers from each successors back to SP (parent pointers) and put all the successors into GRAPH.
6. Assign heuristic cost $f=g+h$ to each successor node, where g is the length of the path from the starting point S to the successor node, h is the estimated length from that successor node to the destination point.

7. If some node m in M already exists in **GRAPH** but does not exist in list **OPEN**, then delete m from M . Compare the cost of m with the cost of the old one already existing in **GRAPH**. If the cost of m is less than the old one, then redirect all its successors to m and recalculate all the associated costs of the successors of m .
8. If some node m in M already exists in **GRAPH** and also exists in list **OPEN** then delete m from M . Compare the cost of m with the cost of the old one already existing in **OPEN**. If the cost of m is less than the old one, then replace the old one in list **OPEN** with m .
9. Place the remaining nodes of M with their associated costs into list **OPEN**.
10. Go to step 3.

Fig. 3.16 is one of the simulation results of the smoothed shortest path by using a PC.

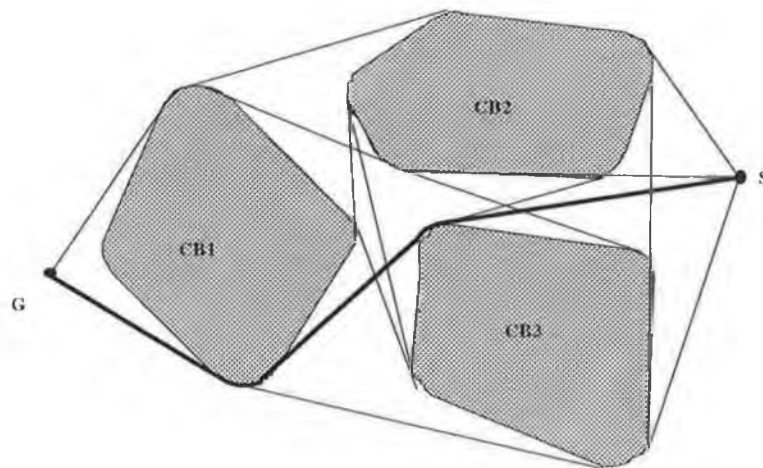


Fig. 3.15: Tangent graph of generalised polygonal obstacles and shortest path

Chapter

4. Minimum Time Path Planning

4.1 Introduction

The shortest path problem has been studied in chapter 3, in which only geometrical factors are considered. An A^* -like algorithm has been proposed for finding the shortest path. But when the inertia and velocity linked factors are taken into consideration, the shortest path is not necessarily the fastest path. In real applications, faster paths are preferable for the operation of mobile robots. This leads to the minimum time path planning of mobile robots.

In this chapter, the minimum time path planning problem will be studied. First, a mathematical model of a mobile robot is established and the minimum time path planning problem is defined based on the mathematical model. An A^* -like algorithm

based on the visibility graph is proposed to search for the minimum time path. In this method, it is assumed that when the robot reaches a corner, the turning speed of a mobile robot and therefore the time for the robot to turn are determined by the angle the robot is about to turn. This is not particularly accurate in a real situation, and it sometimes makes the estimation of the cost for the robot to travel from the present standing point to the destination point very difficult. To make things even worse, finding the velocity at a turning point requires future values of the path. This leads to a depth first search which is very inefficient.

A new approach which solves the minimum time path planning problem in a more practical way is proposed at the end of this chapter. The path of the robot at a turning point is found by smoothing the path with an arc. The robot travels along lines and arcs that make up the path. Along the lines the robot accelerates to its maximum velocity and then decelerates as it approaches an arc to reach the velocity which is decided by the radius of the arc and the clearance between the arc and obstacles, at the tangent point. The robot coasts along the arc with the constant velocity to reach the other tangent point and then accelerate along the other line. This procedure is repeated until the robot reaches the destination point.

Simulation results of these two algorithms by computer are also given.

4.2 Dynamic Planning

Unlike the shortest path planning in which only the geometrical factors are taken into consideration, in minimum time path planning the dynamics related factors of the robot are considered. This will make the problem more complicated, since the planning must take place in the working space (environment) while the dynamics of the robot must be represented in its local coordinates. To simplify the problem, the following assumptions are made.

1. The planning environment is given at the beginning, and the obstacles in the environment are given as polygons. To simplify the problem, it is assumed that all the obstacles are convex polygons. For any concave polygon, it is easy to divide it into a limited number of convex polygons.
2. The mobile robot is assumed to accelerate and decelerate with a specified and constrained rate of speed change, that is, the acceleration and deceleration are a and b respectively, where a and b are constants. The mass of the robot is assumed to be 1 and the real mass of the robot will be reflected in the values of acceleration a and deceleration b .
3. For simplicity, turning (change of orientation) will be considered as pivoting which requires some time to be performed. The turning time is proportional to the turning angle and the mass of the robot.
4. The size of the robot is not taken into consideration, that is, the planning is performed in configuration space.

4.2.1 Mathematical Model of Dynamic Planning

It is assumed above that the robot is a rigid body with negligible dimensions with respect to the environment and obstacles. A single actuator is assumed active at any time, thus only one degree of freedom motion is feasible. Due to physical constraints, the velocity and acceleration are limited by known constants.

The equations of motion for the robot are written in the following form

$$\begin{cases} \frac{dL(t)}{dt} = v(t) \\ m \frac{dv(t)}{dt} = f(t) \end{cases} \quad (4.1)$$

where, $L(t)$ is the position vector of the robot, $v(t)$ is the velocity vector of the robot. The starting position vector of the robot is assumed to be zero and the robot starts moving from zero speed, that is,

$$\begin{cases} L(0) = 0 \\ v(0) = 0 \end{cases} \quad (4.2)$$

Maximum values of speed v_m and driving force f_m are given as

$$\begin{cases} |v(t)| \leq v_m, & \forall t \in R \\ |f(t)| \leq f_m, & \forall t \in R \end{cases} \quad (4.3)$$

Turning is performed as pivoting, and the time of pivoting is proportional to the angle of pivoting, the mass of the robot, and inversely proportional to the speed of motion at the point of pivoting.

$$t_{piv} = k \bullet \theta_{piv} \bullet m \bullet \frac{v_m}{v_t} \quad (4.4)$$

The constraint on turning speed v_t is assumed as the following non-linear function [Meystel, A. 1990]

$$\begin{cases} v_t(\theta) = v_m(a_1\theta^4 + a_2\theta^3 - a_3\theta^2 + 1), & \text{if } 0 \leq \theta \leq \frac{\pi}{2} \\ v_t(\theta) = v_m(a_4\theta + a_5), & \text{if } \frac{\pi}{2} \leq \theta \leq \pi \end{cases} \quad (4.5)$$

An example of the $v_t=Y(\theta)$ is shown in Fig. 4.1 [Meystel, A. 1990]. It is obvious that the robot has to come to a stop when reversing direction and does not have to decelerate if no turn is required (that is, $v_t(0)=v_m$).

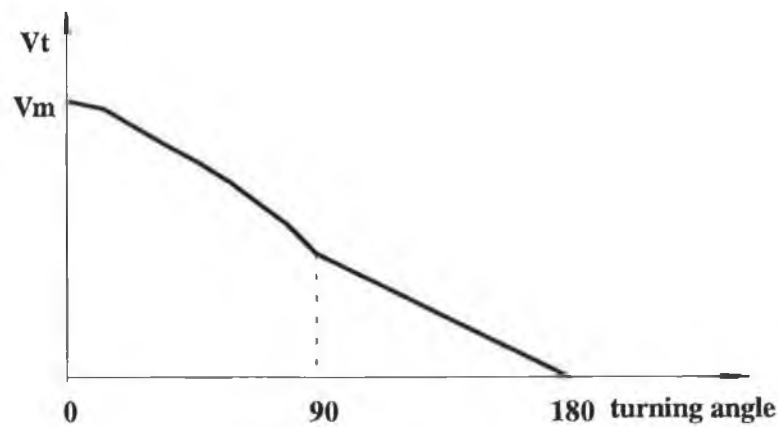


Fig. 4.1: Turning speed v_t Vs turning angle θ

Obstacles are assumed to be polygons O_i , $i=1, 2, \dots, n$ described by a list of their vertices: $v_{ij}=(X_{ij}, Y_{ij})$, $j=1, 2, \dots, N_i$, thus the forbidden region of the robot motion is

$$\sum_{i=1}^n O_i \quad (4.6)$$

which is a subset of the planning environment which can not be considered as a planning alternative by definition.

The general problem is formulated as follows: for the system motion equation (4.1) with initial condition (4.2) and final position assigned at (x_f, y_f) with velocity v_f (usually v_f is zero), find the minimum time trajectory ($t_f=\text{minimum}$) such that the

constraints (4.3), (4.4) and (4.5) are satisfied and the forbidden region (4.6) is avoided.

There are two major difficulties in the solution of the above general problem. The first is that the system dynamics of the robot are represented via local coordinates of the robot whereas the obstacles and destination point are represented in global coordinates. A method of transforming the dynamics of the robot as well as constraints from local into global coordinates is therefore essential.

In [Meystel, A. 1990] [Meystel, A. 1986a] and [Meystel A. 1986b], all the different possible paths in the sense of topology in the planning environment are defined as topways. For an environment with n obstacles, there are 2^n topways. The topways are transformed into a form of slalom situations which are piecewise linear paths. This transformation is considered as an effective method to solve the first difficulty. Although it is not possible to solve the problem in analytical form, but a computation algorithm can be developed for real time applications.

The second difficulty is that the theory of optimal control does not always get an analytical solution for a problem of such arbitrary shape, number and position of obstacles as described by (4.6).

Among the algorithms for finding the optimal solution, the technique of discrete dynamic programming [Larson, R. 1982a] [Larson, R. 1982b] can be considered. Because of the existence of obstacles, the number of admissible states and inputs can be reduced, but the dimensions are still big and the algorithm may not be fast enough. Another approximate algorithm searches all paths along the linear edges of the visibility graph to generate the minimum time path. The solution will be a path along some piecewise line edges which are the line edges of the visibility graph. This is called Minimum Time Path at Visibility Nodes (MTPVN).

4.2.2 Minimum Time Path at Visibility Nodes (MTPVN)

The Minimum Time Path at Visibility Nodes is defined as follows:

Among all the piecewise linear trajectories such that their corners coincide with obstacles vertices, starting at $(x(0), y(0))$ and ending at (x_f, y_f) , find the one with minimal travel time.

The solution to the Minimum Time Path at Visibility Nodes can be obtained by solving a series of subproblems which find the time optimal speed trajectory at a straight line path and then search among the visibility graph for the time minimum path.

4.2.2.1 Optimal Time Solution of a Straight Line Path

For a straight line path which does not intersect with any of the obstacles in the environment, the time optimal speed trajectory can be obtained by using differential calculus to solve the general problem defined above. For simplicity, the following assumptions have been used to construct the model:

1. The robot is point size.
2. Maximum velocity v_m is bounded.
3. Maximum acceleration A_{maxa} and maximum deceleration A_{maxd} are also bounded.
4. All changes in velocity occur at maximum acceleration A_{maxa} or maximum deceleration A_{maxd} .

Assumption (4) has been made to ensure bang-bang control of the robot locomotion and is important for using the differential calculus method. This problem must be divided into two cases: (1) the distance between the starting point and the destination point is not big enough for the robot to accelerate to its maximum speed from the starting speed (usually zero) and then decelerate from its maximum speed to the end speed

(usually also zero), (2) the distance is big enough for the robot to reach its maximum speed and then decelerate to the end speed.

Suppose the starting coordinates and the destination coordinates are (x_s, y_s) and (x_e, y_e) respectively, the starting speed and end speed are v_s and v_e respectively. Calculate the peak speed

$$v_p^2 = \frac{2 \bullet A_{\max a} \bullet A_{\max d} \bullet D + A_{\max d} \bullet v_s^2 + A_{\max a} \bullet v_e^2}{A_{\max a} + A_{\max d}} \quad (4.7)$$

where $A_{\max a}$ and $A_{\max d}$ are the maximum acceleration and maximum deceleration respectively, D is the distance between the starting point and the destination point

$$D = \sqrt{(x_e - x_s)^2 + (y_e - y_s)^2} \quad (4.8)$$

If v_p is bigger than v_m , then the problem belongs to case (2), else it belongs to case (1).

For case (1) the time optimal speed trajectory can be obtained and given as follows

$$v(t) = \begin{cases} v_s + A_{\max a} \bullet t & 0 \leq t \leq t_1 \\ v_p - A_{\max d} \bullet t & t_1 \leq t \leq t_2 \end{cases} \quad (4.9)$$

where

$$t_1 = \frac{v_p - v_s}{A_{\max a}} \quad (4.10)$$

$$t_2 = \frac{v_p - v_s}{A_{\max a}} + \frac{v_p - v_e}{A_{\max d}} \quad (4.11)$$

This case can be illustrated in Fig. 4.2.

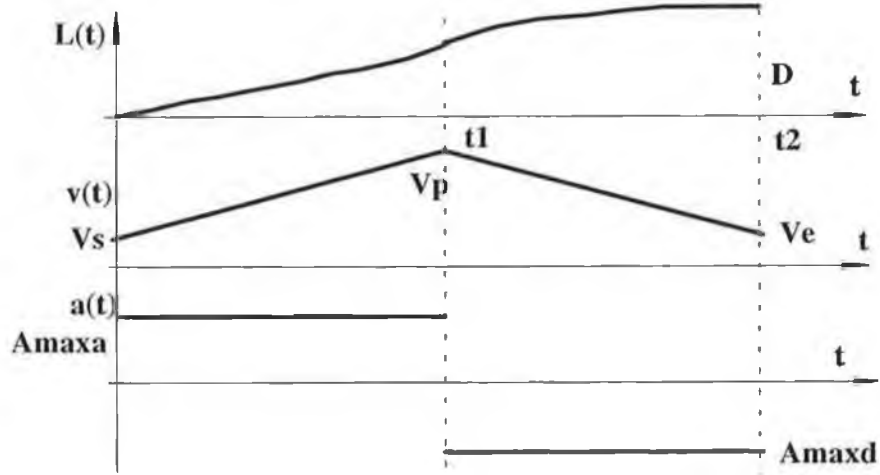


Fig. 4.2: Optimal time trajectory for case (1)

As to case (2), the time optimal speed trajectory can be given by

$$v(t) = \begin{cases} v_s + A_{\max a} \cdot t & 0 \leq t \leq t_1 \\ v_m & t_1 \leq t \leq t_2 \\ v_m - A_{\max d} \cdot t & t_2 \leq t \leq t_3 \end{cases} \quad (4.12)$$

where

$$t_1 = \frac{v_m - v_s}{A_{\max a}} \quad (4.13)$$

$$t_3 = \frac{D}{v_m} + \frac{(v_m - v_s)^2}{2A_{\max a}v_m} + \frac{(v_m - v_e)^2}{2A_{\max d}v_m} \quad (4.14)$$

$$t_2 = t_3 - \frac{v_m - v_e}{A_{\max d}} \quad (4.15)$$

This case is illustrated in Fig. 4.3.

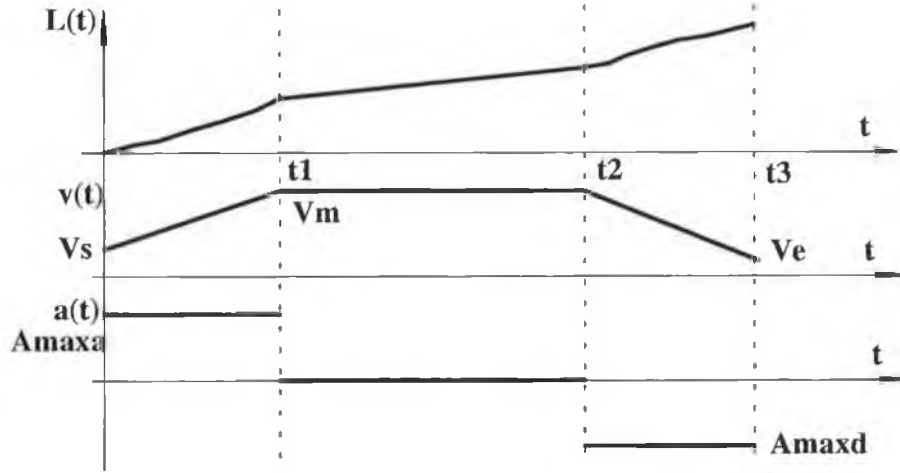


Fig. 4.3: Optimal time trajectory for case (2)

4.2.2.2 Graph Search Algorithm

With the time optimal trajectory for a line path obtained above, a graph search algorithm similar to the shortest path search algorithm can be used to search among the visibility graph to obtain the minimum time path. As shown in Fig. 4.4, the minimum time for the robot to move from point A to point B can be obtained by adding the minimum time of line path AC and line path CB and the time for the robot to turn at corner C, which can be calculated by (4.4), that is,

$$t = t(AB) + t(BC) + t(\theta) \quad (4.16)$$

It should be noted that in (4.16) and Fig. 4.4, the time for the robot to move along any path with piecewise linear segments is related to the speed profile along the line path and the angle which the robot is about to turn at that vertex. The speed profile

along a line path is dependent on the speed at the starting point of the line, the speed at the end point of the line and the length of that line. A knowledge base which is represented by production rules can be constructed based on the results obtained above. The resulting knowledge base has 14 production rules which are shown in Fig. 4.5.

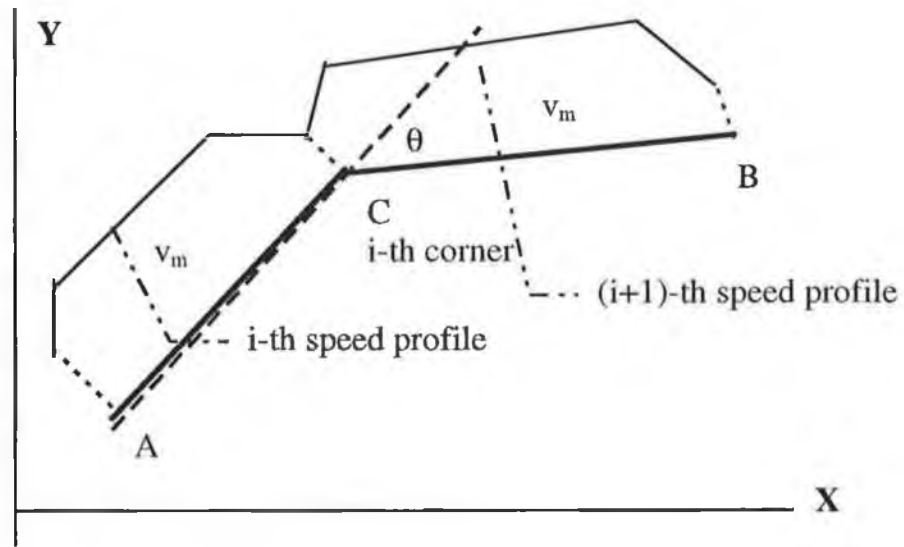
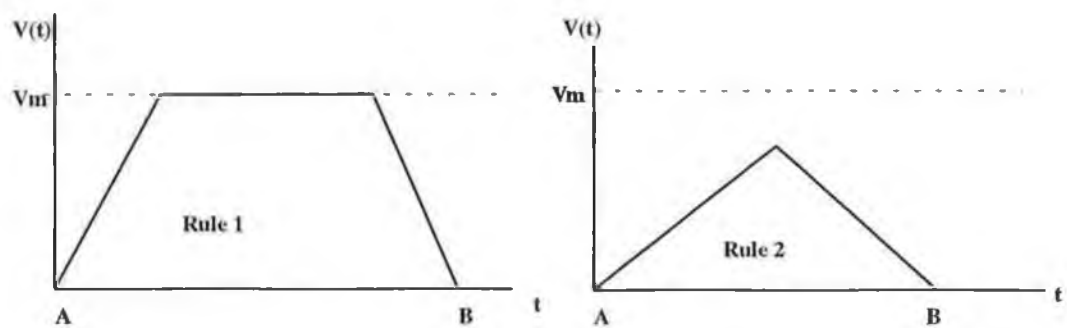
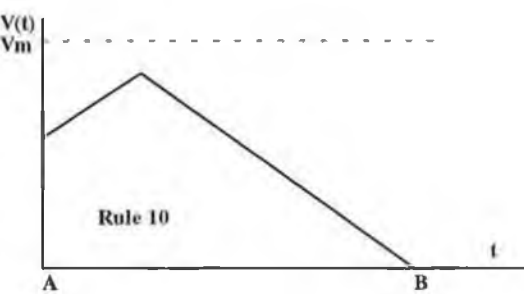
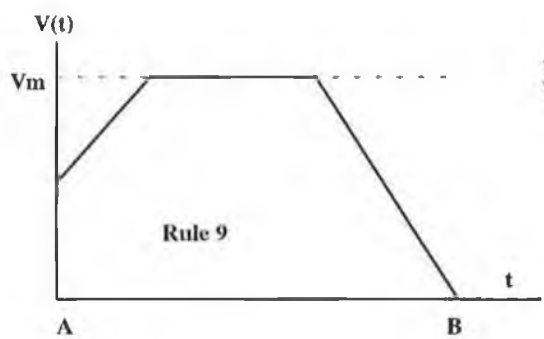
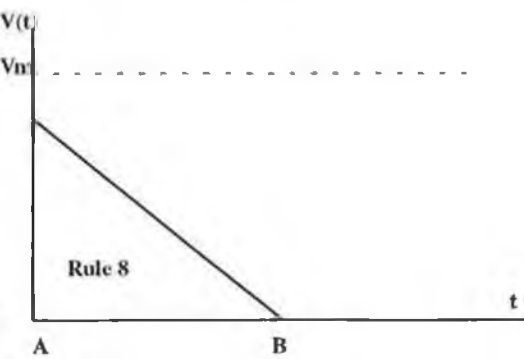
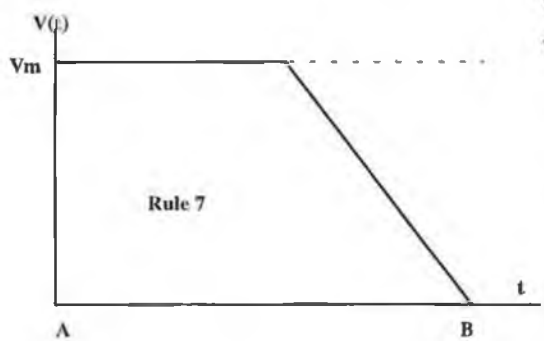
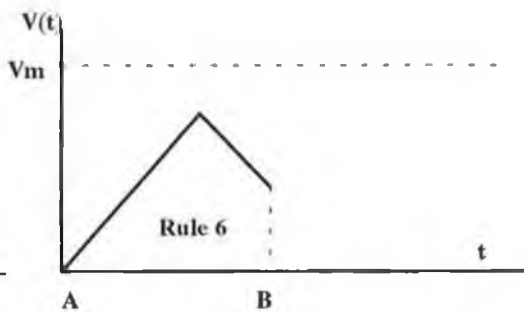
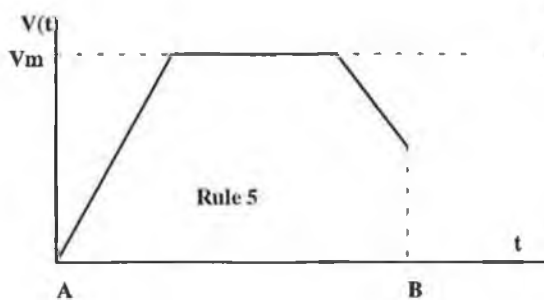
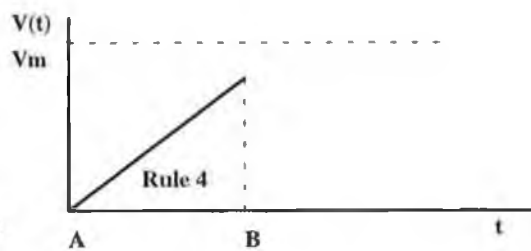
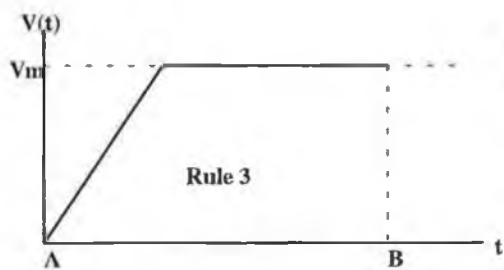


Fig. 4.4: Illustration of piecewise linear time optimal trajectories





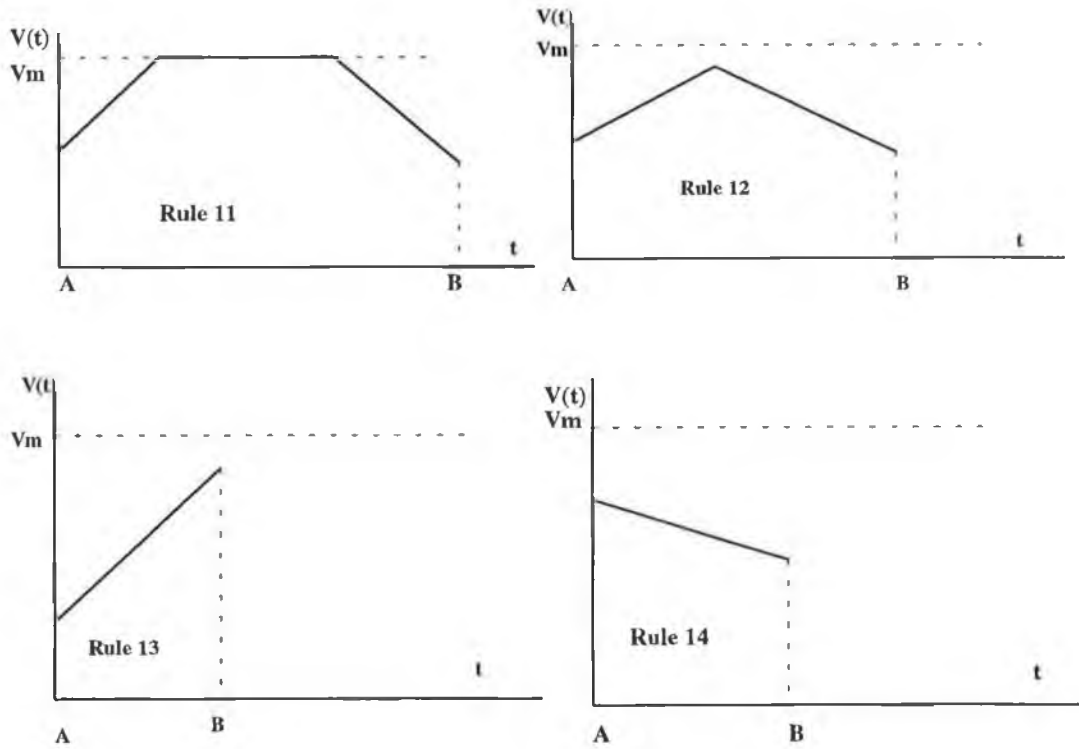


Fig. 4.5: Illustration of the production rules for speed trajectory

Rules 1 and 2 denote the speed trajectories between the line path which links the starting point and the destination and does not intersect with any of the obstacles. The difference between them is that in rule 1 the length of the line path is big enough for the robot to reach its maximum speed, whereas in rule 2 the distance of the line path is not big enough for the robot to reach its maximum speed. Rules 3, 4, 5 and 6 denote the situation where the velocity at point A is zero, such as in the case of point A being the starting point of the robot, and the velocity at point B is not zero. Rules 7, 8, 9 and 10 denote the situation where the velocity at point B is zero, such as in the case of point B being the destination point of the robot, and the velocity at point A is not zero. In rules 11, 12, 13 and 14, neither one of the velocities at points A and B is zero, and these rules are obviously the ones applied at non-terminal segments of paths.

The above rules summarise all possible speed trajectories between any two vertices in the visibility graph and a combinations of them can be used to construct the speed trajectory of a piecewise linear path of any length between starting point and

destination point. A graph search algorithm similar to the one used in chapter 3 to find the shortest path can be used to search among the visibility graph with the cost function being time. However, these rules are constructed under the assumption that the velocity at point B is known when point B is found to be a successor of point A. Unfortunately, this is not the case. To make things even worse, not only is the velocity at point B unknown when point A is expanded, but the problem is complicated by the fact that the velocity of the robot at point B may have several different values, since the turning velocity at B is dependent on the turning angle to B's successor and there may be many such successors. The recursive nature of this problem leads to the construction of the complete visibility graph while performing a depth first search on the same graph which is inefficient and unacceptable for real applications. To solve this problem, the concept of a point of invariance (POI) of a successor m whose parent is node n is introduced. Suppose the linear path connecting n with m with distance D_e is denoted by e, the point of invariance of successor m is defined below.

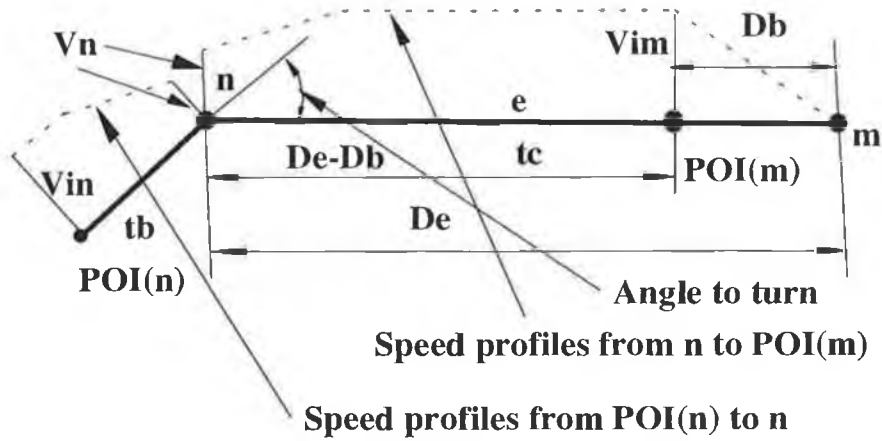


Fig. 4.6: Illustration of the point of invariance of a node

The point of invariance, P, of successor m, is on the linear path e and at a distance D_b from successor m, such that $D_b = (v_i^2) / (2A_{maxd})$, where v_i is the speed at P given the time optimal speed trajectory from node n with starting speed v_s to successor m with end speed $v_e = 0$, and v_i is dependent upon distance D_e between node n and its successor m. This is shown in Fig. 4.6, where the point of invariance of node m

which is a successor of node n , $POI(m)$, is a point on the line segment nm and is D_b away from m , v_{in} and v_{im} are the speeds of the robot at the points of invariance of n and m respectively. The speed profiles along the line segment nm show that the distance D_b is long enough for the robot to decelerate from speed v_{im} at $POI(m)$ to zero speed at m .

By definition, the point of invariance is the point on e at which application of maximum deceleration of A_{maxd} while at speed v_i will bring the robot to a complete stop exactly at m . This approach has been chosen so that all turns possible at m can be accounted for in the time cost function evaluation when knowledge of those turns are produced. In fact, the coordinates of the point of invariance of a node do not need to be stored explicitly in the graph, only the braking distance D_b and the velocity v_i are stored. Therefore, the point of invariance is not an actual node in the graph, but just a delimiter of time cost function evaluation from a node n to its successor m , that is, the time cost of m stored in the graph is not the actual time for the robot to reach m , but the time cost for the robot to reach the point of invariance of m at speed v_i . The cost of edge e can be calculated by the following equation

$$t_e = t_b + t_c + t_{turn} \quad (4.17)$$

where t_b is the time cost of getting from the point of invariance of node n while at velocity v_{in} to node n itself with a turn velocity v_t given by the function of the turn angle in (4.5), t_c is the time cost of getting from node n with an initial velocity v_t to the point of invariance of node m with a final velocity v_{im} (computed by definition), and t_{turn} is the time for the robot to turn from line $POI(n)-n$ to line $n-POI(m)$. This is shown in Fig. 4.6. Time item t_{turn} is decided by (4.4). The algorithms for computing t_b and t_c can be obtained by using the results of time optimal trajectory of a linear path and will be introduced below.

The time cost t_b can be calculated by the following manner.

1. If $v_t=0$ then $t_b = v_{in}/A_{maxd}$.

2. If $v_t = v_m$ the problem must be analysed as follows:

If the condition $v_{in} \geq \frac{v_m}{\sqrt{2}}$ is satisfied, the maximum velocity v_m can be reached, that is, the distance D_b is long enough for the robot to accelerate from v_t to v_m , calculate t_b by $t_b = \frac{v_{in}^2}{2A_{\max d} v_m} + \frac{(v_m - v_{in})^2}{2A_{\max a} v_m}$, else the maximum speed v_m can not be reached, that is, the speed at n can not be v_m , the maximum speed at n can only be $\sqrt{2} \cdot v_{in}$, so assign $\sqrt{2} \cdot v_{in}$ to v_t and calculate t_b by $t_b = \frac{v_t - v_{in}}{A_{\max a}}$. This analysis applies to any v_t at n , which is bigger than v_{in} .

3. Compute the peak velocity as follows:

$$v_p^2 = \frac{2A_{\max a} A_{\max d} D_b + A_{\max d} v_{in}^2 + A_{\max a} v_t^2}{A_{\max a} + A_{\max d}}$$

where $D_b = \frac{v_{in}^2}{2A_{\max d}}$ is the braking distance. So v_p can be calculated as:

$$v_p^2 = v_{in}^2 + \frac{A_{\max a} v_t^2}{A_{\max a} + A_{\max d}}$$

4. Compare v_p with v_m . If v_p is greater than v_m then calculate t_b by:

$$t_b = \frac{D_b}{v_m} + \frac{(v_m - v_{in})^2}{2A_{\max a} v_m} + \frac{(v_m - v_t)^2}{2A_{\max d} v_m}$$

else calculate t_b by $t_b = \frac{v_p - v_{in}}{A_{\max a}} + \frac{v_p - v_t}{A_{\max d}}$.

Time t_c can be calculated as follows:

1. Compute the peak velocity v_p by:

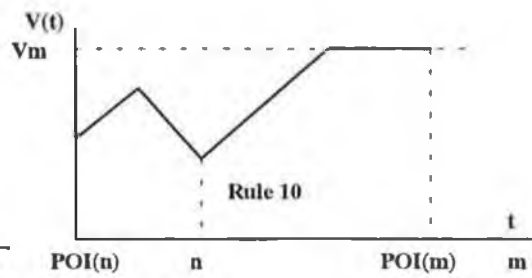
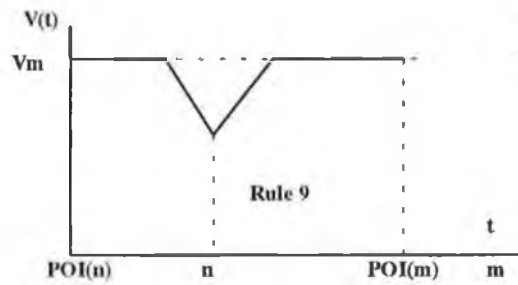
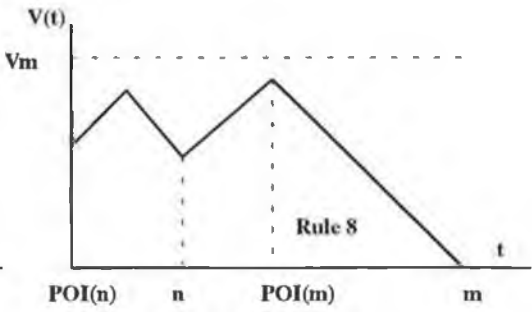
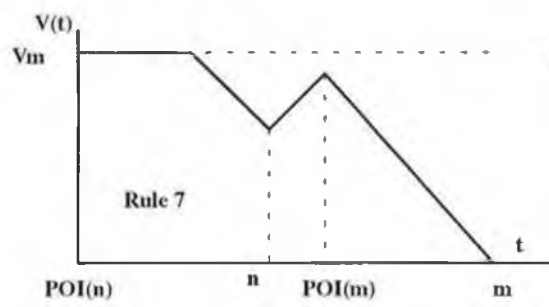
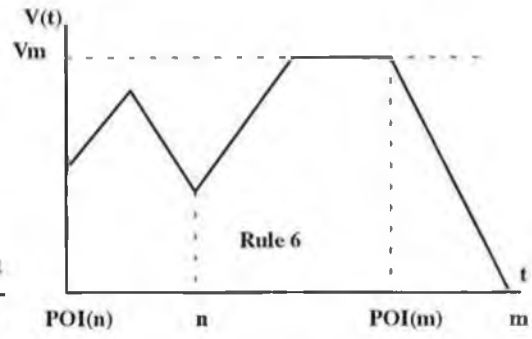
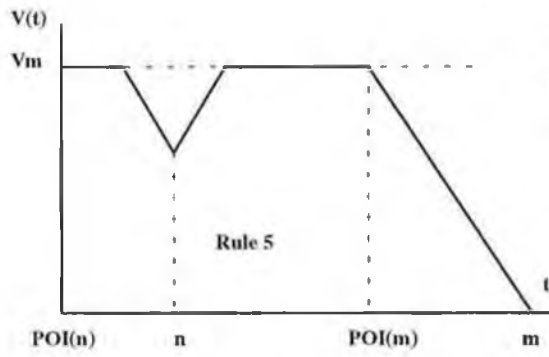
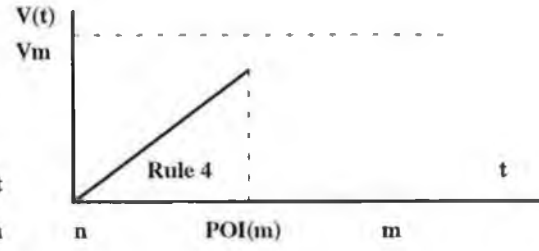
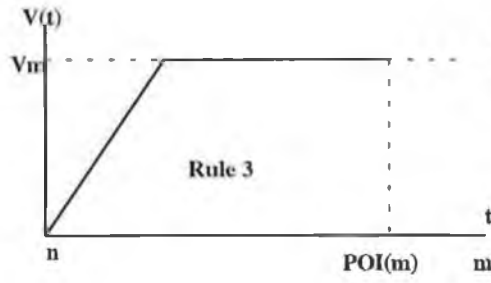
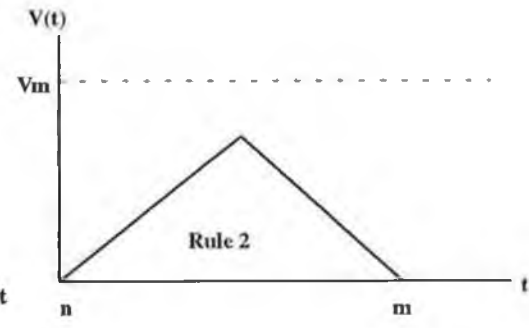
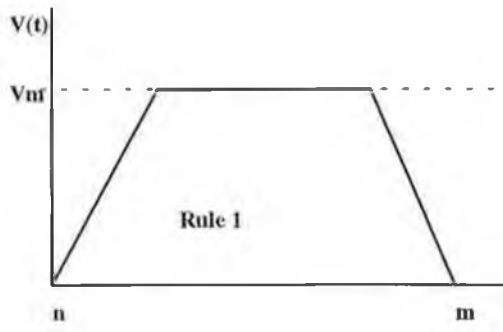
$$v_p^2 = \frac{2A_{\max a}A_{\max d}De + A_{\max d}v_t^2}{A_{\max a} + A_{\max d}}$$

2. Compare v_p to v_m . If v_p is greater than v_m then $v_{im}=v_m$ and calculate t_c by

$$t_c = \frac{De}{v_m} + \frac{(v_m - v_t)^2}{2A_{\max a}v_m} - \frac{v_m}{2A_{\max d}}$$

else $v_{im}=v_p$ and calculate t_c by $t_c = \frac{v_p - v_t}{A_{\max a}}$.

The method of solving the depth first search inefficiency is to evaluate the speed trajectory along the linear path between the standing point n and its successor m 's point of invariance $POI(m)$. The speed at $POI(m)$ can be easily calculated by the approach to the computation of t_c introduced above. Now instead of evaluating the speed trajectory from standing point n to its successor m , the speed trajectory from the point of invariance of n , $POI(n)$, to the point of invariance of m , $POI(m)$ will be evaluated. To simplify the problem, the segment between $POI(n)$ and $POI(m)$ is divided into two parts: braking from $POI(n)$ to n and coasting from n to $POI(m)$. The production rules for evaluation speed trajectories between points of invariance are reconstructed and shown in Fig. 4.7. Rules 1 and 2 are the same as rules in Fig. 4.5. Rules 3 and 4 are the only two possibilities for point n 's velocity being zero while there are four rules for this case in Fig. 4.5. Rules 4, 5, 6 and 7 are analogous to rules 7, 8, 9 and 10 in Fig. 4.5, except that evaluation now begins at $POI(n)$. Rules 9, 10, 11 and 12 are analogous to rules 11, 12, 13 and 14 in Fig. 4.5, except evaluation in Fig. 4.7 begins and ends at points of invariance $POI(n)$ and $POI(m)$ respectively.



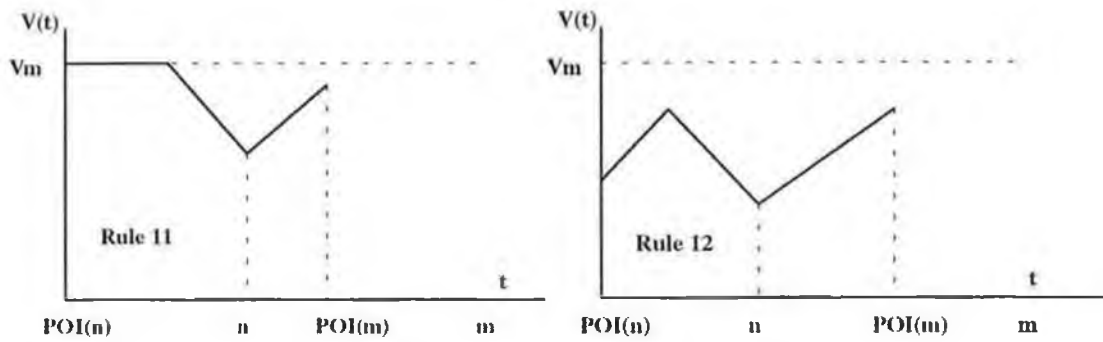


Fig. 4.7: Illustration of production rules with respect to POI

These 12 rules can be described as follows:

If((n=starting point) and (m=destination point))

{

 If($v_p \geq v_m$) then Rule 1, else Rule 2.

}

Else if(n=starting point)

{

 If($v_p \geq v_m$) the Rule 3, else Rule 4.

}

Else if(m=destination point)

{

 If($v_p(\text{POI}(n), n) \geq v_m$)

{

 If($v_p(n, \text{POI}(m)) \geq v_m$) then Rule 5, else Rule 7.

}

 Else

{

 If($v_p(n, \text{POI}(m)) \geq v_m$) then Rule 6, else Rule 8.

}

}

Else

```

{
  If( $v_p(\text{POI}(n), n) \geq v_m$ )
  {
    If( $v_p(n, \text{POI}(m)) \geq v_m$ ) then Rule 9, else Rule 11.
  }
  Else
  {
    If( $v_p(n, \text{POI}(m)) \geq v_m$ ) then Rule 10, else Rule 12.
  }
}

```

Based on the definition of point of invariance and the construction of the production rules, the graph search algorithm which can generate the minimum time path among the visibility graph can be stated as follows.

1. Determine if there is a straight line path with no intersection with any obstacle from the starting point S to the destination point D. If such a path exists, then use rule 1 or rule 2 to generate the speed trajectory along that line path and exit and report the line as the path. Else put the starting point S into list OPEN (OPEN is initially empty) and make S the root node of the graph GRAPH.
2. If list OPEN is empty exit with failure, else choose and delete from list OPEN the node H with lowest associated cost and make H the current SP (standing point).
3. If SP=D then exit successfully with the path obtained in GRAPH by tracing parent pointers from the destination point D to the starting point S.
4. Expand SP by finding all the visible nodes from SP and make them as members of SP's successors set M. Establish pointers from SP to its successors and pointers from each successor back to SP (parent pointers) and put all the successors into GRAPH.

5. Evaluate the time optimal speed trajectory for each line path from SP to each of its successors by using the production rules in Fig. 4.6. Calculate the time for the robot to reach the point of invariance of each successor node of SP and assign it to g ; estimate the time for the robot to reach the destination point from the point of invariance of each successor and assign it to h . Associate the time cost $f=g+h$ and the speed of the robot at the point of invariance of the successor to each of SP's successor.
6. If some node m in M already exists in GRAPH but does not exist in list OPEN, then delete m from M . Compare the cost of m with the cost of the old one already existing in OPEN. If the cost of m is less than the old one's cost, then redirect all its successors to m and recalculate the associated costs of the successors of m .
7. If some node m in M already exists in GRAPH and also exists in list OPEN then delete m from M . Compare the cost of m with the cost of the old one already existing in OPEN. If the cost of m is less than the old one's cost, then replace the old one in list OPEN with m .
8. Place the remaining nodes of M with their associated costs into list OPEN.
9. Go to step 2.

Fig. 4.8 is one of the simulation results of the above algorithm by PC, Fig. 4.9 is the velocity profile of the robot along the path shown in Fig. 4.8.

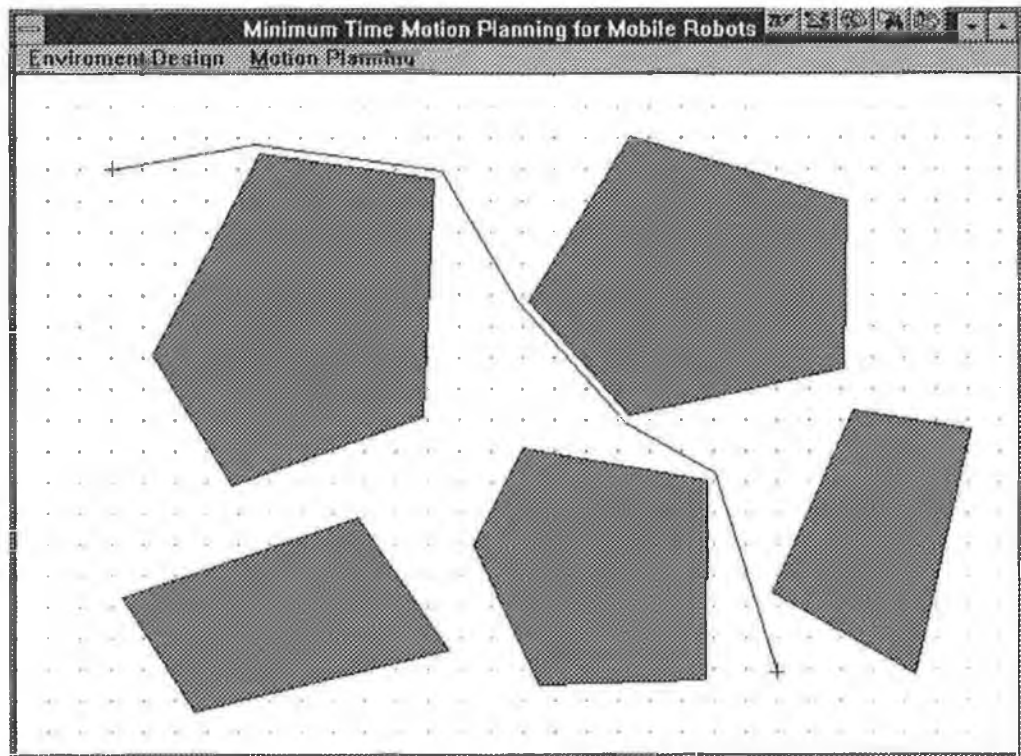


Fig. 4.8: Minimum time path result

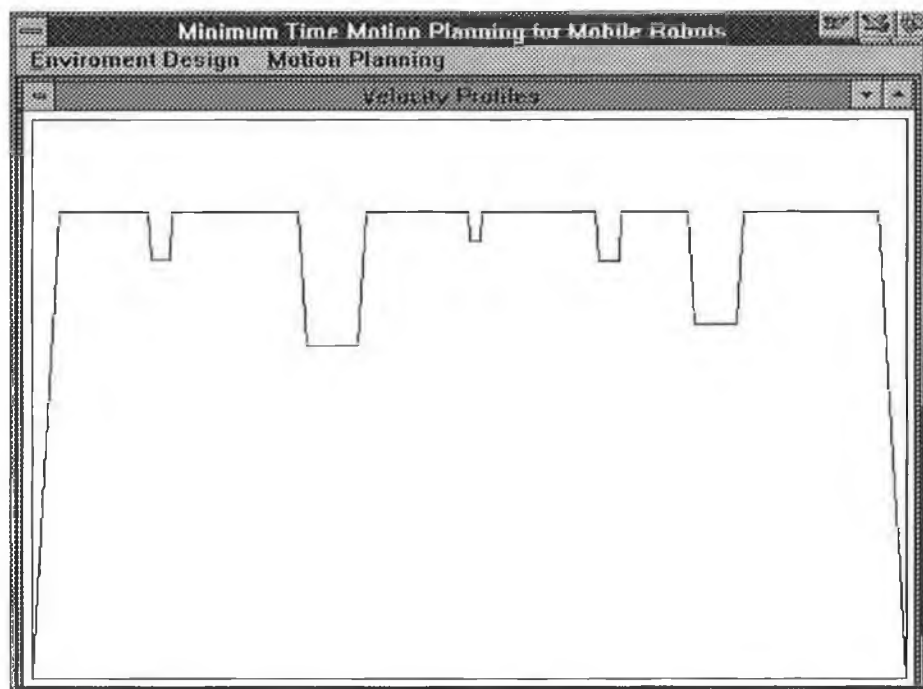


Fig. 4.9: Velocity profiles of the minimum time path

4.3 Smoothed Minimum Time Path

The above minimum time path generating algorithm is effective and practical. However, deciding the velocity at a turning point and the time for the robot to turn its orientation around that corner is sometimes difficult and impractical. For a car like mobile robot, the robot must turn in a smoothed curve. There are several methods for path smoothing [Liu, Y. 1992] [Fraichard, Th. 1991] [Hayashi, A. 1991] [Liu, Y. 1991], the simplest one is to use an arc to smooth the path around a corner. From this point of view, the minimum time path of the robot can be considered by smoothing the path at a turning point as shown in Fig. 4.10.

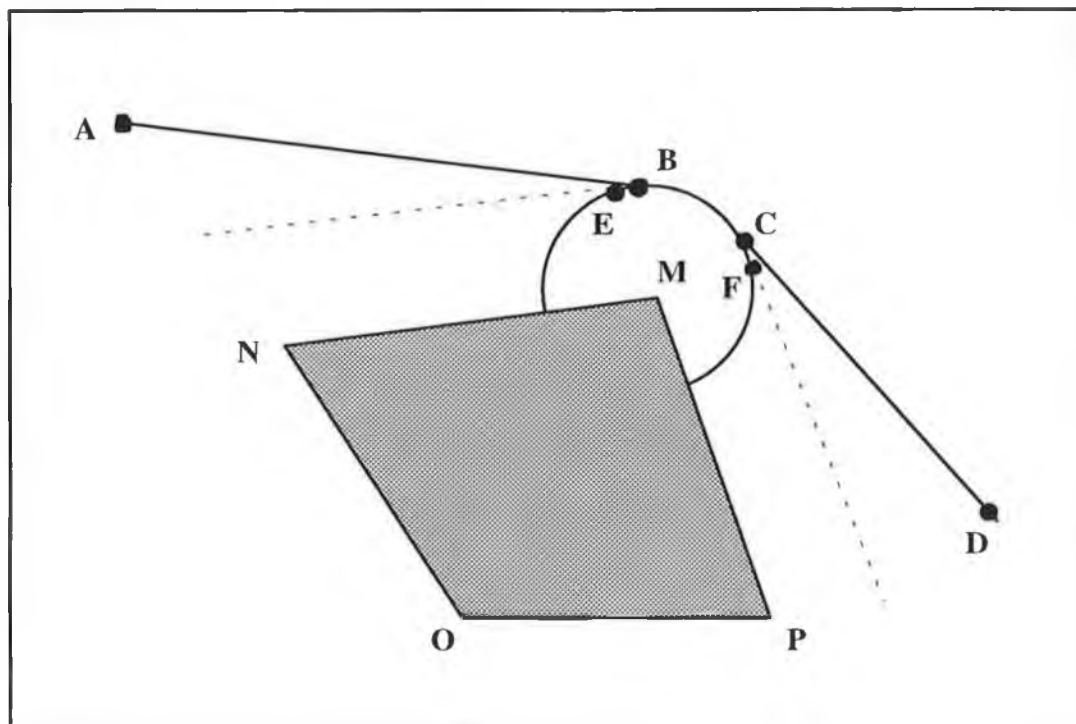


Fig. 4.10: Smoothing the path at a corner

When the robot is about to pass the vertex M of polygon MNOP from point A to point D, the robot moves along a line path AB which is tangent to a circle, reaches the tangent point B at a certain velocity which is constrained by the curvature of the circle, then along an arc path BC at that velocity, and finally along another line path CD which is also tangent to that circle. The radius of the circle is chosen so that the

robot can travel as fast as possible, while the velocity for the robot to move along an arc path is determined by the radius of the arc and the potential field function values along the arc path (that is, the clearance between the arc path and the obstacles, since the potential field function is the measurement of the clearance), the bigger the radius, the bigger, the velocity. So the smoothed minimum time path consists of piecewise linear segments and arc segments. The time optimal speed trajectory along an linear path can be evaluated by using the production rules shown in Fig. 4.5. The time for the robot to coast along an arc path is equal to the length of the arc divided by the speed along the arc path.

Another factor which influences the velocity of the robot along the arc is the distance between the robot and the obstacles, the potential field function value is the measure of the clearance between the robot and the obstacles, so using it to determine the velocity of the robot is reasonable. There are several approaches to define the potential field function [Hwang, Y. 1992] [Latombe, J. 1990]. Here the approach in [Hwang, Y. 1992] is used. Let $\{ g(x, y) \leq 0, g \in L^m, x \in R^n, y \in R^n \}$ be the set of inequalities describing a convex polygon region, where L denotes the set of linear functions. Then the scalar function

$$f(x, y) = \sum_{i=1}^N (g_i(x, y) + |g_i(x, y)|) \quad (4.18)$$

is zero inside the polygon region and grows linearly as the distance from the region increases, where N is the number of bound segments of the polygon. The potential field function can be defined as

$$p(x, y) = \frac{1}{\delta + f(x, y)} = \frac{1}{\delta + \sum_{i=1}^N (g_i(x, y) + |g_i(x, y)|)} \quad (4.19)$$

where δ is a small positive constant. It is obvious that the potential field function inside the polygon region is δ^{-1} and decreases in inverse proportion to distance outside the polygon region. If there is more than one polygonal obstacle in the planning environment, the potential field function value at any point is the maximum value of all the potential field function values due to all the polygons at that point. It is obvious that the bigger the potential field function value at a point, the less the distance between that point and the obstacles, so the less the velocity of the robot.

The potential field function at a point can represent the distance between that point and the obstacles. However, there are an infinite number of points in an arc and it is impossible to evaluate all the potential field function values for the arc. The integration of the potential field function along the arc is used to judge the clearance between the arc and the obstacles instead. There is a problem with the integration range, because some part of the circle must be inside the interior of the obstacle, so integration along the whole circle is meaningless. Here as shown in Fig. 4.10, the tangent point E of the circle with the line which is parallel to MN is selected as the starting point for the integration, and the tangent point F of the circle with the line which is parallel to MP is selected as the ending point for the integration, that is, the integration of the potential field function will be carried out along arc EF. It can be denoted as in (4.20).

$$p = \int_E^F p(s) ds \quad (4.20)$$

This selection seems to expand the polygonal obstacle into a generalised polygon in configuration space by the size of the radius of the circle. And the selection is reasonable, since if the common tangent lines between this generalised polygon and other generalised polygon are drawn, the tangent points on this generalised polygon must lie on the arc selected for the integration.

With p defined in (4.20) as the measure for the clearance between an arc and the obstacles in the environment, the velocity for the robot to coast along the arc path can be given by

$$\begin{cases} \left(\frac{r}{R_{\max}}\right)^n \bullet \frac{v_m}{p} & \text{if } r \leq R_{\max} \\ \frac{v_m}{p} & \text{if } r \geq R_{\max} \end{cases} \quad (4.21)$$

where n is an integer which is bigger than 1, $R_{\max}$ is the radius of the arc along which the robot can move with its maximum velocity and v_m is the maximum velocity of the robot. n is used to emphasise the importance of the radius, the bigger n is, the more important the radius is in relation to the velocity along an arc.

Fuzzy logic is effective for systems which can not easily be expressed in analytical form. It can also be used to generate the velocity for the robot to move along an arc path. Table 4.1 shows the most simple inference rules to generate the velocity along an arc.

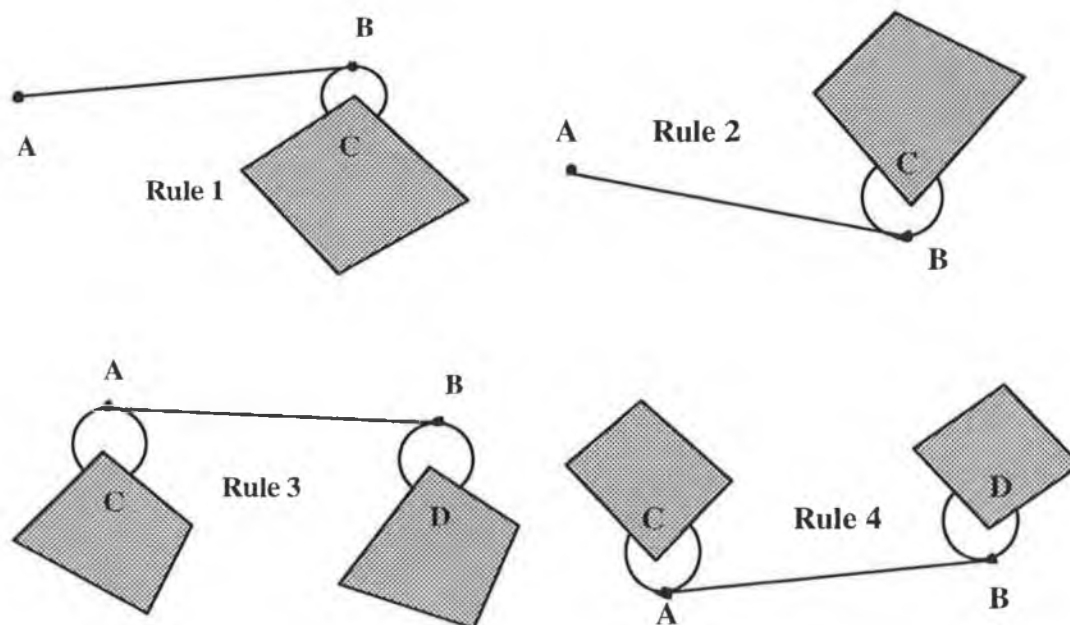
	Potential Function Value			
Radius		Big	Medium	Small
	Big	Small	Medium	Big
	Medium	Small	Small	Medium
	Small	Small	Small	Small

Table 4.1: Fuzzy inference rules

It is easy to use fuzzy logic to generate the velocity when the radius and the potential value of the arc are known, but it is difficult to solve the inverse problem. Given a

vertex the robot is about to pass, it is difficult to find the radius of the arc around that vertex along which the robot can travel with the fastest velocity by fuzzy logic. So the analytical form (4.21) is usually used to solve such a problem.

With the radius of the arc generated, the next step is to find the tangent point from the current standing point. There are two cases: the first is that the current standing point is the starting point or the successor of the current standing point is the destination point, in this case, the tangent line of a circle from a point (the starting point or the destination point) is to be found and so is the tangent point. The second case needs to find the common tangent points of two circles and the common tangent line which consists the time optimal path. Production rules for generating the tangent points and tangent line are also constructed as shown in Fig. 4.11. There are two tangent lines from a point to a circle, rule 1 and rule 2 denote the selection of which line as the path segment with respect to the position of the obstacle. There are four common tangent lines for two circles, rule 3, rule 4, rule 5 and rule 6 denote the selection of which common tangent line as the path segment.



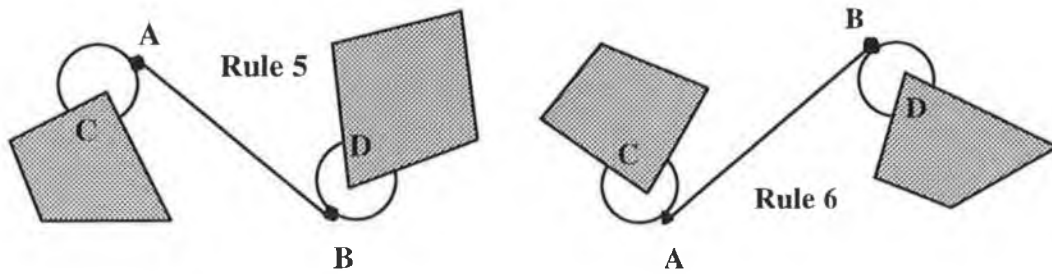


Fig. 4.11: Production rules for finding the tangent points and lines

The graph search algorithm which can generate the smoothed minimum time path can be stated as follows.

1. Determine if there is a straight line path with no intersection with any obstacle from the starting point S to the destination point D. If such a path exists, then use rule 1 or rule 2 of Fig. 4.5 to generate the speed trajectory along that line path and exit and report the line as the path. Else put the starting point S into list OPEN (OPEN is initially empty) and make S the root node of the graph GRAPH.
2. If list OPEN is empty exit with failure, else choose and delete from list OPEN the node H with lowest associated cost and make H the current SP (standing point).
3. If SP=D then exit successfully with the path obtained in GRAPH by tracing parent pointers from the destination point D to the starting point S.
4. Expand SP as follows: (1) calculate the fastest radius for every vertex of the obstacles; (2) find the tangent points and tangent line from SP to each circle by using the production rules in Fig. 4.11 and see if the tangent line intersects with any of the obstacles (the same meaning as visible). (3) if the tangent line does not intersect with any of the obstacles, make that vertex a member of SP's successors set M. Establish pointers from SP to its successors and pointers from

each successor back to SP (parent pointers) and put all the successors into GRAPH.

5. Evaluate the time optimal speed trajectory for each line path from SP to each of its successors by using the production rules in Fig. 4.5. Calculate the time for the robot to reach the tangent point of each successor node's circle and assign it to g ; estimate the time for the robot to reach the destination point from the tangent point of that circle and assign it to h . Associate the time cost $f=g+h$, the velocity for the robot to travel around the circle and the radius of the circle of the successor to each of SP's successor.
6. If some node m in M already exists in GRAPH but does not exist in list OPEN, then delete m from M . Compare the cost of m with the cost of the old one already existing in OPEN. If the cost of m is less than the old one's cost, then redirect all its successors to m and recalculate the associated costs of the successors of m .
7. If some node m in M already exists in GRAPH and also exists in list OPEN then delete m from M . Compare the cost of m with the cost of the old one already existing in OPEN. If the cost of m is less than the old one's cost, then replace the old one in list OPEN with m .
8. Place the remaining nodes of M with their associated costs into list OPEN.
9. Go to step 2.

Fig. 4.12 is one of the simulation results by using the above algorithm and Fig. 4.13 is the velocity profiles along the path given in Fig. 4.12.

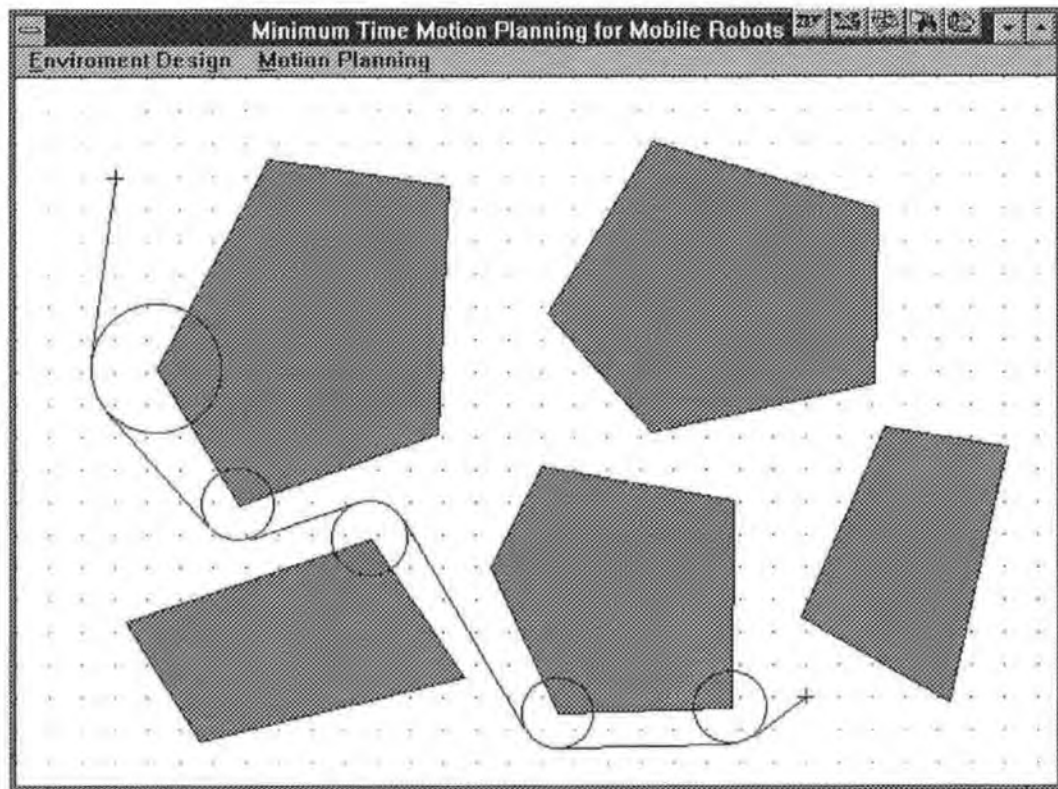


Fig. 4.12: Smoothed minimum time path result

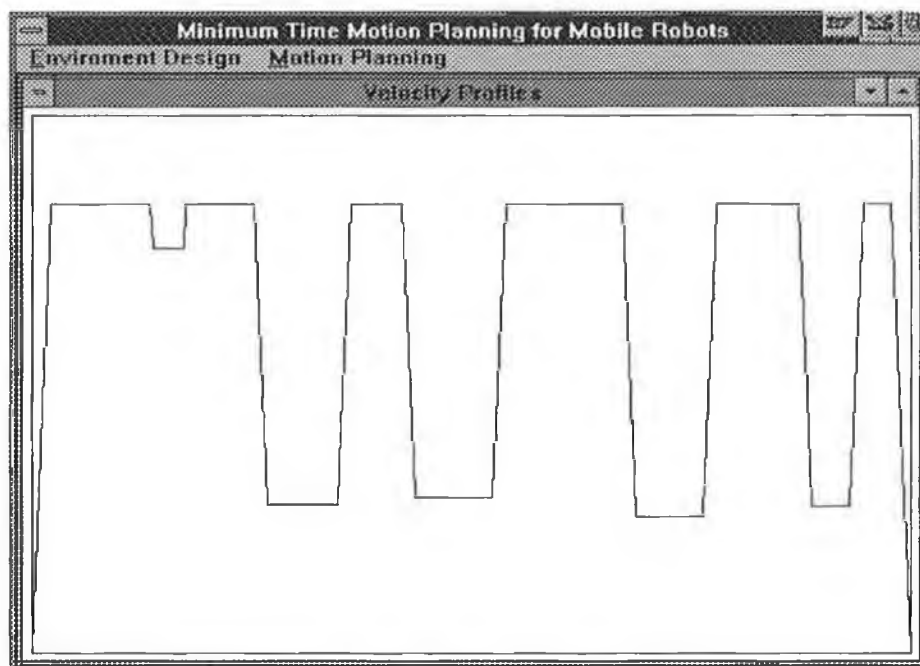


Fig. 4.13: Velocity profiles along the path shown in Fig. 4.12

4.4 Summary and Discussion

Minimum time path planning has been studied in this chapter. Two planning approaches have been used to generate different minimum time paths. The first approach is based on the visibility graph, if the acceleration and deceleration of the robot are infinite, that is, the robot is massless, the minimum time path in this particular case will be the shortest path. Compare the simulation results in Fig. 4.8 and Fig. 4.12, the first approach tries to find the minimum time path which is shorter and less turning total angle, while the second approach tries to find the safer path (along which the robot has bigger clearance with the obstacles so it can travel faster). In Fig. 4.11, when determining the velocity for the robot to coast along an arc, n is assigned 3 in (4.21) and $R_{\max}$ is 50, so the turning velocity is emphasised. That is why the velocity profile in Fig. 4.12 changes so sharply. Different n and $R_{\max}$ in (4.21) will give different velocity profiles. It is difficult to say which approach is better. It depends which aspect is emphasised. These two approaches give alternatives for minimum time path planning for mobile robots and the second approach seems to be more practical for car-like mobile robots.

Another aspect of these two approaches is that in the first approach, the concept of point of invariance of a successor is needed to avoid the complete construction of the visibility graph and the depth first search in the same graph, while in the second approach, there is no such need, because when the radius of the circle around an vertex is given, the speed for the robot to reach the tangent point of that circle is known. But finding the radius of a circle around a vertex and its tangent points are very time consuming. An alternative to the integration in (4.20) is to discretize the point along arc EF in Fig. 9 and sum up all the values of the potential field function at each discretized point as p or just find the biggest value among all the values of the discretized points as p .

In the first approach, special attention must be paid to the cost evaluation of h (the remaining cost for the robot to travel from the selected successor to the destination point). As can be seen in Fig. 4.14, normal time evaluation that includes the turn from

successor A to the destination point D may result in a higher time value than the true time needed to reach the destination point D from A through B, since a combination longer path AB and BD with small turns of real turn angle 1 and real turn angle 2 will be faster than a shorter path AD with a sharp turn, heuristic turn angle. To solve this problem, an artificial destination point C is positioned at a location such that the distance stays constant but the turn angle is zero as shown in Fig. 4.14, that is, the cost of the turning will not be taken into consideration when evaluating h. A similar way can also be used to cope with the similar situation in the second approach.

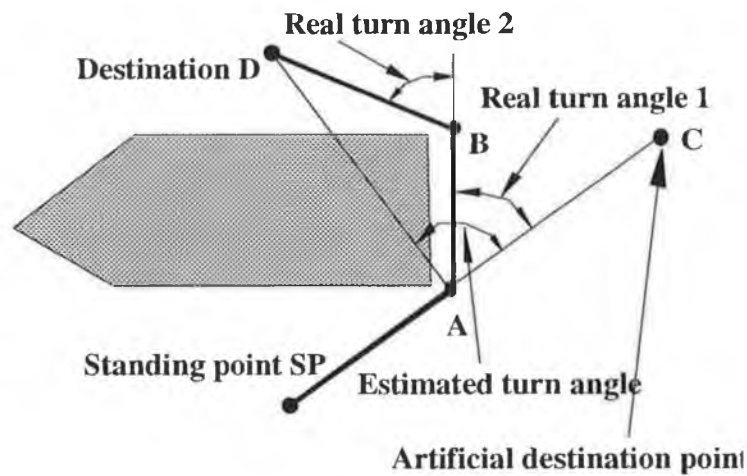


Fig. 4.14: Real cost is lower than the heuristic cost

It should be pointed out again that there is no analytical solution to the minimum time path planning as represented by (4.1) to (4.5). The real minimum time path may not pass the vertices of the obstacles. It may not pass the graph edges constructed in the smoothed minimum time path planning approach either. The minimum time paths generated by these two approaches are only near minimum time paths, or approximate minimum time paths. But they are effective for practical applications.

Chapter

5. Motion Planning in Dynamic Environments

5.1 Introduction

Shortest path planning and minimum time path planning in two dimensional static environments have been studied in chapter 3 and chapter 4. It has been seen that the visibility graph and configuration space play a very important role in planning shortest path and minimum time path among stationary obstacles in a two dimensional world. In this chapter, motion planning problems for a mobile robot among moving obstacles in a two dimensional world will be studied.

In chapter 3 and chapter 4 where the motion planning is performed in static environments, a path is specified as a subset in the plane and the term trajectory is usually used to refer to velocity and acceleration information along a path. In a dynamic environment, a collision free path for a particular time period may not be collision free in another time period. Therefore, a path must also be specified as a

function of time. In this chapter, the term “motion” is used to mean a path with timing information, while the term path is used to specify a curve in space without timing information.

The concept of accessibility which is a generalisation of the concept of visibility will be studied first. Making use of accessibility, a graph called the accessibility graph is defined to represent moving objects for the purpose of planning the motion of a mobile robot. The robot is assumed to be a point that moves in a two dimensional environment in which polygonal obstacles, as well as the destination point, are in motion. The accessibility graph is shown to be a generalisation of the visibility graph in the sense that paths to the destination point are found as sequences of edges of the graph. In fact when all the obstacles are moving with zero velocity, the accessibility graph becomes the visibility graph of these polygonal obstacles. More importantly, if the robot is able to move faster than any of the obstacles, then the graph exhibits a property: a time minimum motion is represented as a sequence of edges of the accessibility graph. An algorithm based on the concept of accessibility for generating the minimum time path among moving obstacles will also be given in the first part of this chapter.

The accessibility graph algorithm can only generate a minimum time motion for a massless robot (that is, the robot can accelerate from zero speed to its maximum speed in no time) among moving obstacles. For robots with mass (that is, it takes time for the robot to accelerate from zero speed to its maximum speed), the approach usually used is the space time (or state time) approach in which another dimension time t is added to represent the motion of the obstacles and the robot (see chapter 2 for more details), that is, the two dimensional problem is transformed into a three dimensional problem. This will make the problem more complicated and require more memory and time to solve. In the second part of this chapter, the concept of the accessibility polygon of a moving polygonal obstacle is first introduced. Based on the concept of the accessibility polygon, the concept of improved accessibility will be studied and the improved accessibility graph is used to generate minimum time path among moving obstacles for a robot with mass. The computation time for the

improved accessibility graph algorithm is almost the same as the accessibility graph algorithm which is in $O(n^2 \cdot \log(n))$, where n is the total number of vertices of the polygonal obstacles in the environment [Fujimura, K. 1988] [Fujimura, K. 1992]. This is a great improvement to the space time algorithm in which the memory and the computation time required are of exponential order. This algorithm can also be extended to solve motion planning problems in three dimensional dynamic environments where the space time algorithm will become a four dimensional problem which is difficult to represent in geometrical space and difficult to solve. This extension will be studied in chapter 6.

Some special cases of motion planning in two dimensional dynamic environments will also be studied in the last part of this chapter. First, the application of sector theory which is studied in chapter 3 to accelerate the search among the accessibility graph or improved accessibility graph is studied. With the application of the theory of sectors, the computation time for the accessibility graph algorithm is $O(n \cdot k \cdot \log(k))$, where k is the total number of polygonal obstacles in the environment and n is the total number of all the vertices of those polygonal obstacles. Then the accessibility graph algorithm and the improved accessibility graph algorithm are extended to solve motion planning problems in dynamic environments with special types of obstacles such as circular obstacles, growing and shrinking obstacles, obstacles with piecewise linear motion, obstacles with accelerating movements, obstacles in rotating trajectories, and transient obstacles etc.

5.2 Accessibility Graph

5.2.1 Definition

Through out this chapter, S , D and R are used to represent the starting point, the destination point and the point robot, respectively, unless there is special specification

for other representations. First the motion of the obstacles, the point robot and the destination point are defined.

The obstacles in the planning environment are considered as polygons. To make the problem simple, the polygons are treated as convex polygons. For a concave polygon, it can be divided into a limited number of convex polygons [Meystel, A. 1990].

Motion of an Obstacle: As mentioned above any obstacle in the environment is a convex polygon. The motion of an obstacle is defined as a polygon that moves in a fixed direction at a constant speed. Such a straight motion is called a **movement**. To simplify the problem, the motion of a polygon is treated as the motion of a set of lines (edges) which constitute the polygon. A **movement** is defined as the motion of a line and is expressed as a tuple (L, d_L, v_L) that represents the motion of a line segment L in direction d_L at speed v_L . So the obstacles in the environment are defined as a set of limit movements $M = \{ M_{11}, M_{12}, \dots, M_{1n_1}, M_{21}, M_{22}, \dots, M_{2n_2}, \dots, M_{k1}, M_{k2}, \dots, M_{kn_k} \}$, where k is the total polygon number and n_i ($i=1, 2, \dots, k$) is the number of lines (edges) of the polygons, in other words there are $n = n_1 + n_2 + \dots + n_k$ movements in the environment.

The motion planning problem can be considered in configuration space. To simplify the problem, the robot is considered as a point and the obstacles are enlarged by half the maximum size of the robot.

Motion of a Point Robot: After leaving the starting point at the starting time, the point robot R can have any motion as long as its speed does not exceed a given maximum speed and as long as it does not pass through the interior of any obstacle. The maximum speed of R is assumed to be greater than that of any of the obstacles and that of the destination point (if the destination point is also moving).

Motion of a Destination Point: The motion of the destination point consists of a finite series of steps. Within a step, the destination point moves in a constant direction at a constant speed. The number of steps is assumed to be bounded.

Based on the above definitions, the concepts of meeting point, accessibility, collision front, accessibility vertex set and accessibility graph can be defined as follows.

Meeting Point: Consider a possibly moving point P (P is either the destination point or a point on a moving polygonal obstacle). Let R be a point initially located at starting point S at starting time t_0 . Suppose that R starts moving at time t_0 at a speed v . After R starts moving, it moves in a fixed direction. It is said that P and R meet if there exists a location X through which both P and R pass at the same time t , where $t > t_0$. The location X is defined as the meeting point of P with respect to v and S . The time t is defined as the meeting time of the meeting point X with respect to P and is denoted by $t(X)$.

It is obvious that when R moves faster than P , the meeting point of P is uniquely determined. However, when the speed of R is slower than that of P , it is possible that R is never able to meet P . The meeting point of a possibly moving point P depends on the speed and the initial location of R .

Accessibility: A point robot R is planning to move from starting point S to goal point D in an environment with a set of obstacle movements $M=\{M_1, M_2, \dots, M_n\}$. Suppose R starts moving at time t_0 at a speed v . After R starts moving, it will move in a fixed direction. A point P (P is either the destination point or a point on any polygonal obstacle) is said to be accessible from starting point S (with respect to t_0 and v), if R meets P without any prior interception by any other movement. The meeting point and meeting time are defined as the accessible point and accessible time of P respectively. Obviously if P is a stationary point and is accessible then its accessible point is P itself.

Collision Front: Consider an environment that contains one movement (L, d_L, v_L) . Let A and B be the two endpoints of L and let m_A and m_B be the accessible points of A and B with respect to R's initial point S, starting time t_0 and speed v respectively. The set of accessible points corresponding to all points in L forms a curve segment and this curve segment is called the collision front of L with respect to starting point S, starting time t_0 and speed v . Fig. 5.1 shows an example of the collision front. The starting point of the robot is assumed to be the origin of the co-ordinates and L is a line segment which is parallel to the Y axis with length Y_L , the equations of the collision front are:

$$\begin{cases} t(A) = \frac{\sqrt{x^2 + y^2}}{V_{\max}} = \frac{\sqrt{(x - x_0)^2 + (y - a)^2}}{v_L} \\ y = (x - x_0) \tan \theta + a \\ y_0 \leq a \leq y_0 + Y_L \end{cases} \quad (5.1)$$

where, $t(A)$ is the accessible time and θ is the angle between the X axis and the velocity vector. The points (x, y) which satisfy (5.1) form the collision front of line BA. The equations in (5.1) define a quadratic relationship, that is, the collision front lies either on a parabola, hyperbola, or ellipse. The curve is a hyperbola when $v_{\max} > v_L \cos(\theta)$, parabola when $v_{\max} = v_L \cos(\theta)$, and ellipse when $v_{\max} < v_L \cos(\theta)$, respectively. The collision front degenerates to a line segment when the direction of motion d_L is parallel to L or when the initial position of L is collinear with the starting point S. So the collision front of a movement takes the form of either a straight line segment or a conic curve segment.

For an environment which contains more than one movement, there can be more than one collision front, each of which corresponds to a movement. In this case, however, it is possible that only part of a collision front is accessible. It is obvious that if two movements do not collide, then the corresponding collision fronts do not intersect.

The characterisation of the collision front is as follows: Suppose that the robot R departs S at time t_0 and keeps moving at a constant speed v along a ray r emanating from S . The point robot R does not meet the line L if and only if r does not intersect with the collision front of L . The collision front of a stationary edge L is L itself or consists of subsets of L .

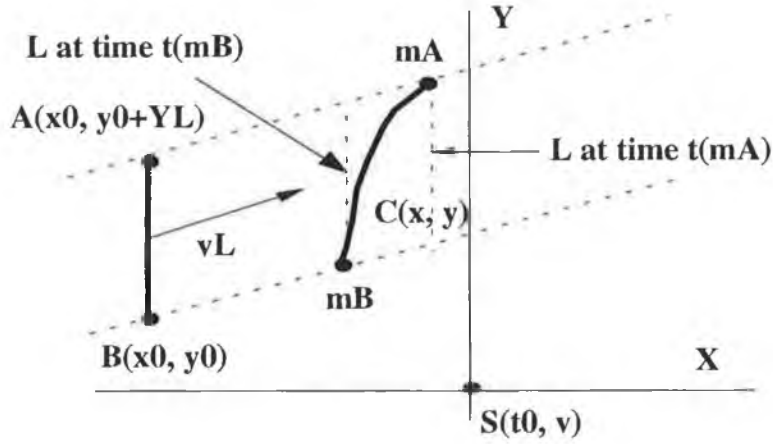


Fig. 5.1: Collision front of a movement

Accessible Vertex Set: Let VS be the set consisting of the destination point and all the vertices of the polygonal obstacles $M=\{M_1, M_2, \dots, M_n\}$ in the environment. The set of accessible points corresponding to elements of VS with respect to starting point S , starting time t_0 and speed v is defined as the accessible vertex set and denoted as $AVS(M, S, t_0, v)$. Since some vertices in VS may not be accessible from S , the size of AVS is at most the size of VS .

Accessibility Graph: Let S be the starting point, t_0 be the starting time, v be the speed of the mobile robot and D be the destination point. With each accessible point X corresponding to point V , an item of time $t(X)$ is associated with X to denote X 's accessible time with respect to V . The **accessibility graph** is defined as follows and denoted as $AG(M, S, t_0, v)$:

1. Insert S in the vertex set of the accessibility graph AG and set its default accessible time to t_0 .
2. For every newly added vertex U in the set of vertices in AG , calculate the accessible vertex set with U as the initial point, i.e. $AVS(M, U, t(U), v)$. Insert the elements of this AVS in the vertex set of AG and the edges from U to these vertices points in the edges of AG .

The accessibility graph is a directed tree rooted at S . For a given set of polygonal obstacles $M=\{M_1, M_2, \dots M_n\}$, starting point S and destination point D , the accessibility graph AG varies for different values of t_0 and v . The accessibility graph is infinite according to the definition. However it can be made finite by adopting the rule that when a vertex appears more than once, then the instance with the smallest accessible time is kept. Under this rule, the number of vertices in the accessibility graph does not exceed the number of vertices in VS which is defined above. It should be pointed out that the vertices in the accessibility graph are the accessible points, not the original vertices of the obstacles.

The accessibility graph is related to the visibility graph in the following manner. For an environment with a stationary set of polygonal obstacles $M=\{M_1, M_2, \dots M_n\}$ which do not overlap with each other, construct the infinite accessibility graph of M , and make all directed edges in the accessibility graph undirected and remove all the accessible time items which are associated with all the vertices. The resulting graph is the visibility graph of M .

5.2.2 Minimum Time Motion Planning Algorithm

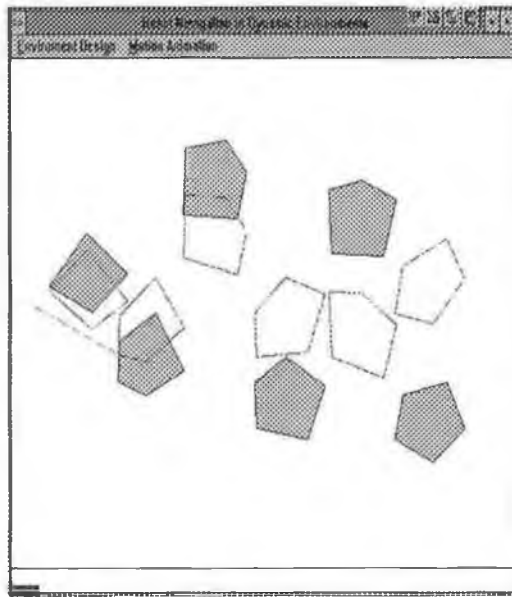
Based on the accessibility graph, the following algorithm can generate a minimum time motion for a mobile robot among moving polygonal obstacles.

1. Insert the starting vertex S into the queue $OPEN$.

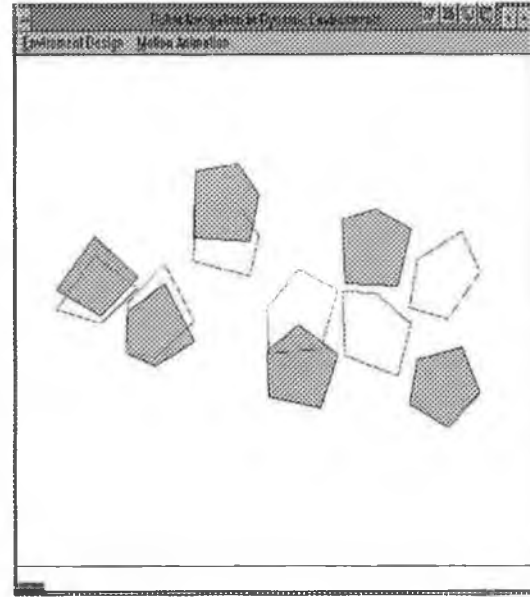
2. Remove a vertex, say V , from the queue OPEN whose associated accessible time is the least.
3. If vertex V in step (2) is the destination point D , then report the motion and exit. Otherwise insert all the vertices that are adjacent to V in the accessibility graph into queue OPEN, and repeat step (2) and (3).

It has been proven that it takes $O(n^2 \cdot \log(n))$ time for this algorithm to compute a minimum time motion in a dynamic environment, where n is the total number of vertices of all the polygonal obstacles in the environment [Fujimura, K. 1988] [Fujimura, K. 1992].

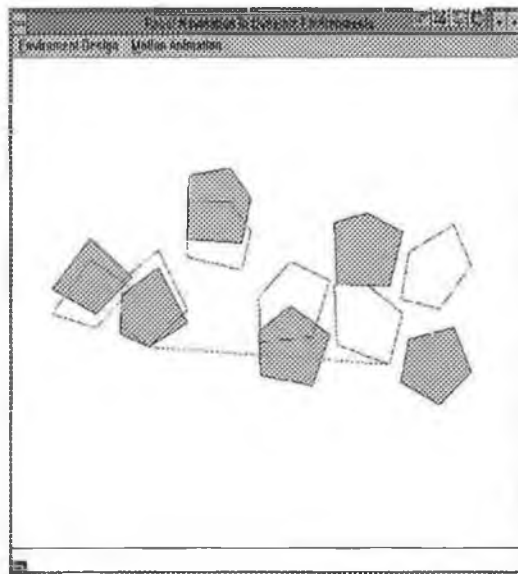
Fig. 5.2 is one of the simulation results by using this algorithm. There are six polygonal obstacles in the environment which are all moving. All the obstacles are numbered from left to right with ascendant order. Obstacles 1, 3 and 5 move in the up down direction; obstacles 2, 4 and 6 move in the down up direction. The filled polygons in the Fig. are the obstacles and the unfilled polygons are the accessibility polygons (the concept of the accessibility polygon will be defined later) of the obstacles. The robot moves from the starting point to the accessible point of a vertex of obstacle 2 in the first step and meet that vertex of obstacle 2 at the end of this step. Then the robot moves from that vertex of obstacle 2 which the robot meets at the end of the first step to the accessible point of another vertex of obstacle 2 in the second step and meet another vertex of obstacle 2 at the end of the second step. And then the robot moves from the vertex of obstacle 2 to the accessible point of a vertex of obstacle 5 in the third step and meet the vertex of obstacle 5 at the end of this step. Finally, the robot moves from that vertex of obstacle 5 which the robot meets at the end of the third step to the destination point in the fourth step and meet the destination point at the end of this step.



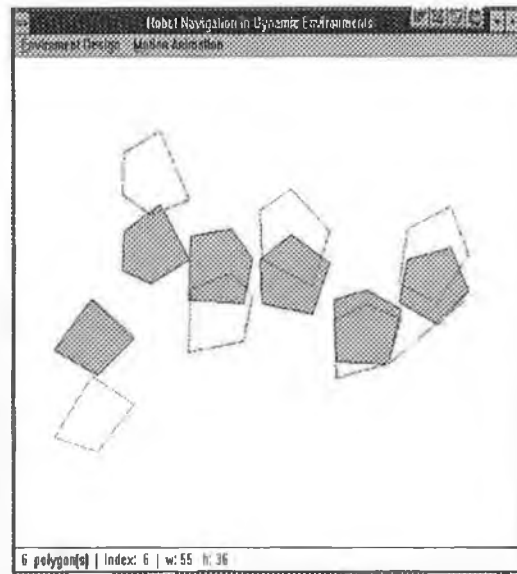
Step 1



Step 2



Step 3



Step 4

Fig. 5.2: Simulation result by accessibility graph algorithm

This algorithm assumes that the robot can accelerate from zero speed to its maximum speed in no time (that is, the robot is massless) and when the robot is about to turn around a corner to change its orientation, there is no need for the robot to decelerate. This is not practical in real applications. In real applications, the mass of the robot must be taken into consideration, that is, it takes time for the robot to accelerate from

zero speed to its maximum speed. When the robot is about to turn around a corner to change its orientation, the robot should decelerate to a speed which is less than its maximum speed and is evaluated by (4.5) in chapter 4. In [Fujimura, K. 1989] and [Fujimura, K. 1992], this problem is called the constrained motion planning problem. The space time approach is used to solve the problem, details are given in chapter 2. The problem of motion planning in a two dimensional dynamic environment is transformed into a three dimensional space problem. The octree representation is used to divide the three dimensional space until in each cell there is only one element of the obstacles (a line edge or a vertex of a polygon is called a element of the polygon) or the cell is empty. The memory needed for the computation is of exponential order and the computation is very time consuming.

Because of the inefficiency of the space time approach, a more efficient approach is needed to solve the motion planning problem for robots with mass. The accessibility graph algorithm will be improved and extended to realise this.

5.3 Accessibility Polygon and Its Applications in Motion Planning

The concept of the accessibility polygon is very important to extend the above algorithm to solve the constrained motion planning problems. It is defined as follows.

Accessibility Polygon: Suppose a polygon P moves in direction d_L with speed v_L and the robot is initially at S at time t_0 and the maximum speed of the robot is v . Only the movement of vertex points of the polygon are considered. If v is bigger than v_L the robot can meet all the vertex points of the polygon after some time. Calculate all the meeting points of the vertices of P and draw lines (edges) which link these meeting points in the same order corresponding to the lines (edges) which form polygon P . The meeting time of every vertex is also associated with every meeting point. A polygon is so formed and is defined as the accessibility polygon of P with respect to starting point S , starting time t_0 and speed v .

Fig. 5.3 is a polygon ABCDE which moves in a fixed direction with speed v_L . The meeting points of its vertices A, B, C, D and E are m_A , m_B , m_C , m_D and m_E respectively. Draw the lines $m_A m_B$, $m_B m_C$, $m_C m_D$, $m_D m_E$ and $m_E m_A$ and the polygon $m_A m_B m_C m_D m_E$ is formed and is the accessibility polygon of polygon ABCDE with respect to starting point S, starting time t_0 and speed v .

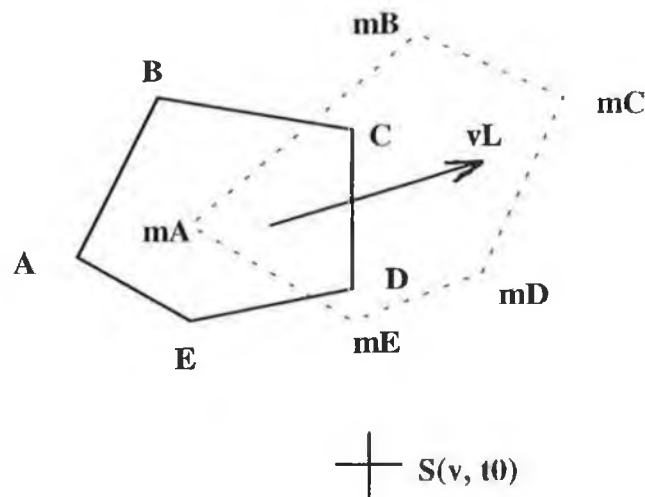


Fig. 5.3: Accessibility polygon

If polygon P is a convex polygon, it is easy to prove that the accessibility polygon of P is also a convex polygon.

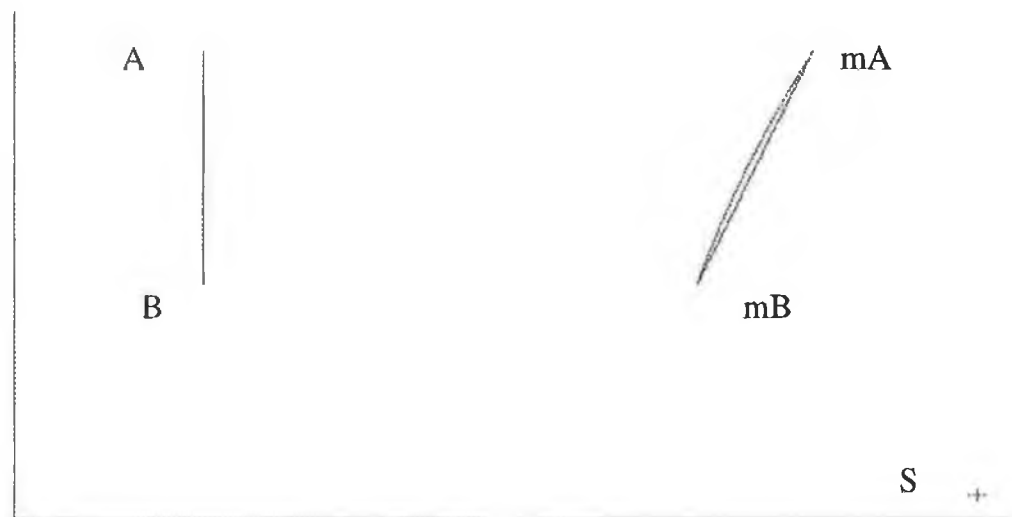


Fig. 5.4: Error Between Line and Collision Front

When the accessibility graph is constructed, the vertex point of a polygonal obstacle must be checked to see if it is accessible from a starting point or not. The check of accessibility is carried out by seeing if the line which links the starting point and the meeting point of the checked point with respect to the starting point intersects with any of the collision fronts of all the movements in the environment or not. Because the collision front of a movement is usually a conic curve segment, the check of its intersection with a line is not as easy as the check of the intersection of two lines. If a line which links the two ending meeting points of a movement is used instead of the conic curve segment collision front, the check of intersection will be easier. Fig. 5.4 shows the error between the line $mAmB$ and the collision front. Line AB is moving in X direction with speed 20 and the robot's initial position is at S with maximum speed 40. mA and mB are meeting points of A and B with respect to S and speed 40 respectively. It is easy to prove that when the accessibility graph is constructed the check of accessibility of any vertex of any polygon in the environment is equal to checking if the meeting point of that vertex is visible or not in the visibility graph of all the accessibility polygons of the polygonal obstacles in the environment (with the selected starting point and destination point as starting point and destination point respectively in the construction of visibility graph). The accessibility polygon in fact is the polygon which is formed by replacing the collision fronts of a moving polygon with line segments.

The accessibility graph can be constructed by the following procedure based on the concept of accessibility polygon.

1. Insert the starting point S in the vertex of the accessibility graph and set its default accessible time to t_0 .
2. For every newly added vertex U in the set of vertices in the accessibility graph, calculate all the accessibility polygons of the polygonal obstacles in the environment with U as the initial point. Find all the visible vertices of the

accessibility polygons and insert them and all the line edges from U to these visible vertices into the accessibility graph together with their accessible time items.

The accessibility graph constructed by this procedure can also be made finite by adopting the same rule that when a vertex appears more than once, only the instance with smallest accessible time is kept.

The minimum time motion planning algorithm can also be stated in the following manner based on the concept of the accessibility polygon.

1. Insert the starting point S into the queue OPEN.
2. Remove a vertex V whose associated accessible time is the least from queue OPEN.
3. If V is the destination point then report the motion and exit with success. Otherwise calculate all the accessibility polygons of the moving polygonal obstacles in the environment with V as the initial point. Find all the visible vertices (including the destination point) of the accessibility polygons from V and put them into V 's successors set VM. Compare every node in VM with the nodes in OPEN, if the node does not appear in OPEN then put it into OPEN, else compare the associated accessible time of the node with the associated accessible time of the old node in OPEN, if the associated accessible time of the node in VM is less, then replace the old node in OPEN with that node in VM, else delete the node from VM. Go to step 2.

The simulation result of this algorithm is the same as the algorithm given in the above section (see Fig. 5.2).

5.4 Improved Accessibility Graph

The accessibility graph algorithm can only generate a minimum time motion for a mobile robot whose mass is not taken into consideration. If the mass of the robot is considered, its mathematical model is the same as (4.1) to (4.6), except that the obstacles in (4.6) are moving. It is pointed out in chapter 4 that there is no analytical solution to such a problem. An approximate solution, the Minimum Time Path at Visibility Node, is obtained in chapter 4. In this section, an approximate solution to the minimum time motion planning for a mobile robot with mass will be discussed based on the improved accessibility graph. To simplify problems, we will use the term “minimum time motion”, although it is only an approximate minimum time motion.

As discussed above, based on the concept of the accessibility polygon, a similar motion planning algorithm can be used to plan the minimum time motion for a mobile robot in a dynamic environment. If the accessibility polygon is revised, the accessibility graph can also be revised. Based on the revision of the accessibility polygon and the accessibility graph, a new approach for planning the minimum time motion of a mobile robot with mass (that is, it can not accelerate from zero speed to its maximum speed in no time) can be derived.

5.4.1 Improved Accessibility Polygon

The improved accessibility polygon is based on the definitions of the lowest vertex and the highest vertex of a moving polygon with respect to the polygon's velocity.

Lowest Vertex: Suppose a convex polygon moves in a fixed direction with constant speed, as shown in Fig. 5.5. The co-ordinate frame XY is rotated in counter-clock direction so that the new X axis X' is in the same direction as the moving velocity v_L of the moving polygon. Calculate all the co-ordinates of the vertices of the polygon in $X'Y'$. The vertex with the least Y' co-ordinate is defined as the **Lowest Vertex**. Vertex D in Fig. 5.5 is the lowest vertex of polygon $ABCDE$.

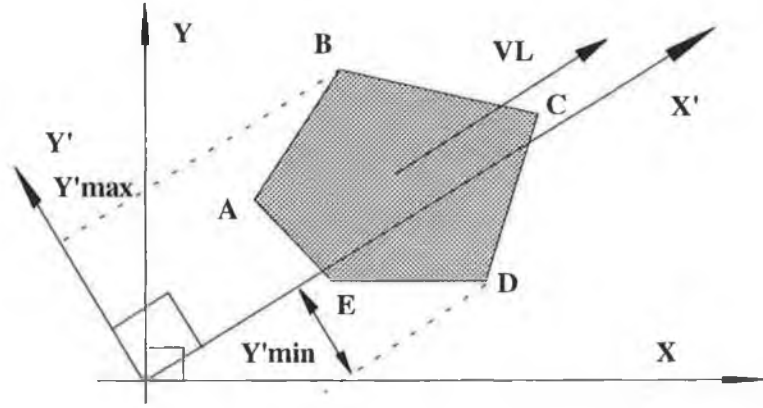


Fig. 5.5: Definitions of lowest vertex and highest vertex

Highest Vertex: Highest Vertex is the opposition of lowest vertex. As shown in Fig. 5.5 the same co-ordinate transformation is made as in the definition of lowest vertex. The vertex with biggest Y' co-ordinate is defined as the **Highest Vertex**. Vertex B in Fig. 5.5 is the highest vertex of polygon ABCDE.

Suppose a mobile robot moves from point A to B along the line path AB, the maximum speed of the robot is $V_{\max}$, the maximum acceleration and deceleration are $A_{\max a}$ and $A_{\max d}$ respectively, the starting speed and ending speed are V_s and V_e respectively. Calculate the peak velocity V_p as follows:

$$V_p^2 = \frac{2A_{\max a}A_{\max d}D_{AB}}{A_{\max a} + A_{\max d}} + \frac{A_{\max d}V_s^2}{A_{\max a} + A_{\max d}} + \frac{A_{\max a}V_e^2}{A_{\max a} + A_{\max d}} \quad (5.2)$$

where, D_{AB} is the distance of line AB. If V_p is equal to or bigger than $V_{\max}$, the distance D_{AB} is said to be long enough for the robot to accelerate from V_s to its maximum speed $V_{\max}$ and decelerate from $V_{\max}$ to V_e . And the time for the robot to move from A to B with the starting speed V_s at point A and ending speed V_e at point B is

$$t = \frac{D_{AB}}{V_{\max}} + \frac{(V_{\max} - V_s)^2}{2A_{\max a} V_{\max}} + \frac{(V_{\max} - V_e)^2}{2A_{\max d} V_{\max}} \quad (5.3)$$

Improved Accessibility: A point robot R with maximum speed $V_{\max}$, maximum acceleration $A_{\max a}$ and maximum deceleration $A_{\max d}$ is planning to move from starting point S with initial speed V_s to destination point D in an environment with moving polygonal obstacles $M=\{M_1, M_2, \dots, M_n\}$. Suppose R starts moving at time t_0 at speed V_s . After R starts moving, it will move in a fixed direction and it will accelerate to its maximum speed $V_{\max}$. A point P (P is either the destination point or a point on any polygonal obstacle) is said to be improved accessible from starting point S (with respect to t_0 and initial speed V_s) if R meets P at speed V_e without intersecting with any other movement. The meeting point and meeting time are defined as the improved accessible point (improved meeting point) and improved accessible time of P with respect to starting point S, starting time t_0 , initial speed V_s and ending speed V_e respectively. It is obvious that if P is a stationary point and is improved accessible, then its improved accessible point is P itself.

For simplicity, the **meeting point** with respect to the starting point S and starting time t_0 of a moving point is defined as the point where the robot which can accelerate from initial zero speed to its maximum speed $V_{\max}$ in no time meets with that moving point and is denoted as $MP(S, t_0, V_{\max})$. The **improved meeting point** with respect to starting point S, starting time t_0 , initial speed V_s and ending speed V_e of a moving point is defined as the point where the robot (which starts from S with initial speed V_s , then accelerates to its maximum speed $V_{\max}$ and before it meets with the moving point accelerates to ending speed V_e) meets with that moving point at speed V_e and is denoted as $IMP(S, t_0, V_s, V_e, V_{\max}, A_{\max a}, A_{\max d})$.

To simplify the problem, suppose the robot starts moving from the origin of the co-ordinate (0, 0) at time t_0 and at that time the moving point is at position (x_0, y_0) . The vector of the speed of the moving point is (V_{Lx}, V_{Ly}) . The co-ordinates (x, y) of $MP(S, t_0, V_{\max})$ can be given by solving the following equations:

$$\begin{cases} t = \frac{\sqrt{x^2 + y^2}}{V_{\max}} \\ x = x_0 + V_{Lx} \times t \\ y = y_0 + V_{Ly} \times t \end{cases} \quad (5.4)$$

In the same way the co-ordinates (x, y) of IMP(S, t_0 , V_s , V_e , $V_{\max}$, $A_{\max a}$, $A_{\max d}$) can be given by solving the following equations:

$$\begin{cases} t = \frac{\sqrt{x^2 + y^2}}{V_{\max}} + \frac{(V_{\max} - V_s)^2}{2A_{\max a} V_{\max}} + \frac{(V_{\max} - V_e)^2}{2A_{\max d} V_{\max}} \\ x = x_0 + V_{Lx} \times t \\ y = y_0 + V_{Ly} \times t \end{cases} \quad (5.5)$$

Improved Collision Front: Consider an environment that contains one movement (L, d_L, v_L) . Let A and B be the two endpoints of L and let imA and imB be the improved accessible points of A and B with respect to the point robot R's starting point S, starting time t_0 , initial speed V_s and ending speed V_e , respectively. The set of improved accessible points corresponding to all points on L forms a curve segment and this curve segment is called the improved collision front of L with respect to starting point S, starting time t_0 , initial speed V_s and ending speed V_e . Fig. 5.6 shows an example of the improved collision front. The starting point of the robot is assumed to be the origin of the co-ordinates and L is a line segment which is parallel to the Y axis with length Y_L , the equations of the improved collision front are:

$$\begin{cases} t(A) = \frac{\sqrt{x^2 + y^2}}{V_{\max}} + \frac{(V_{\max} - V_s)^2}{2A_{\max a} V_{\max}} + \frac{(V_{\max} - V_s)^2}{2A_{\max d} V_{\max}} = \frac{\sqrt{(x - x_0)^2 + (y - a)^2}}{v_L} \\ y = (x - x_0) \tan \theta + a \\ y_0 \leq a \leq y_0 + Y_L \end{cases}$$

where $t(A)$ is the improved accessible time, θ is the angle between the X axis and the velocity vector of the movement, $V_{\max}$ is the maximum speed of the robot, $A_{\max a}$ and $A_{\max d}$ are the maximum acceleration and maximum deceleration of the robot respectively. The curve form of the improved collision front is the same as that of the collision front.

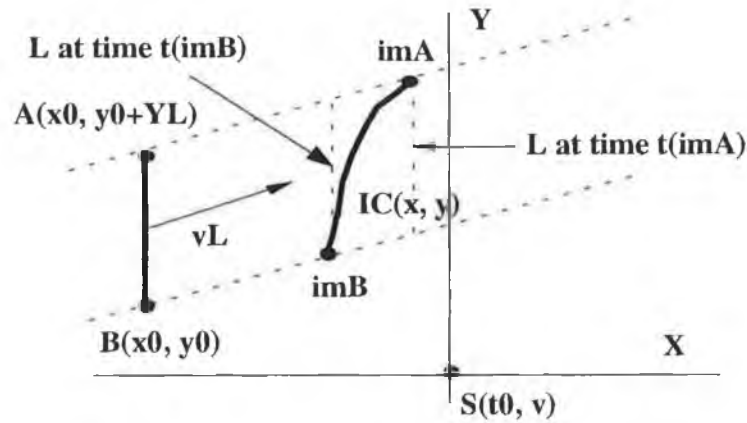


Fig. 5.6: Improved collision front of a movement

Improved Accessibility Polygon: Suppose a polygon P moves in direction d_L with speed v_L and the robot is initially at S at time t_0 with initial speed 0 and the maximum speed of the robot is $V_{\max}$. Only the movements of the vertices points of the polygon are considered. If $V_{\max}$ is bigger than v_L the robot can meet all the vertex points of the polygon after some time. Suppose the polygon is far enough away from the robot at the very beginning so that the robot can accelerate to its maximum speed before it meets with any vertex of the polygon. Construct a polygon as follows:

1. Calculate $\text{IMP}(S, t_0, 0, 0, V_{\max}, A_{\max a}, A_{\max d})$ and $\text{MP}(S, t_0, V_{\max})$ of the lowest vertex and $\text{IMP}(S, t_0, 0, 0, V_{\max}, A_{\max a}, A_{\max d})$ and $\text{MP}(S, t_0, V_{\max})$ of the highest vertex of the moving polygon respectively.
2. Assign the lowest vertex an index of 0 and then run through all the vertices of P in the direction of the velocity vector of P, assign each vertex an index in ascendant order from 0.
3. For every other vertex except the lowest vertex and the highest vertex, if the index assigned to it in (2) is bigger than 0 and less than the index of the highest index, then calculate the improved meeting point $\text{IMP}(S, t_0, 0, 0, V_{\max}, A_{\max a}, A_{\max d})$ of that vertex. Else calculate the meeting point $\text{MP}(S, t_0, V_{\max})$ of the vertex.
4. Draw a line (edge) between $\text{IMP}(S, t_0, 0, 0, V_{\max}, A_{\max a}, A_{\max d})$ and $\text{MP}(S, t_0, V_{\max})$ of the lowest vertex and the highest vertex respectively.
5. Draw lines (edges) which link all the IMPs in the same order with respect to the same order in the moving polygon P. In the same manner, draw lines (edges) which link all the MPs in the same order as the moving polygon P.

The polygon constructed in the above procedure is defined as the **improved accessibility polygon**. As show in Fig. 5.7, polygon ABCDEF is moving with velocity V_L . A and D are the lowest vertex and highest vertex of the polygon respectively. So the index of A is 0, index of B is 1, index of C is 2, index of D is 3, index of E is 4 and index of F is 5. mA_1 is the meeting point of vertex A, mA_2 is the improved meeting point of A and mD_1 is the meeting point of vertex D, mD_2 is the improved meeting point of D. The indexes of vertices B and C are bigger than 0 and less than 3 so mB and mC are the improved meeting points of vertex B and vertex C respectively. In the same manner, mE and mF are the meeting points of vertex E and vertex F respectively. Polygon $mA_1mA_2mBmCmD_2mD_1mEmF$ is the improved accessibility polygon of polygon ABCDEF (see Fig. 5.7).

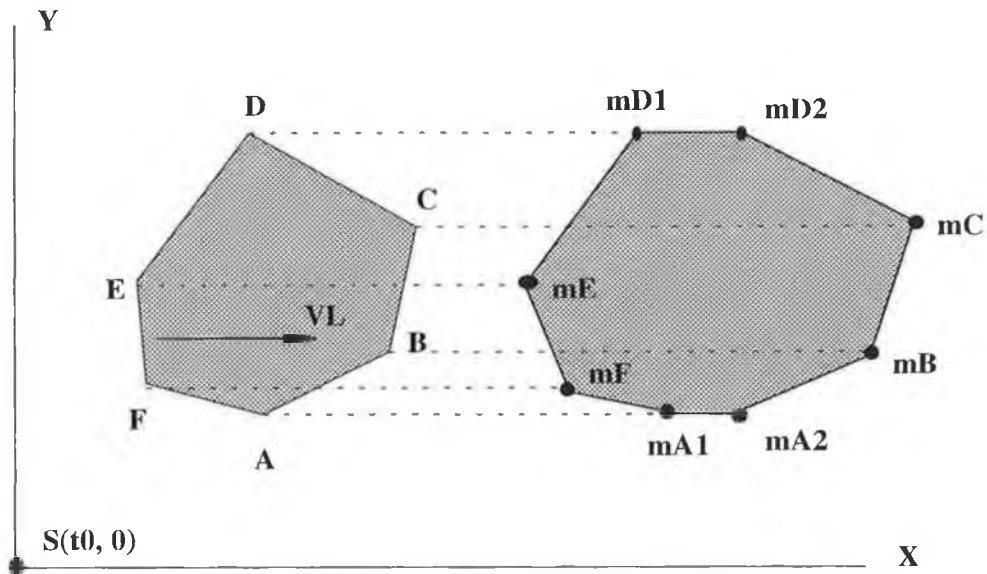


Fig. 5.7: Definition of improved accessibility polygon

The improved accessibility polygon can also be described in the following manner. For edges of a moving polygonal obstacle which are ahead of both the its highest point and lowest point in the direction of the motion, calculate their improved collision fronts and use line segments to replace them. For other edges, calculate their collision fronts and also use line segments to replace them. Link the accessible points and the improved accessible points of the highest vertex and the lowest vertex with line segments respectively. This will form the improved accessibility polygon of a polygonal obstacle.

5.4.2 Improved Accessibility Graph

In the last section, the accessibility graph can be defined and constructed based on the accessibility polygons of the polygonal obstacles $M=\{M_1, M_2, \dots, M_n\}$ in the environment. The improved accessibility graph can be defined and constructed in the similar manner based on the definition of the improved accessibility polygon.

Improved Accessibility Graph (IAG): Suppose a robot is planning to move from a starting point S to a destination point D in a environment with moving obstacles $M=\{M_1, M_2, \dots, M_n\}$. Based on the definition of improved accessibility polygon, the improved accessibility graph can be defined and constructed as follows:

Let S be the starting point, t_0 be the starting time, $v_{\max}$ be the maximum speed of the mobile robot, $A_{\max a}$ and $A_{\max d}$ be the maximum acceleration and maximum deceleration of the robot respectively, D be the destination point. With each accessible point or improved accessible point X corresponding to point V , an item of time $t(X)$ (accessible time or improved accessible time) is associated with X to denote X 's accessible time with respect to V .

1. Insert the starting point S into the improved accessibility graph IAG and set its improved accessible time to t_0 .
2. For every newly added node N in IAG, calculate the improved accessibility polygons (with respect to starting point N) of all the polygonal obstacles $M=\{M_1, M_2, \dots, M_n\}$. Find all the visible vertices (including the destination point D) of the improved accessibility polygons with N as the standing point and insert them and all the line edges from N to these visible vertices into the improved accessibility graph IAG together with their improved accessible time or accessible time items.

According to the definition, the improved accessibility graph is infinite. But it can be made finite by adopting the rule that when a vertex of an improved accessibility polygon appears more than once, then the instance with the smallest improved accessible time is kept. Under this rule, the number of vertices of the improved accessibility graph does not exceed the number of vertices of all the improved accessibility polygons of the polygonal obstacles $M=\{M_1, M_2, \dots, M_n\}$ plus 2 (starting and destination points must be added).

5.4.3 Motion Planning Algorithm for Robots with Mass

Suppose when the robot arrives at a turning point (a vertex of the improved accessibility graph) it must decelerate to zero speed to pass that point and then accelerate in the next line path. The following procedure can generate the minimum time path for a mobile robot with mass (that is, it can not accelerate from zero speed to its maximum speed in no time).

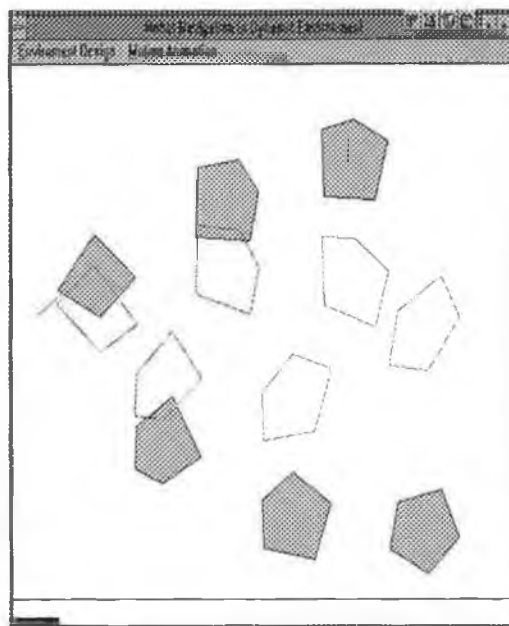
1. Insert the starting point S into the queue OPEN.
2. Remove a vertex V whose associated improved accessible time is the least from OPEN.
3. If V is the destination point then report the motion and exit with success. Otherwise calculate all the improved accessibility polygons of the polygonal obstacles in the environment with V as the initial point. Find all the visible vertices (including the destination point) of the improved accessibility polygons from V and put them into V 's successors set VM. Compare every node in VM with the nodes in OPEN, if the node does not appear in OPEN then put it into OPEN, else compare the associated improved accessible time of the node with the associated improved accessible time of the old node in OPEN, if the associated accessible time of the node in VM is less, then replace the old node in OPEN with that node in VM, else delete the node from VM. Go to step 2.

The supposition that the robot must decelerate to zero speed at a turning point is not practical. As discussed in chapter 4, the velocity of the robot at a turning point can be decided by (4.5), that is, the velocity is a function of the angle which the robot is about to turn to change its orientation. This assumption will lead to the complete construction of the improved accessibility graph and the depth first search strategy for the same graph, which is very inefficient and not acceptable for practical applications. This problem is also solved in a similar manner as discussed in chapter 4 by introduction of the concept of point of invariance (POI) of a vertex point. The

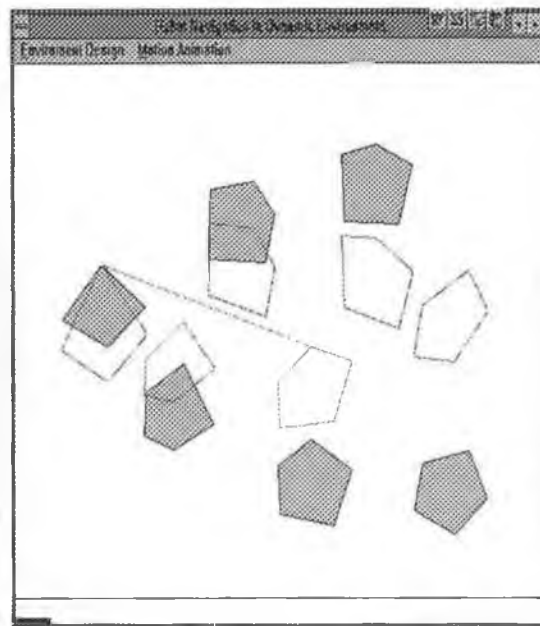
definition of POI is given in chapter 4. The evaluation of the improved accessible time for every visible vertex of the improved accessibility polygons is the time for the robot to travel from the point of invariance of the current standing point to the point of invariance of that visible vertex, that is, using (4.17) to calculate the improved accessibility time items. Production rules developed in Fig. 4.7 in chapter 4 can be also used to evaluate the time optimal trajectory for the robot to move from the POI of the current standing point to the POI of the visible vertex point of the improved accessibility polygon. Based on this, the algorithm which can generate the minimum time motion for a mobile robot with mass can be stated as follows.

1. Insert the starting point S into the queue OPEN.
2. Remove a vertex V whose associated improved accessible time is the least from OPEN.
3. If V is the destination point then report the motion and exit with success. Otherwise calculate all the improved accessibility polygons of the polygonal obstacles in the environment with V as the initial point. Find all the visible vertices (including the destination point) of the improved accessibility polygons from V and put them into V 's successors set VM. Using production rules developed in Fig. 4.7 in chapter 4 to evaluate the improved accessible time for each visible vertex from the POI of V to the POI of that vertex. Compare every node in VM with the nodes in OPEN, if the node does not appear in OPEN then put it into OPEN, else compare the associated improved accessible time of the node with the associated improved accessible time of the old node in OPEN, if the associated accessible time of the node in VM is less, then replace the old node in OPEN with that node in VM, else delete the node from VM. Go to step 2.

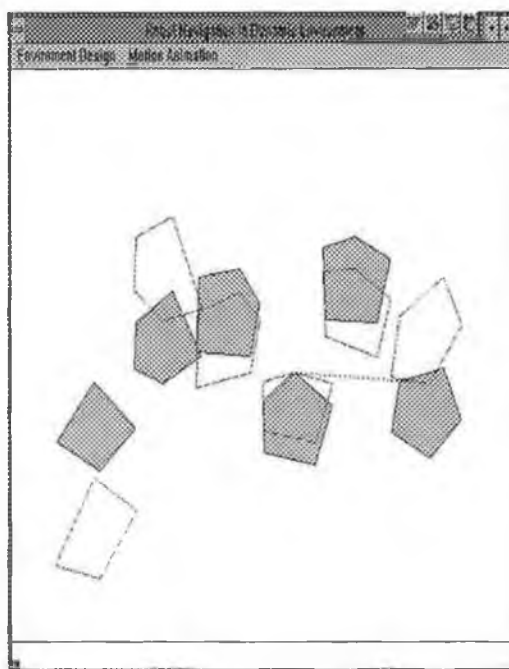
Fig. 5.8 is one of the simulation results obtained by this algorithm. Fig. 5.9 is the velocity profile of the robot along the path in Fig. 5.8.



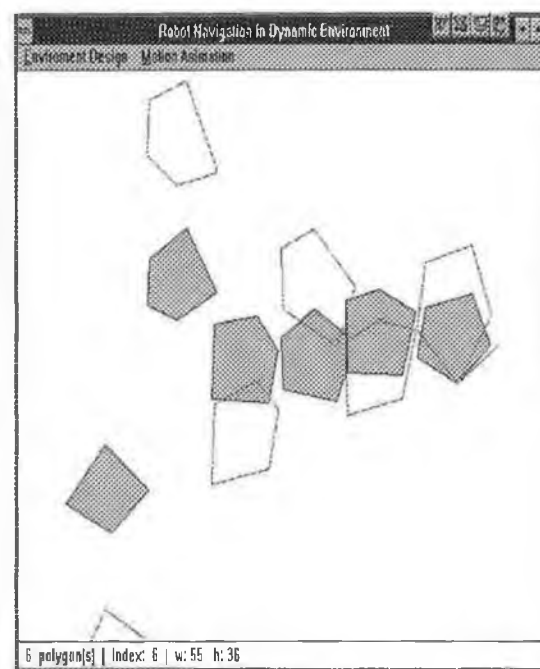
Step 1



Step 2



Step 3



Step 4

Fig. 5.8: Simulation result by the improved accessibility graph algorithm

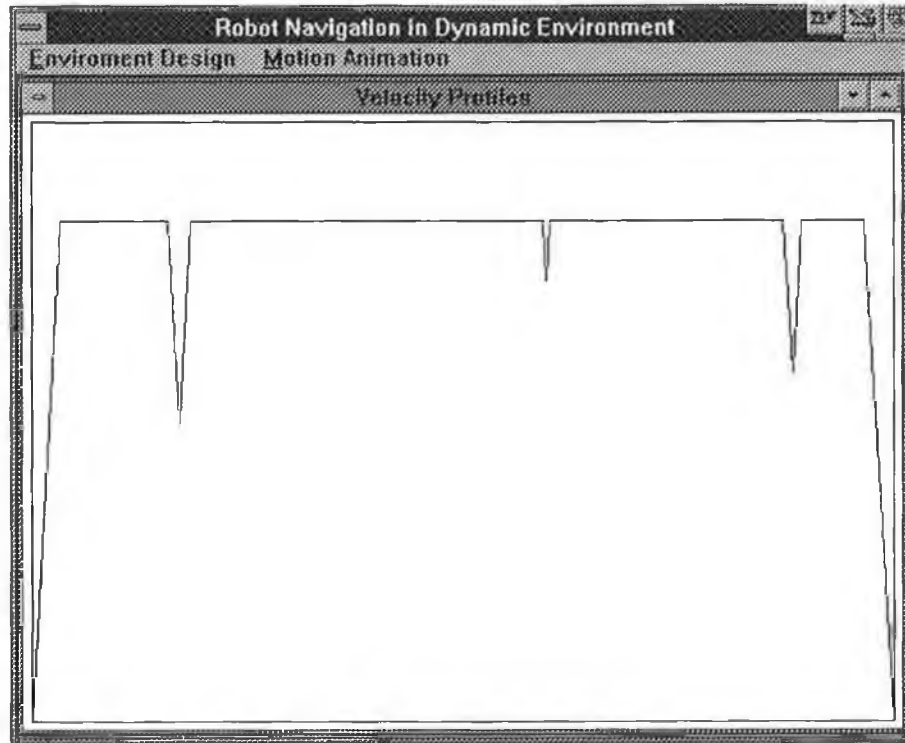


Fig. 5.9: Velocity profiles for the motion in Fig. 5.8

As mentioned above, it takes $O(n^2 \cdot \log(n))$ time for the accessibility graph algorithm to compute the minimum time motion in a dynamic environment, where n is the total number of vertices of all the polygonal obstacles. It is easy to evaluate the computation time for the improved accessibility graph algorithm. For an environment with k polygonal obstacles, n is the total number of vertices of these polygons, then there are $n+2k$ vertices among all the improved accessibility polygons of those k polygonal obstacles. Replace n with $n+2k$ in the above result, the following result is obvious.

It takes $O((n+2k)^2 \cdot \log(n+2k))$ time for the improved accessibility graph algorithm to compute the minimum time motion for a mobile robot with mass in a dynamic environment, where k is the number of polygonal obstacles in the environment and n is the total number of vertices of all these k polygonal obstacles. Ordinarily k is much bigger than 2, that is, k can be used to replace $2k$, so the improved accessibility graph algorithm takes $O((n+k)^2 \cdot \log(n+k))$ time to generate a minimum time motion.

Because k is much less than n , the computation time for the accessibility graph algorithm and the improved accessibility graph algorithm is almost the same.

5.5 Applications

The general properties of the accessibility graph and the improved accessibility graph have been presented and the algorithms based on them for motion planning have been developed in the above sections. Simulation results show that they are effective. This section will introduce the applications of the general properties of them in some special cases.

5.5.1 Applications of Theory of Sectors to Accelerate the Search

The discussion of applications of the theory of sectors which is introduced in chapter 3 is applicable to both the accessibility graph algorithm and the improved accessibility graph algorithm. When calculating the accessibility polygon or the improved accessibility polygon, the assumption that the moving polygon is a convex polygon is not necessary. In fact, both algorithms can be used to generate the minimum time motion in dynamic environments with concave polygonal obstacles without any modification. The graph search of both algorithms can be accelerated by using the theory of sectors when finding the visible vertices of the accessibility polygons or the improved accessibility polygons from the current standing point. The approach is the same as in chapter 3, that is, there are at most two vertices of each accessibility polygon or improved accessibility polygon which comprise the main sector to be expanded into queue OPEN. With the applications of the theory of sectors, it takes $O(n \cdot k \cdot \log(k))$ times for the accessibility graph algorithm to generate the minimum time motion in a dynamic environment, where k is the total number of obstacles in the environment and n is the total number of vertices of all those polygonal obstacles [Fujimura, K. 1992]. For the improved accessibility graph algorithm, just replace n with $n+k$, that is, it takes $O((n+k) \cdot k \cdot \log(k))$ times for the improved accessibility

graph algorithm to generate the minimum time motion. Because k is much less than n , it is obvious that the applications of theory of sectors can accelerate the search for both the accessibility graph algorithm and the improved accessibility graph algorithm.

5.5.2 Circular Obstacles and Convex Curved Obstacles

There are two approaches to solve the minimum time motion planning problems in a dynamic environment with moving circular obstacles, that is, the moving obstacles are in the form of circle. The first approach represents the circle with an approximating polygon and use the above algorithm to plan the minimum time motion for the robot. This will be discussed in detail in chapter 7. The second approach uses the similarity between the accessibility polygons of a dynamic environment and the polygonal obstacles in a static environment and the result of constructing the visibility graph among generalised polygons. In fact, the construction of the accessibility graph is similar to the construction of the visibility graph of the accessibility polygons. The difference is that when a new standing point is selected, the accessibility polygons have to be calculated with respect to that standing point. For a circular moving obstacle, its meeting points corresponding to the points along the circle will comprise a convex curved polygon, as shown in Fig. 5.10. A and B are the lowest vertex and highest vertex of the moving circular obstacle respectively and m_A and m_B are the meeting points of A and B respectively. If the mass of the robot is not taken into consideration, the convex curved meeting polygon will be called the generalised accessibility polygon and the points along its border will be the meeting points calculated by (5.4). If the mass of the robot is taken into consideration, that is, it takes time for the robot to accelerate from zero speed to its maximum speed, the meeting curved polygon will be called the generalised improved accessibility polygon and is shown in Fig. 5.11. m_{A_1} and m_{A_2} are the meeting point and improved meeting point of A respectively and m_{B_1} and m_{B_2} are the meeting point and improved meeting point of B respectively. The points along the circle which are ahead of AB in the direction of v_L will be calculated by using (5.5) to get the corresponding improved meeting points which are points from m_{A_2} to m_{B_2} in Fig. 5.11. The other

points along the circle will be calculated by using (5.4) to get the corresponding meeting points which are points from mB_1 to mA_1 .

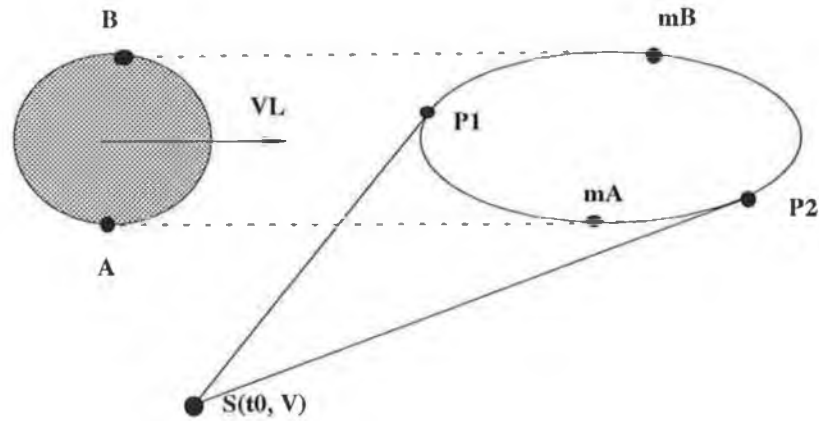


Fig. 5.10: Generalised accessibility polygon of a circle

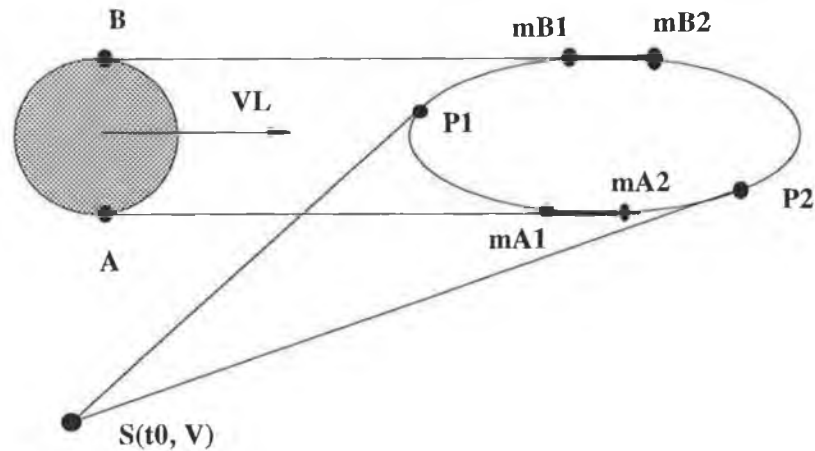


Fig. 5.11: Generalised improved accessibility polygon of a circle

With the generalised accessibility polygon or generalised improved accessibility polygon defined above, the tangent points on the generalised accessibility polygons or generalised improved accessibility polygons from the current standing point SP will be found and selected as successors of SP as the visible vertices points in the algorithms developed above. Two cases must be considered when expanding the successors of SP. The first case is that SP is the starting point S which is shown in Fig. 5.10 and Fig. 5.11. The generalised accessibility polygon or the generalised

improved accessibility polygon is computed and tangent points (on the generalised accessibility polygon or the generalised improved accessibility polygon) from SP will be successors of SP, P_1 and P_2 are two of these points in Fig. 5.10 or Fig. 5.11.

The second case is more complicated than the first case. As shown in Fig. 5.12, the current SP is a point on the generalised accessibility polygon or the generalised improved accessibility polygon which is P_2 in the Fig. The generalised accessibility polygons or generalised improved accessibility polygons are computed with P_2 as the initial point, here, there are two circular obstacles, the improved accessibility polygon of the upper obstacle with P_2 as the initial point is shown in Fig. 5.12 as $mA_1mA_2mB_2mB_1$. To expand P_2 's successors, the common tangent lines between two generalised accessibility polygons are computed, EP in Fig. 5.12 is one of these lines and the tangent point P on the upper generalised accessibility polygon is one of the successors of P_2 . If the velocity of the robot is faster than any velocity of the obstacles, the robot can move along the border of the generalised accessibility polygon or generalised improved accessibility polygon which is P_2E in the Fig., to the next tangent point which is E (E is also called the exit tangent point). After that, the robot will move along the tangent line EP to P_2 's successor P. Because the upper generalised accessibility polygon $mA_1mA_2mB_2mB_1$ is computed with P_2 as the initial point, if the time for the robot to move along P_2E and EP is longer than it takes to move along the line segment P_2P , the robot will collide with the circular obstacle at P. In this case, the generalised accessibility polygon $mA_1mA_2mB_2mB_1$ must be revised to avoid such collision. This is realised by extending the generalised accessibility polygon in the direction of the movement of that circular obstacle and is shown in Fig. 5.12. With the generalised accessibility polygon extended, the successor of P_2 now becomes V which is shown in Fig. 5.12. The robot will next move along VU and UD to reach the destination point D as shown in Fig. 5.12. This is similar to the smoothed shortest path in chapter 3.

To extend the generalised accessibility polygon to avoid the collision between the obstacle and the robot, some compensating time must be added when the generalised accessibility polygon or generalised improved accessibility polygon are computed by

(5.4) and (5.5). For points which are ahead of the lowest and highest vertex points in the direction of motion of the obstacle, the compensating time is added to (5.4) and (5.5), while for the other points on the bound of the circular obstacle, there is no need to add this compensating time. It is difficult to find this compensating time, since it is difficult to find the tangent point for the robot to leave the border of the generalised accessibility polygon or the generalised improved accessibility polygon. It is recommended that a longer time is added to give more clearance between the obstacle and the robot, resulting in a safer motion.

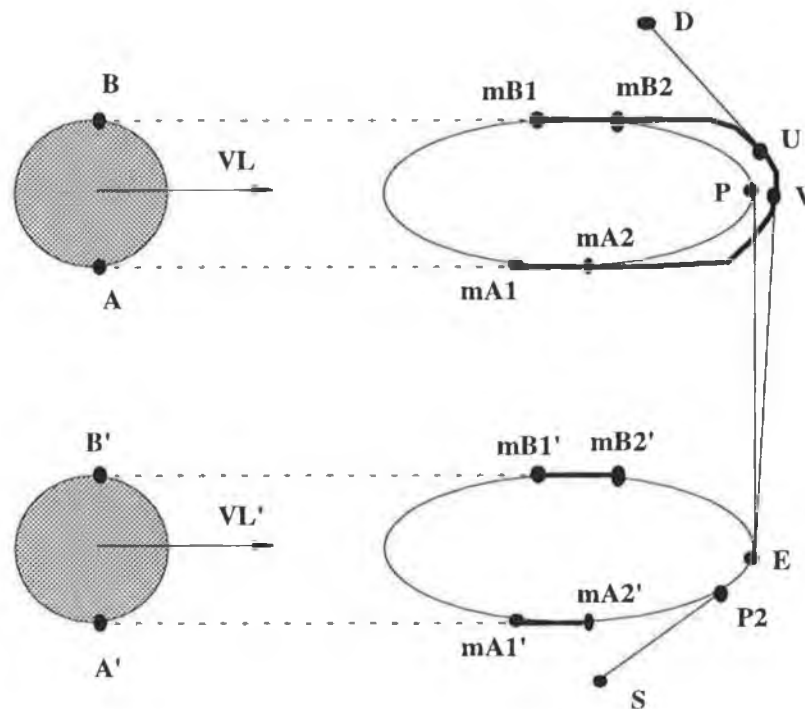


Fig. 5.12: Enlarge the generalised accessibility polygon

It should be pointed out that the approach proposed above to deal with circular obstacles can be used to deal with the convex curved obstacles without any modification.

The other approach which approximately represents a curved polygon with a polygon provides an alternative and will be discussed in detail in chapter 7.

If the mass of the robot is not taken into consideration, the meeting points of the vertices of the growing or shrinking obstacle with respect to a starting point can be calculated and the accessibility polygon with respect to starting point S , starting time t_0 and velocity v can be formed with the similar manner developed above. To make the problem simple, the starting point of the robot is supposed to be at the origin of the co-ordinates, that is, $S(0, 0)$, the meeting point can be calculated by solving the following equations.

$$\begin{cases} t - t_0 = \frac{\sqrt{x^2 + y^2}}{v} \\ x = c(t - t_0)(x_0 - x_P) \\ y = c(t - t_0)(y_0 - y_P) \end{cases} \quad (5.7)$$

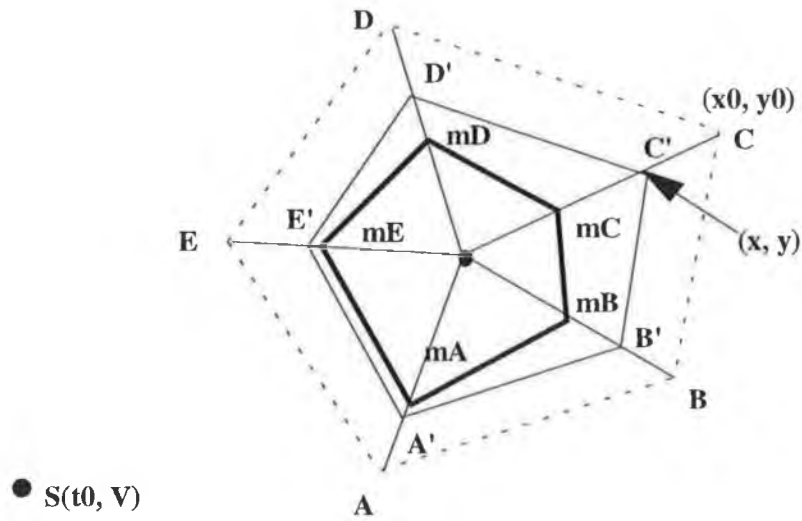


Fig. 5.14: Shrinking obstacle and its accessibility polygon

The accessibility polygon of a growing or shrinking polygonal obstacle can thus be formed by linking all the meeting points of its vertices with respect to S , time t_0 and velocity v by line segments which are in the same sequence as the line segments that form the polygon.

If the mass of the robot must be taken into consideration, that is, it takes time for the robot to accelerate from zero speed to its maximum speed, there will be difference between the growing obstacle and the shrinking obstacle when the improved accessibility polygon is formed. For a shrinking obstacle, the formation of its improved accessibility polygon is the same as the formation of the accessibility polygon mentioned above. For a growing obstacle, the following equations are solved to get the improved meeting points of its vertices with respect to starting point $S(0, 0)$, starting time t_0 and velocity $v_{\max}$.

$$\begin{cases} t - t_0 = \frac{\sqrt{x^2 + y^2}}{v_{\max}} + \frac{v_{\max}}{2A_{\max a}} + \frac{v_{\max}}{2A_{\max d}} \\ x = c(t - t_0)(x_0 - x_P) \\ y = c(t - t_0)(y_0 - y_P) \end{cases} \quad (5.8)$$

where, $A_{\max a}$ and $A_{\max d}$ are the maximum acceleration and maximum deceleration of the robot respectively. With the improved meeting points calculated, the improved accessibility polygon can be formed.

The accessibility graph algorithm and the improved accessibility graph algorithm developed in the above sections can be used to search among the accessibility polygons or improved accessibility polygons to generate the minimum time motion in an environment with growing and shrinking obstacles.

It might seem that a dynamic environment with growing and shrinking obstacles is only a theoretical problem, with no real applications. In fact, the problem may be relevant for a ship navigating on the sea where obstacles (e. g., islands) may grow or shrink with tide changes.

5.5.4 Piecewise Linear Motion of the Obstacles

A straight line motion or a single motion velocity may not be sufficient to describe the motion of the obstacles in real applications. A more general description of the motion of obstacles is a piecewise linear motion, that is, the motion consists of a finite number of time intervals during each of which the obstacle moves in a fixed direction with a constant speed.

It is simple to extend the representation of a movement definition to this situation. The movement of a line segment L can be defined as (L, d_L, v_L, TI_L) , where TI_L represent the time interval during which L moves in direction d_L at speed v_L . When the motion of L is piecewise linear as shown in Fig. 5.15 where the line segment L moves in direction d_1 until it overlaps CD , after which it moves in direction d_2 , the collision front of the line segment due to the motion in d_1 can be calculated without any consideration of the next motion in direction d_2 (see Fig. 5.15).

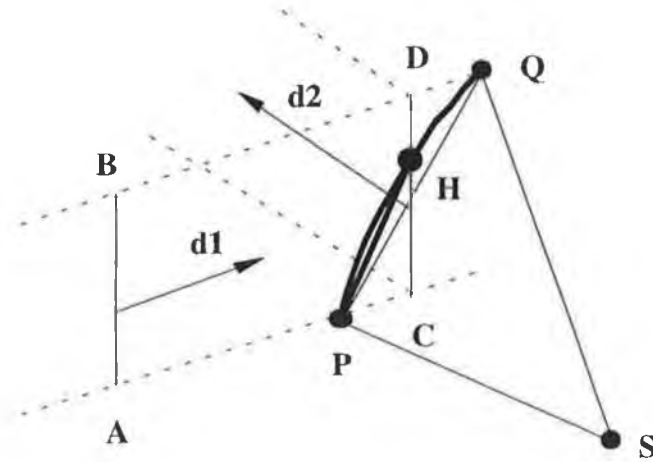


Fig. 5.15: Collision front of a line with piecewise linear motion

The curve PQ is the collision front of the line segment L with respect to the starting point S without any consideration of the changing of the direction when the line overlaps with CD . If the motion of the line segment is not in piecewise linear motion, line segment PQ is an edge of the accessibility polygon of the moving polygonal

obstacle with line AB as one of its edges. Because the accessible time for the points along the collision front curve HQ is out the range of the time interval for the motion of line AB in the direction d_1 , the robot can not meet the corresponding point on line AB at a point on the curve segment HQ. The real collision front of line AB for that piecewise linear motion in direction d_1 is curve segment PH, that is, if a ray from S does not intersect with curve segment PH, a robot moving in the direction of that ray will not intersect with the line segment AB when AB is in the piecewise linear motion in direction d_1 . As discussed in the definition of the accessibility polygon, line segment PH can be one edge of the accessibility polygon of a piecewise linear moving polygonal obstacle with AB as one of its edges when the piecewise linear motion in direction d_1 is considered. The other piecewise linear motions in the other directions can be considered independently as if they are several independent moving obstacles. Special attention must be paid to the time interval when the obstacle changes its moving direction or speed. The accessibility polygon of every linear motion can be constructed as if it is an ordinary motion without changing direction as discussed in the above sections.

With the accessibility polygon of every linear motion constructed in the above manner, the accessibility graph search algorithm can be used to search among the accessibility polygons to generate the minimum time motion in the environment with piecewise linear motion obstacles. The computation time for this search is bigger than the ordinary accessibility graph algorithm. Suppose that there are k polygonal obstacles $M=\{M_1, M_2, \dots, M_k\}$ with piecewise linear motion in the environment, $p_1, p_2, \dots, p_k$ are the numbers of movements for $M_1, M_2, \dots, M_k$ respectively, and $n_1, n_2, \dots, n_k$ are the numbers of vertices of $M_1, M_2, \dots, M_k$ respectively. If all $p_1, p_2, \dots, p_k$ are equal to 1, this problem is the ordinary problem discussed in the above sections, it takes $O(n^2 \cdot \log(n))$ time to generate a minimum time motion in a dynamic environment, where $n=n_1+n_2+\dots+n_k$ is the total number of vertices of the accessibility polygons. Now for piecewise linear motion, there will be $n=p_1 \cdot n_1 + p_2 \cdot n_2 + \dots + p_k \cdot n_k$ vertices in the accessibility polygons. So it is obvious that it takes $O(n^2 \cdot \log(n))$ time to compute the minimum time motion in a dynamic environment with k polygonal obstacles $M=\{M_1, M_2, \dots, M_k\}$, where $n=p_1 \cdot n_1 + p_2 \cdot n_2 + \dots +$

$p_k \bullet n_k$. If the obstacle M_i is an ordinary moving obstacle, its piecewise number of movements p_i will be 1.

When considering motion planning of mobile robots with mass in environments with piecewise linear motion obstacles, the solution is almost the same as the algorithm for the massless robot discussed above. The only difference is that when calculating the improved accessibility polygon of every linear motion of a polygonal obstacle, the method for calculating the improved accessibility polygon must be used instead of the method for the accessibility polygon. The other steps are the same.

5.5.5 Motion Planning in Environments with Accelerating Obstacles

The motions of the obstacles discussed earlier have constant velocity. In this section obstacles that move with an acceleration will be considered. There are two approaches to solve the motion planning problem with accelerating obstacles. The first approach is to approximate the accelerating motion of an obstacle with a number of piecewise linear motions. In other words, the motion with an acceleration may be viewed as a motion where successive approximate motions by piecewise linear movements converge when the number of pieces approaches infinity. In this way, if the speed of the obstacle does not exceed the maximum speed of the robot, the approach to the piecewise linear moving obstacles can generate the approximate minimum time motion for the robot. In real applications, the problem of how many pieces are used to approximate the acceleration remains to be solved.

Another approach to the accelerating movement of obstacles is to solve the non-linear equations to get the meeting point of every vertex of the polygonal obstacle with respect to the current standing point and construct the accessibility polygon of the obstacle. Accessibility graph search algorithm can be used to generate the minimum time motion for the robot.

In general, the x , y components of the accelerating motion are given in the form:

$$\begin{cases} x(t) = a_x t^2 + b_x t + c_x \\ y(t) = a_y t^2 + b_y t + c_y \end{cases} \quad (5.9)$$

Suppose the starting point of the robot is at the origin of the co-ordinates and the velocity of the robot is v , then the co-ordinates of the meeting point of any vertex of the polygonal obstacle can be given by solving the following equations.

$$\begin{cases} t = \frac{\sqrt{x^2 + y^2}}{v} \\ x = a_x t^2 + b_x t + c_x \\ y = a_y t^2 + b_y t + c_y \end{cases} \quad (5.10)$$

where v is the velocity of the robot.

If the mass of the robot is taken into consideration, the improved meeting point can be calculated by solving the equations:

$$\begin{cases} t = \frac{\sqrt{x^2 + y^2}}{v} + \frac{v_{\max}}{2A_{\max a}} + \frac{v_{\max}}{2A_{\max d}} \\ x = a_x t^2 + b_x t + c_x \\ y = a_y t^2 + b_y t + c_y \end{cases} \quad (5.11)$$

where $v_{\max}$ is the maximum speed of the robot, $A_{\max a}$ and $A_{\max d}$ are the maximum acceleration and maximum deceleration respectively.

From (5.10) and (5.11), it is obvious that a polynomial equation of higher degree must be solved to find the meeting point or the improved meeting point of a vertex of an accelerating polygonal obstacle with respect to the starting position of the robot. With the meeting points or improved meeting points of the vertices of an obstacle obtained, the accessibility polygon or the improved accessibility polygon can be formed and the algorithms developed in the above sections can be used to search for the minimum time motion.

Another related problem of moving obstacles with an acceleration is that there is a bound on the maximum speed of the obstacles, that is, the obstacle starts moving with zero speed, accelerates to its maximum speed and then coasts at its maximum speed. Equations (5.9), (5.10) and (5.11) must be revised to deal with such motion planning problems. Before the obstacle reaches its maximum speed, it is an accelerating obstacle whose position in the global coordinates is constrained by (5.9). After it reaches its maximum speed, it becomes a non-accelerating obstacle. To solve this problem, the movement of such an obstacle can be divided into two movements: the accelerating movement and the coasting movement. A combination of the approach developed for the environment with piecewise linear motion obstacles and the approach developed in this section to solve the accelerating obstacle problem can be used to solve this problem.

5.5.6 Motion Planning in Environments with Rotating Obstacles

If a polygonal obstacle follows a circular trajectory, it is called a rotating obstacle. In this case, different points on the obstacle may move in different speeds. It is also assumed that the mobile robot can still move faster than the fastest moving point on the obstacle. Suppose that a point A which is a point on the rotating obstacle is initially at $A(x_0, y_0)$ and the centre of rotation is $C(x_c, y_c)$ as shown in Fig. 5.16. The meeting point $X(x, y)$ with respect to the starting point $S(0, 0)$ and velocity v can be computed by solving the following equations.

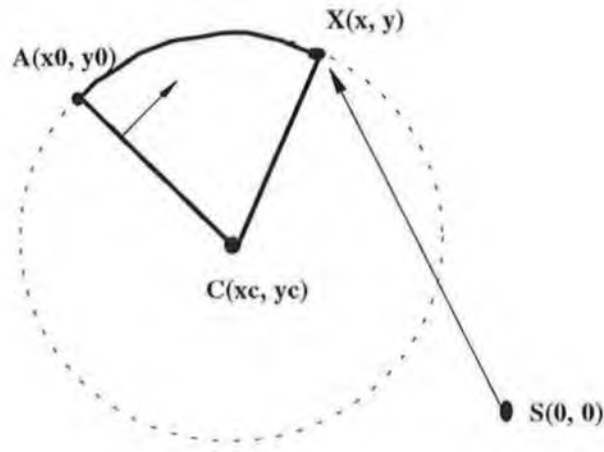


Fig. 5.16: A point in circular trajectory and its meeting point

$$\begin{cases} \frac{\sqrt{x^2 + y^2}}{v} = \frac{\theta}{\omega} \\ \sqrt{(x_0 - x_c)^2 + (y_0 - y_c)^2} = \sqrt{(x - x_c)^2 + (y - y_c)^2} \\ \theta = \cos^{-1} \left(\frac{(x_0 - x_c) \cdot (x - x_c) + (y_0 - y_c) \cdot (y - y_c)}{\sqrt{(x_0 - x_c)^2 + (y_0 - y_c)^2} \cdot \sqrt{(x - x_c)^2 + (y - y_c)^2}} \right) \end{cases} \quad (5.12)$$

where ω is the angular velocity of $A(x_0, y_0)$, θ is the angle between line CA and line CX in Fig. 5.16.

If the mass of the robot is taken into consideration, the improved meeting point of $A(x_0, y_0)$ can be computed by

$$\begin{cases} \frac{\sqrt{x^2 + y^2}}{v_{\max}} + \frac{v_{\max}}{2A_{\max a}} + \frac{v_{\max}}{2A_{\max d}} = \frac{\theta}{\omega} \\ \sqrt{(x_0 - x_c)^2 + (y_0 - y_c)^2} = \sqrt{(x - x_c)^2 + (y - y_c)^2} \\ \theta = \cos^{-1} \left(\frac{(x_0 - x_c) \cdot (x - x_c) + (y_0 - y_c) \cdot (y - y_c)}{\sqrt{(x_0 - x_c)^2 + (y_0 - y_c)^2} \cdot \sqrt{(x - x_c)^2 + (y - y_c)^2}} \right) \end{cases} \quad (5.13)$$

where $v_{\max}$ is the maximum velocity of the robot, and $A_{\max a}$ and $A_{\max d}$ are the maximum acceleration and maximum deceleration of the robot respectively.

The accessibility polygon and the improved accessibility polygon of a polygonal obstacle which moves in a circular trajectory can be computed by (5.12) and (5.13). Then the accessibility graph algorithm and the improved accessibility graph algorithm can be used to generate the minimum time motion for the robot. The most difficult step in this case, however, is to solve (5.12) and (5.13). A numerical approach must be used to generate the solution.

5.5.7 Motion Planning in Environments with Transient Obstacles

Most of the prior approaches to various motion planning problems assume that the obstacles always exist in the environment, whether they are stationary or in motion. In this section, motion planning in an environment which contains obstacles whose existence is dependent on time will be studied, that is, they may disappear and reappear in the environment. Such an obstacle is called a transient obstacle.

An environment with transient obstacles can be used to model a number of time varying situations which can arise in many application areas.

1. A traffic light is such a dynamic obstacle on the road.
2. For robot navigation, an area that the robot is not allowed to pass through for a certain period of time is a transient obstacle.
3. For a mobile robot which must work on an aircraft carrier or in an airport, aircraft that land and take off may be modelled as appearing and disappearing obstacles.

4. In certain situations, some other robot picks up one of the objects in the workspace and puts it back at some other location in the same environment. For a mobile robot in that workspace, these objects may be modelled as disappearing from the scene and appearing again in the environment. Such a situation may arise when two or more than two robots operate simultaneously in the same workspace.
5. A fast moving obstacle can be approximated by a series of transient obstacles. This makes it possible to give a near time minimal solution amidst fast and slowly moving obstacles in polynomial time.

The problem of motion planning in environments which contains transient obstacles can be stated as follows: There are a set of polygonal obstacles, starting point and a destination point in the environment. Each obstacle is a polygon which exists in the scene for a certain interval of time $[t_1, t_2]$, where $t_1 < t_2$. For time interval $[t_1, t_2]$, the obstacle is either fixed at a certain location or moving in the environment. For ease of explanation, the obstacles are classified into the following four types, depending on t_1 and t_2 , which are shown in Table 5.1.

Type	Name	Definition
1	Disappearing Obstacle	The obstacle exists in the scene for an interval of time $[t_1, t_2]$, where $t_1=0$ and $t_2=t_{\text{disappear}} < \infty$.
2	Appearing Obstacle	The obstacle exists in the scene for an interval of time $[t_1, t_2]$, where $t_1 > 0$ and $t_2 = \infty$.
3	Temporary Obstacle	The obstacle exists in the scene for an interval of time $[t_1, t_2]$, where $t_1 > 0$ and $t_2 < \infty$.
4	Permanent Obstacle	The obstacle exists in the scene for an interval of time $[t_1, t_2]$, where $t_1=0$ and $t_2 = \infty$.

Table 5.1: Classification of obstacles

When an obstacle is moved around in the environment by pick-up and put-down operations, it may appear and disappear in the scene many times. An obstacle that exists in the scene during k intervals of time (either at the same location or different locations in the environment) is treated as k different obstacles in the approach introduced below.

The approach to motion planning in an environment with transient obstacles, which is proposed in this section, is similar to the approach used in environments with piecewise linear motion obstacles. The accessibility polygon or the improved accessibility polygon of a polygonal obstacle is first calculated and formed without considering if the obstacle is disappearing or not.

For an appearing obstacle, it is considered as an ordinary obstacle except that the time is relative to the instance when the that it appears. When the meeting point or the improved meeting point of a vertex is calculated, it must be calculated relative to the time when the obstacle appears, and that time must be transformed to coincide with the starting time of the robot.

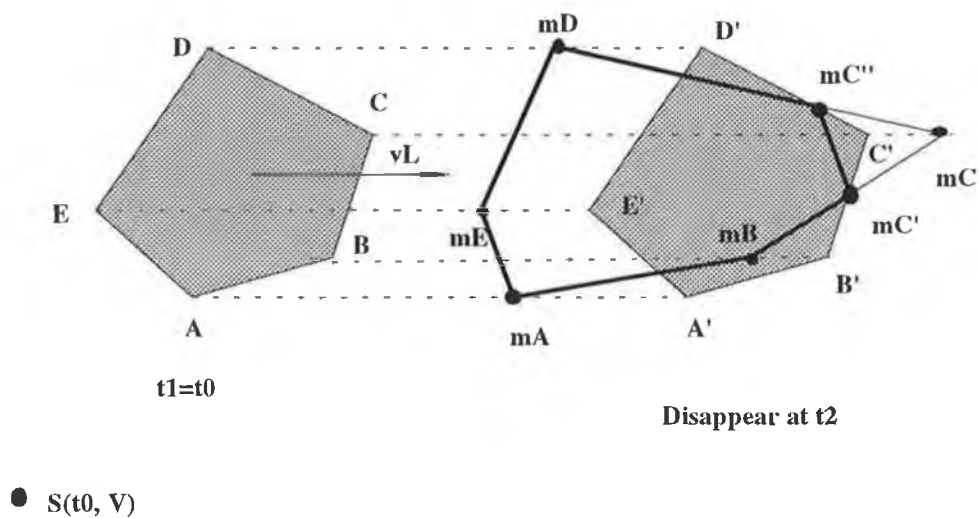


Fig. 5.17: Accessibility polygon of a disappearing obstacle

For a disappearing obstacle, if the meeting time or the improved meeting time of a vertex of the accessibility polygon or improved accessibility polygon is longer than

the time when it disappears, that vertex must be modified to the point on the collision front whose meeting time is the same as the disappearing time. This can be shown in Fig. 5.17. Polygon ABCDE starts moving at time $t_1=t_0$ with velocity v_L in the direction shown in the Fig. When it reaches A'B'C'D'E' at time t_2 , it disappears. Polygon mAmBmCmDmE is the accessibility polygon with respect to S without considering the disappearance of the obstacle. Because the accessible time of any point on the collision front mCmC' and mCmC'' is greater than the disappearing time t_2 (the collision front is in fact a curve, a line segment is used to simplify the discussion, see Fig. 5.15). So the accessibility polygon of the disappearing obstacle ABCDE is mAmBmC'mC''mDmE which is shown in Fig. 5.17 with thick line segments.

If the mass of the robot is taken into consideration, the improved accessibility polygon of the disappearing obstacle can be formed in a similar manner. The improved accessible time of a point on the collision front which is greater than the disappearing time will be deleted from the improved accessibility polygon. The point on the collision front whose improved accessible time is equal to the disappearing time will be a vertex of the improved accessibility polygon.

With the accessibility polygon or improved accessibility polygon of a disappearing polygonal obstacle or an appearing obstacle formed in the above manner, the accessibility graph algorithm or the improved accessibility graph algorithm can be used to search among the accessibility polygons or the improved accessibility polygons of the polygonal obstacles in the environment to generate the minimum time motion for the robot.

5.6 Summary and Discussion

The accessibility graph algorithm is effective for motion planning of a massless mobile robot in dynamic environments. When the mass of the robot is taken into consideration, that is, it takes time for the robot to accelerate from zero speed to its

maximum speed, the approaches used heretofore are based on space time or state time representation. For two dimensional dynamic environments, the two dimensional space problem becomes a three dimensional space problem. The memory required for space time representation has been proven to be of exponential order [Fujimura, K. 1989] [Fujimura, K. 1992]. The improved accessibility graph algorithm based on the concepts of improved accessibility polygon and improved accessibility graph developed in this chapter solves the motion planning problem in dynamic environments for a mobile robot with mass and the computation time needed is almost the same as for the accessibility graph. This new algorithm can also be extended to deal with some special cases which are discussed in the above sections. Further more, it can be extended to solve motion planning problem in three dimensional dynamic environments. This is a great improvement to the space time approach not only in computation time, but also in application extensions.

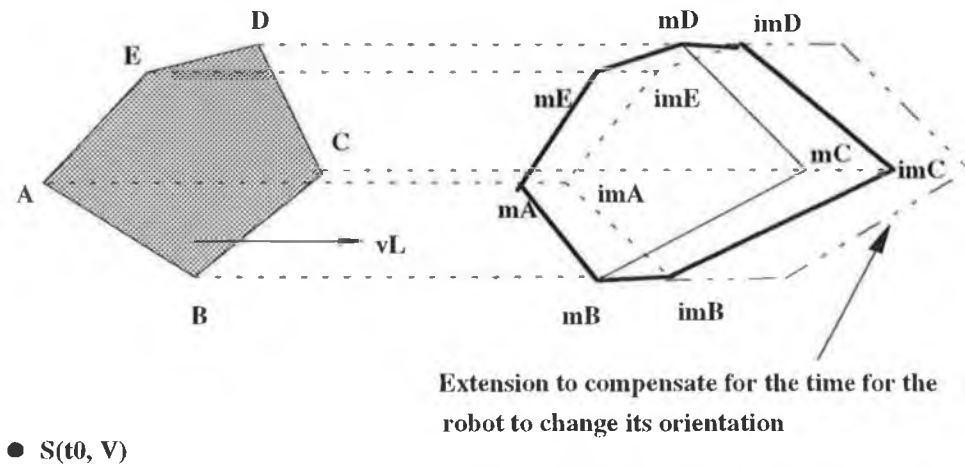


Fig. 5.18: Accessibility polygon and improved accessibility polygon

The computation process of the improved accessibility graph algorithm is almost the same as that of the accessibility graph algorithm. In fact, the improved accessibility polygon just expands the region of the accessibility polygon which is the forbidden region for the robot when planning motion for massless robots. The purpose of expanding the region of the accessibility polygon is to give the robot time to accelerate and decelerate. This can be shown in Fig. 5.18. The accessibility polygon of moving polygon ABCDE with respect to S is $mAmBmCmDmE$ and the polygon

which is formed by linking all the improved meeting points of vertices of ABCDE with line segments is $imAimBimCimDimE$. The improved accessibility polygon of ABCDE with respect to S is the polygon with minimum area which contains these two polygons, that is, $mAmBimBimCimDmDmE$ which is shown Fig. 5.18 with thick line segments.

In the development of the improved accessibility graph algorithm, it is assumed that when the robot is about to turn around a corner, the robot must decelerate to a certain speed which is governed by (4.5) in chapter 4. This assumption is practical for real applications. However, it is also assumed that it takes no time for the robot to change its orientation. This assumption is not practical. If the time for the robot to change its orientation when it reaches a corner is taken into consideration (which is governed by (4.4) in chapter 4), the improved accessibility polygon must be extended in the direction of the motion of the obstacle as shown in Fig. 5.12 to compensate for the time delay due to this change. The compensating time item is the time given by (4.4) when θ is equal to π . This compensation is also shown in Fig. 5.18. With the accessibility polygon revised, the improved accessibility graph algorithm can be used to search among all the improved accessibility polygons to generate the minimum time motion for the robot which takes time to accelerate from zero speed to its maximum speed and takes time to change its orientation when it turns around a corner. The only change is that the time item for the robot to change its orientation which is given by (4.4) must be added to the cost function when the search is performed.

The improved accessibility graph algorithm is easily extended to special applications. What one has to do is to compute and form the accessibility polygons or the improved accessibility polygons of the polygonal obstacles for these special cases. For example, the approach to motion planning problem for environments with transient obstacles developed by K. Fujimura [Fujimura, K. 1992] is constrained by the assumption that the transient obstacles can not move during its existing interval. The approach which is an extension of the improved accessibility graph algorithm proposed in the above section is not constrained by that assumption. For fixed

transient obstacles during their existence period, the accessibility polygon or improved accessibility polygon of the obstacle is the obstacle itself.

Another advantage of the improved accessibility graph algorithm is that unlike the space time approach which converts a two dimensional space problem into a three dimensional space problem, it tries to solve the two dimensional space problem in two dimensional space. It is easy to extend it to solve the similar three dimensional problem. For three dimensional environments, the space time approach has to solve a four dimensional space problem. It will be so abstract that it is difficult to express in geometrical space. The extension of the improved accessibility graph algorithm to three dimensional environments will be discussed later in chapter 6.

It is obvious that heuristics play an important role in accelerating the search processes in chapter 3 and chapter 4. It is also possible to introduce heuristics to the accessibility graph algorithm and improved accessibility graph algorithm to accelerate the search process. The only constraint on the heuristics is that the estimation of the time for the robot to travel from the current standing point to the destination point must be less than the actual time that the robot requires.

Chapter

6. Motion Planning in Three Dimensional Environments

6.1 Introduction

Motion planning in two dimensional static and dynamic environments has been studied in chapters 3, 4 and 5. This chapter will focus on the study of motion planning problems in three dimensional environments. First, the obstacles in a three dimensional environment and the configuration space of the robot in three dimensional space are first defined. Then, the shortest path in a three dimensional environment is discussed. The three important properties of a shortest path in a three dimensional environment are described and approaches to solve the shortest path planning problem based on these three properties are introduced. The visibility graph algorithm for the shortest path which is studied in chapter 3 will be extended to solve the three dimensional environment shortest path problem.

Next, the minimum time motion planning problem in a three dynamic environment will be studied. There are also three important properties for a minimum time motion in a three dimensional dynamic environment, which are similar to the properties of the shortest path. The collision front of a face (a plane) is defined and is proven to be in the form of either a hyperboloid, a cone or a plane. The concept of the accessibility polyhedron is proposed based on the approximation of a curved collision front by several faces. Based on the concept of the accessibility polyhedron, the three dimensional accessibility graph is defined and an algorithm which is similar to the two dimensional accessibility graph algorithm (see chapter 5) is proposed to generate the minimum time motion for a mobile robot in a three dimensional dynamic environment.

Finally, the concepts of the improved collision front of a face and the improved accessibility polyhedron are defined. An algorithm which can generate the minimum time motion for a mobile robot with mass in a three dimensional dynamic environment is proposed based on the concept of the improved accessibility polyhedron.

6.2 Obstacles in a Three Dimensional Environment

6.2.1 Definition of Polyhedra

A plane P_l in three dimensional Euclidean space R^3 decomposes the three dimensional space into two half spaces. Let $h(x, y, z)=0$ be the equation of P_l . The two closed half spaces determined by $h(x, y, z) \leq 0$ and $h(x, y, z) \geq 0$ are denoted by h^- and h^+ , respectively.

Definition: A convex polyhedral region CPR in R^3 is the intersection of a finite number of closed half spaces. A polyhedral region PR is any subset of

$\mathbb{R}^3$ obtained by taking the union of a finite number of convex polyhedral regions.

A polyhedral region defined above may not be bounded, nor connected. A polyhedron is bounded and simply connected. It is defined as follows.

Definition: A polyhedron is any polyhedral region that is homeomorphic to the closed unit ball in $\mathbb{R}^3$. A convex polyhedron is any convex polyhedral region that is homeomorphic to the closed unit ball in $\mathbb{R}^3$.

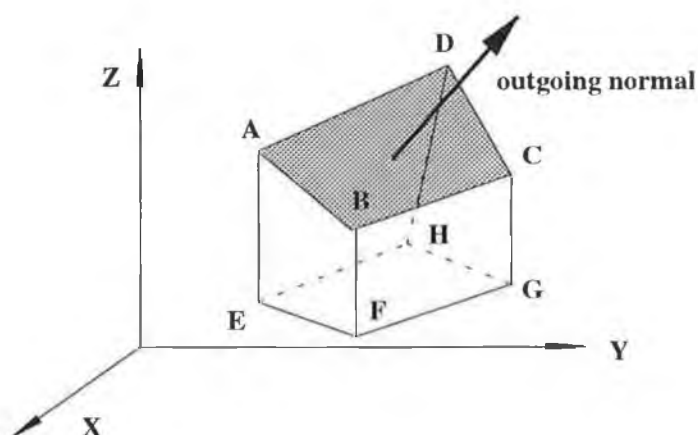


Fig. 6.1: A polyhedron

Let P be a polyhedron. Each maximal connected planar region in P 's boundary that is not intersected by the rest of P 's boundary except possibly along its own boundary, is called a face. Each face is a polygonal region. Its edges are called edges of P . Every endpoint of an edge is called a vertex of P . It is obvious that each edge of a polyhedron is shared by exactly two faces. Fig. 6.1 is an example of a polyhedron, where the polygon region $ABCD$ is a face of the polyhedron $ABCDEFGH$, line segment AB is one of its edges.

Every face F_i of P is contained in a plane Pl_i called the supporting plane of F_i . As shown in Fig. 6.1, face $ABCD$'s supporting plane's equation can be given by

$$\begin{vmatrix} x - x_A & y - y_A & z - z_A \\ x_B - x_A & y_B - y_A & z_B - z_A \\ x_C - x_A & y_C - y_A & z_C - z_A \end{vmatrix} = 0 \quad (6.1)$$

where (x_A, y_A, z_A) , (x_B, y_B, z_B) and (x_C, y_C, z_C) are the coordinates of A, B and C respectively.

For every point on the open face (the polygonal region which does not include the edges is called an open face) of a polyhedron, there is a neighbourhood whose intersection with the polyhedron only contains points of the polyhedron. These points must correspond to $h(x, y, z) \leq 0$ or $h(x, y, z) \geq 0$. The normal of that face which points to the direction of these points is called the ingoing normal of the face. The normal of that face which points to the opposite direction of the ingoing normal is called the outgoing normal. The outgoing normal of face ABCD is shown in Fig. 6.1.

To simplify the problem, an obstacle in a three dimensional environment is defined as a polyhedron.

6.2.2 Obstacles in Configuration Space

As pointed out in chapter 3, the configuration space can be interpreted as a rigid body transformation which preserves distance and orientation. A rigid body in a three dimensional Euclidean space can be denoted by six parameters, the position co-ordinates (x, y, z) of the origin of the co-ordinate system which is fixed with it in the global co-ordinate system, and the orientation of its co-ordinate system in the global co-ordinate system. So the configuration space of a rigid body in a three dimensional Euclidean space is a six dimensional Euclidean space.

If the mobile robot in a three dimensional environment is considered as a polyhedron, the transformation of the polyhedral obstacles into the robot's configuration space is

very complicated [Latombe, J. 1990]. To simplify the problem, in this chapter, the robot is considered as a ball with radius R . The orientation of the robot will not be taken into consideration. The robot is considered as point size and the polyhedral obstacles are expanded by the size R . The expansion of a polyhedron can be realised by moving the supporting plane of every face in its outgoing direction by the distance R . The intersection of the half spaces of these new planes after moving will form another polyhedron. This polyhedron is the obstacle of the original obstacle in the robot's configuration space. Fig. 6.2 is an example of this expansion, the original obstacle is ABCDEFGH, the obstacle in configuration space is KMNOPQRS.

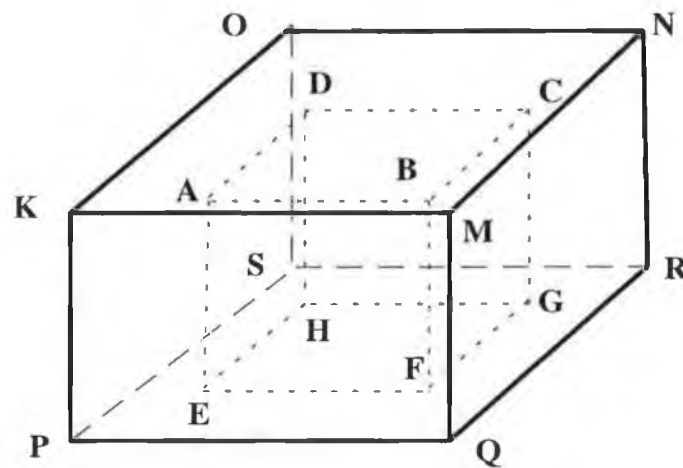


Fig. 6.2: Expansion of a polyhedron

6.3 Shortest Paths in a Three Dimensional Static Environment

6.3.1 Properties of a Shortest Path

In the case of stationary polyhedral obstacles in a three dimensional environment, a shortest path which avoids passing through the interiors of the obstacles is known to have the following three properties [Sharir, M. 1986] [Akman, V. 1987].

1. A shortest path L from a starting point S to a destination point D in a three dimensional environment is a serial chain of line segments that bends at internal points of edges or at non-convex vertices of the polyhedral obstacles. In other words, L can be represented as a sequence of one or more line segments $L = (L_1, L_2, \dots, L_p)$ such that each L_i is a line segment $P_i P_{i+1}$, where $P_0 = S$ and $P_{p+1} = D$. Each P_i ($i=1, 2, \dots, p$) is either an internal point of an obstacle edge or a non-convex vertex of an obstacle.
2. Suppose that a shortest path L passes through an internal point of an obstacle edge e . The path L enters and leaves e at equal angles.
3. If L is a shortest path from S to D which passes through interior points of a given sequence of edges $E = (e_1, e_2, \dots, e_p)$, then L is the unique shortest path from S to D which is constrained to pass through E in this order.

Unlike the two dimensional environment case, shortest paths in a three dimensional environment bend also at interior points of edges of the polyhedral obstacles rather than just vertices. When all polyhedra are convex, shortest paths never bend at vertices of the obstacles. Using the above three properties, the problem of finding a shortest path amidst polyhedral obstacles can be solved [Sharir, M. 1986]. The problem is decomposed into two subproblems: (1) given a sequence of edges, find a shortest path that is constrained to pass through the edge sequence, and (2)

determine the sequence of edges and non-convex vertices through which a shortest path passes.

Both subproblems (1) and (2) are known to be computationally hard to solve. The fastest algorithms known so far solve subproblem (1) in singly-exponential time [Reif, J. 1985] [Canny, J. 1988a] and subproblem (2) is known to be NP-hard [Canny, J. 1988a]. Subproblem (2) can be simply solved by examining all possible permutations of sequences of edges and vertices, and compute the length of a shortest path constrained to pass through each of these sequences. For subproblem (1), two methods (numerical computation and algebraic computation) are proposed to obtain a shortest path by making use of the above three properties [Sharir, M. 1986].

The numerical method for subproblem (1) begins with an arbitrary polygonal path passing through a given sequence of edges. Then, the path is interactively shortened by picking one of the contact points that does not satisfy the second property regarding equal entry and exit angles and replacing it by another point on the same edge that satisfies the property. Since the path is strictly shortened after each iteration, the path will converge to the desired shortest path by this process. However, it is not known in analytical form how fast the path converges to the desired shortest path, although in practice the process is observed to be extremely fast [Fujimura, K. 1992].

The algebraic computation gives an exact shortest path as a solution of a system of n equations with respect to n contact points at the given edges. M. Sharir and A. Schorr show that it takes doubly-exponential time to solve the system by elimination in the number of edges of the input polyhedra in the environment [Sharir, M. 1986] [Fujimura, K. 1992]. This complexity has been improved to a singly-exponential time by using more efficient reduction of the system [Reif, J. 1985] [Canny, J. 1988a] [Fujimura, K. 1992].

For the case that the number of polyhedra in the environment is a fixed number, a shortest path can be computed in a polynomial time in the number of total edges of

the polygons and exponential in the number of polyhedral obstacles [Sharir, M. 1987]. L. P. Gewali, S. Ntafos and I. G. Tollis consider restricted cases of the problem in which polyhedral obstacles are all vertical (a vertical polyhedral obstacle is a polyhedron whose faces are either vertical to or parallel to a certain plane). They show that an algorithm which runs in $O(n^{6k-1})$, where k and n represent the numbers of vertical obstacles and vertices of the obstacles, respectively [Gewali, L. 1990]. For the special case that the number of polyhedra is one, D. M. Mount shows an algorithm that runs in $O(n^2 \cdot \log(n))$ time to find a shortest path between two points on the surface of the polyhedron [Fujimura, K. 1992].

6.3.2 Three Dimensional Visibility Graph

The two dimensional visibility graph which is discussed in chapter 3 can be searched to generate the shortest path. However, if the visibility graph whose nodes are the vertices of the obstacles in a three dimensional environment is constructed and searched in a similar way, it does not necessarily generate the shortest path because the properties of the shortest path in a three dimensional environment only guarantees that the shortest path is a polygonal line whose vertices are contained in edges or non-convex vertices. However, fictitious vertices on the edges of a polyhedral obstacle can be added to the visibility graph to generate an approximate shortest path. The fictitious vertices can be added in such a way that no resulting edge is longer than a prespecified maximum length. This is shown in Fig. 6.3, where there is only one polyhedral obstacle in the environment. The nodes of the visibility graph are the vertices of the polyhedral obstacle plus the fictitious vertices on the edges of the obstacle, only part of the edges of the visibility graph is shown in Fig. 6.3.

With the three dimensional visibility graph defined above, the graph search algorithm which is similar to the algorithm proposed in chapter 3 can be used to search the visibility graph and generate an approximate shortest path. The thick polygonal line segments in Fig. 6.3 represent this approximate shortest path from the starting point S to the destination point D .

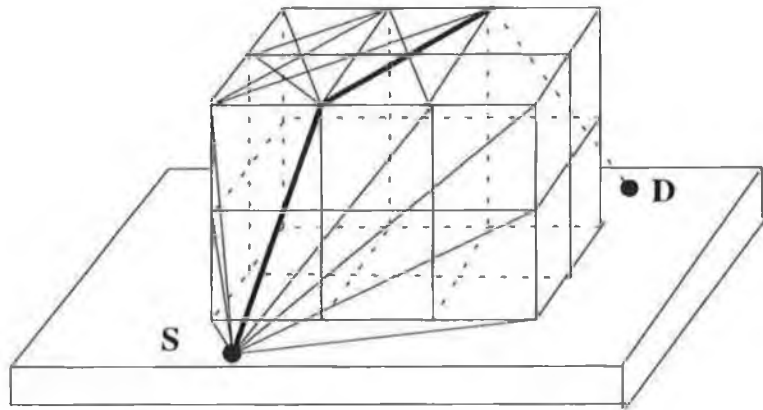


Fig. 6.3: Three dimensional visibility graph and approximate shortest path

Little is known about the minimum time path in a three dimensional static environment. Does the minimum time path in a three dimensional static environment defined by the system motion equation (4.1) and constraints (4.2)-(4.6) have the above three properties of the shortest path? With the three dimensional visibility graph constructed above, it is not necessary to know if the minimum time path has these three properties or not. The Minimum Time Path at Visibility Nodes (MTPVN) can be defined in a similar manner as in chapter 4. A similar approach can also be used to search the three dimensional visibility graph to generate the approximate minimum time path.

6.4 Motion Planning in a Three Dimensional Dynamic Environment

Regarding motion planning in the presence a set of moving polyhedral obstacles in a three dimensional environment, relatively little is known. J. Reif and M. Sharir have establish that the problem is PSPACE-hard in the presence of rotating obstacles.

They also show a decision algorithm that runs in $O(2^{n^{O(1)}})$ time and in $n^{O(n \cdot \log(n))}$ space. They observe that just avoiding slowly moving obstacles is possible [Reif, J. 1985] [Fujimura, K. 1992]. K. Fujimura has proven that the minimum time motion in

the presence of slowly moving polyhedral obstacles in a three dimensional environment also has three properties similar to the properties of the shortest path. Based on these three properties, an approach which is similar to the shortest path approach of obtaining the solution of a system of algebraic equations is also proposed [Fujimura, K. 1992].

6.4.1 Properties of a Minimum Time Motion

A minimum time motion among slowly moving polyhedral obstacles in a three dynamic environment exhibits the following three properties that are analogous to the above three properties for shortest paths [Fujimura, K. 1990]. They can be described as follows.

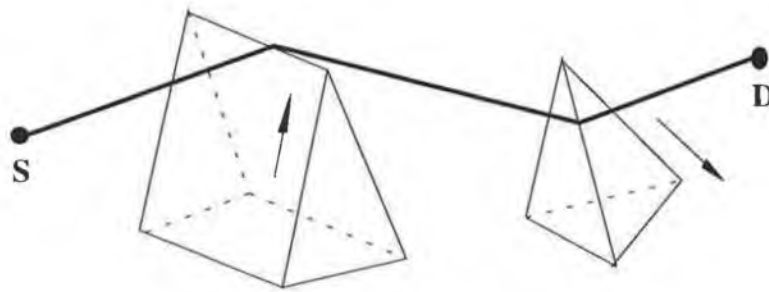


Fig. 6.4: A polygonal path

1. A minimum time motion L from a starting point S to a destination point D consists of a serial chain of straight motion segments which bends either at internal points on obstacle edges or vertices of the polyhedra (see Fig. 6.4). Motion L is traversed at the maximum speed of the point robot through out L from S to D . In other words, L can be represented as a sequence of one or more motion segments $L=(L_1, L_2, \dots, L_p)$ such that each L_i is a straight line motion from P_i to P_{i+1} traversed at the robot's top speed, where, $P_0=S$ and $P_{p+1}=D$. For each $i=1, 2, \dots, p$, P_i is coincident either with an internal point on an obstacle edge or a vertex of an obstacle.

2. Suppose that a minimum time motion L passes through an internal point of an edge e_i . The motion L enters and leaves the boundary segment of the collision front generated by e_i at equal angles.
3. If $L=(L_1, L_2, \dots, L_p)$ is a minimum time motion from S to D starting at S at time t_0 that passes through interior points of a given sequence of moving edges $e=(e_1, e_2, \dots, e_p)$, then L is the unique minimum time motion from S to D starting at S at time t_0 that is constrained to pass through e in this order.

The concept of collision front in property 2 is a generalisation of that of two dimensions to three dimensions and will be defined below. In chapter 5 where the two dimensional dynamic environment is considered, the motion of a line segment is defined as a movement and an obstacle in the environment is defined as a sequence of movements. In a similar manner, the motion of a face can be defined as a movement and a polyhedral obstacle in a three dimensional dynamic environment is defined as a sequence of movements. A face movement can be denoted as a tuple (F, d_F, v_F) which represents a motion of a face F moving in the direction d_F at speed v_F . Because a face of a polyhedron is a polygonal region, the edges of the face are line segments. An edge movement is defined for an edge by a tuple (E, d_E, v_E) in a similar manner. The concepts of accessibility and collision front in two dimensional environments can be extended to three dimensional environments.

Accessibility (3D): Consider a set of face movements $M=\{M_1, M_2, \dots, M_n\}$ and D , the destination point. Let R be a point robot located initially at S at time t_0 . Suppose that R starts moving at time t_0 at a speed v . After R starts moving, it will move in a fixed direction at constant speed v . A point V (V is either the destination point or a point on a face $M_j \in M$) is said to be accessible at X from S , if there exists a direction of the motion of R such that R meets V without being intercepted by any other movement. The meeting point X is called an accessible point of V and the meeting time is called the accessible time of X and denoted by $t(X)$.

Collision Front (3D): Consider an environment that contains only one face movement (F, d_F, v_F) . Let S be the starting point of the robot at time t_0 . The set of accessible points from S corresponding to all points in F forms a curved surface. This surface is called a collision front due to F generated about S (with respect to the robot's starting time t_0 and speed v). When there is more than one face in the environment, there will be more than one collision front and it is possible that only some portion of the faces is accessible.

Without loss of generality, a face F which is always parallel to the x - y plane of the co-ordinate as it moves, is considered. Let v_F and v be the speeds of F and the robot respectively. Let S denote the starting location of the robot. Let Pl be the plane which moves vertically (in the direction of the z axis) at the speed equal to the z -component of v_F . Let Pl coincide with $z=a$ ($a \geq 0$) at the starting time.

Suppose that a point $P(x, y, a)$ on Pl at the starting time (time 0) is accessible from the starting position, the origin $(0, 0)$, at point $A(x, y, z)$. Refer to Fig. 6.5, x , y and z must satisfy

$$t(A) = \frac{z - a}{u} = \frac{\sqrt{x^2 + y^2 + z^2}}{v} \quad (6.2)$$

where u is the z component of the velocity of face v_F .

If a is not equal to zero and v is bigger than v_F so is bigger than u , the collision front of plane Pl defined by (6.2) is one side of a hyperboloid and this is shown in Fig. 6.5. It is easy to see that the collision front due to F is the intersection of the hyperboloid and the trajectory of F , thus the collision front of F is a portion of the hyperboloid (see Fig. 6.5).

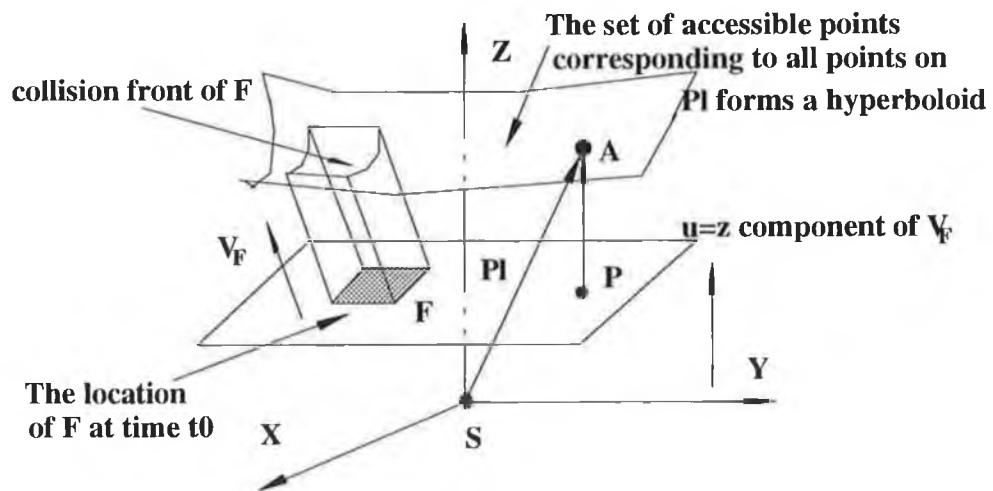


Fig. 6.5: The case of the collision front of a face being a hyperboloid

If a is equal to zero in (6.2), that is, the starting point S is on the plane Pl , (6.2) can be rewritten as

$$z = \frac{u}{\sqrt{v^2 - u^2}} \cdot \sqrt{x^2 + y^2} \quad (6.3)$$

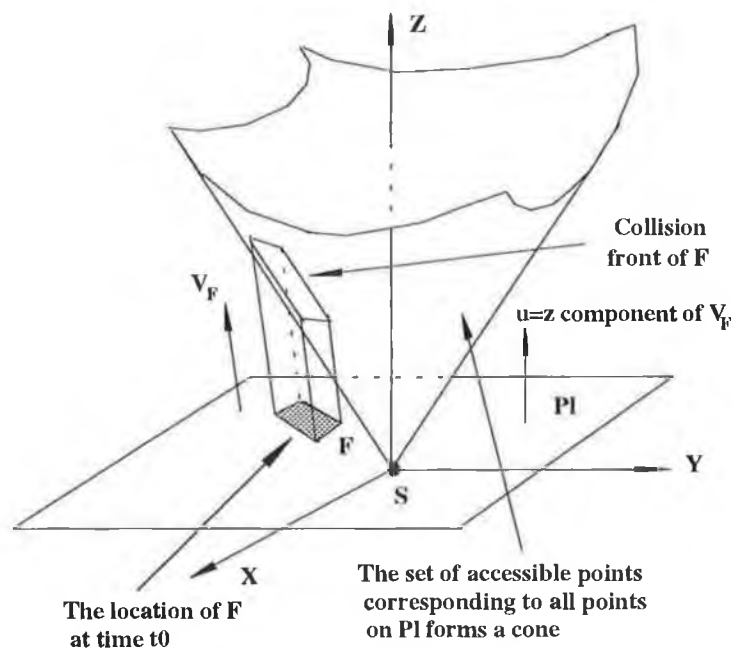


Fig. 6.6: The case of the collision front of a face being a cone

It is easy to see from (6.3) that the collision front of Pl is a cone. In this case the collision front due to F is the intersection of the cone and the trajectory of F , thus the collision front of F is a portion of the cone (see Fig. 6.6).

Another case is the degenerate case, in which the z component of v_F , u is equal to zero. The collision front of F in this case is a subset of the plane Pl (see Fig. 6.7).

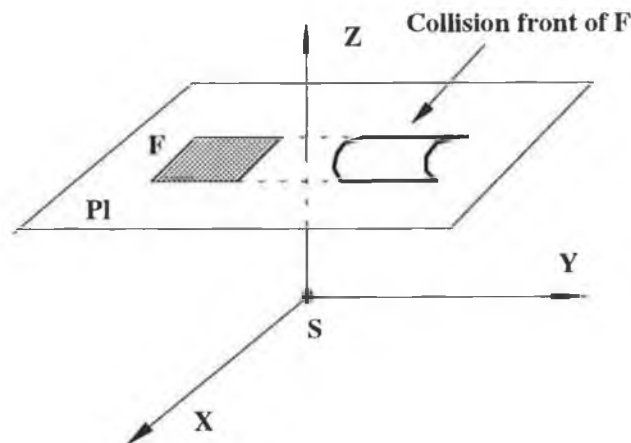


Fig. 6.7: The case of the collision front of a face being part of a plane

With the collision front of a moving face defined and analysed above, it is obvious the collision front generated by an edge is also a conic curve segment. The second property of a minimum time motion can be illustrated in Fig. 6.8 where the enter angle is equal to the exit angle.

The above three properties of a minimum time motion in a three dimensional dynamic environment are proved in [Fujimura, K. 1992]. Making use of the three properties, a minimum time motion can be determined in the presence of a set of slowly moving polyhedral obstacles. As in the case of stationary obstacles, an exact solution can be obtained as a solution of a system of algebraic equations with n variables with respect to the positions of contact points at edges. Another method is the numerical method. Given an edge sequence that a minimum time motion passes through, the positions of contact points at the edges can be obtained numerically as follows: First find an

arbitrary motion that passes through the given edge sequence. The arrival time is interactively shortened by choosing one of the contact points, P_i , that does not satisfy property 2 with respect to incoming and outgoing angles and replacing it by another point on the same boundary of the collision front where property 2 holds. As a result, the outgoing motion segment emanating from the new contact point arrives at the contact point on the next edge earlier than before. This effect can be propagated for the rest of the motion, resulting in a motion that arrives at the destination point earlier than the original motion. Since this process shortens the arrival time of the motion, the motion converges to the desired minimum time motion by repeatedly applying the process [Fujimura, K. 1992].

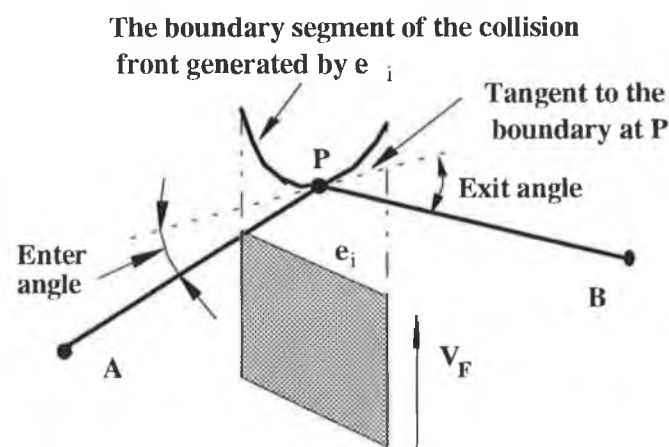


Fig. 6.8: Equal incoming and outgoing angles

6.4.2 Three Dimensional Accessibility Graph

In chapter 5, the time varying graph (the accessibility graph) plays an important role in the minimum time motion planning in a two dimensional dynamic environment. With a line segment replacing a curved collision front, the concept of the accessibility polygon is defined. The definition of the accessibility polygon makes the construction of the accessibility graph very similar to the construction of the visibility graph. Can a similar method be used to solve the motion planning problem in a three dimensional dynamic environment ?

In a two dimensional environment, a line segment which links the starting point and the ending point of a curve collision front segment can be used to approximately replace the collision front and the accessibility polygon of a moving polygonal obstacle is formed by doing this. However, in a three dimensional environment, it is difficult to use a face to approximate a subset of a hyperboloid or a cone (the collision front of a moving face). But several faces can be used to approximate the collision front of a face and the concept of the accessibility polyhedron of a moving polyhedral obstacle can be derived by this approximation.

By definition, the boundary of a polyhedron in a three dimensional Euclidean space is composed by a finite number of limited faces. Every face is a polygonal region, so the edges of the face are some line segments. The collision front of every edge of a face is a conic curve segment (see chapter 5). If a line segment is used to replace every conic curve segment, a three dimensional polygon is formed. The number of the edges of that three dimensional polygon is equal to the number of edges of the face. This is shown in Fig. 6.9, where ABCD is the moving face, the accessible points of A, B, C and D are mA, mB, mC and mD respectively. Line segment mAmB is used to replace conic curve mAmB, so are mBmC, mCmD and mDmA. A three dimensional polygon mAmBmCmD is formed and the vertices of this polygon are usually not in one plane, so it is called a three dimensional polygon.

In a three dimensional Euclidean space, two intersection lines can form a plane. As shown in Fig. 6.9, line segments mAmB and mBmC can form the plane P11; line segments mCmD and mDmA can form another plane P12. The intersection of these two planes will form a line and the line segment mAmC is a subset of this line. Line mAmC separates plane P11 and P12 into two parts. The intersection of the trajectory of the moving face ABCD and these two planes, P11 and P12, will form two faces, mAmBmC and mCmDmA. These two faces can be used to approximate the collision front of the moving face ABCD (see Fig. 6.9).

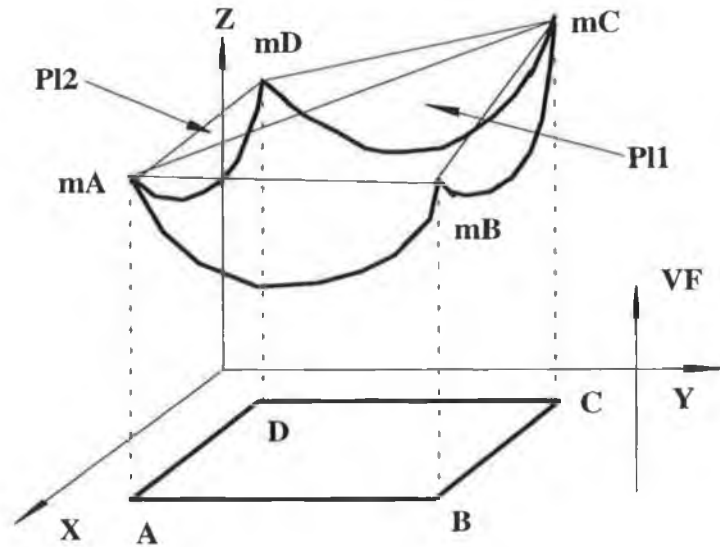


Fig. 6.9: Three dimensional collision polygon

The above discussion is for the special moving face ABCD. For a general moving face, the same process can also be applied. Every two adjacent edges of the moving face can form a plane in the above manner. If the number of edges of the moving face is even, $n/2$ faces can be used to replace the collision front of that moving face, where, n is the number of edges of the moving face. If the number of edges of the moving face is odd, the first $n-1$ collision edges can form $(n-1)/2$ faces, the last edge and the first edge can be used to form another plane, so $(n+1)/2$ faces can be used to replace the collision front of the moving face, where n is the number of edges of the moving face.

Fig. 6.10 shows one face which moves with velocity vector $v_F=20$ (the robot's velocity is 40). The co-ordinate system is selected as follows: Z axis is in the direction of v_F and pass the starting point of the robot. The origin of the co-ordinate system is the intersection point between Z axis and the supporting plane of the moving face. With the co-ordinate system selected above, the collision front of the face shown in Fig. 6.10 can be calculated by the following equation:

$$\left\{ \begin{array}{l} t = \frac{\sqrt{x^2 + y^2 + (z - A)^2}}{v} \\ x = x_0 + v_{Fx} \bullet t \\ y = y_0 + v_{Fy} \bullet t \\ z = z_0 + v_{Fz} \bullet t \end{array} \right. \quad (6.4)$$

where, v is the speed of the robot, (x_0, y_0, z_0) is the co-ordinate of a point on the moving face at the starting time (time 0), (v_{Fx}, v_{Fy}, v_{Fz}) are the components of v_F on X, Y and Z axes respectively, A is the Z component of the starting point of the robot. Fig. 6.11 shows the collision front of the moving face in Fig. 6.10. Fig. 6.12 shows the approximation of the collision front by several faces.

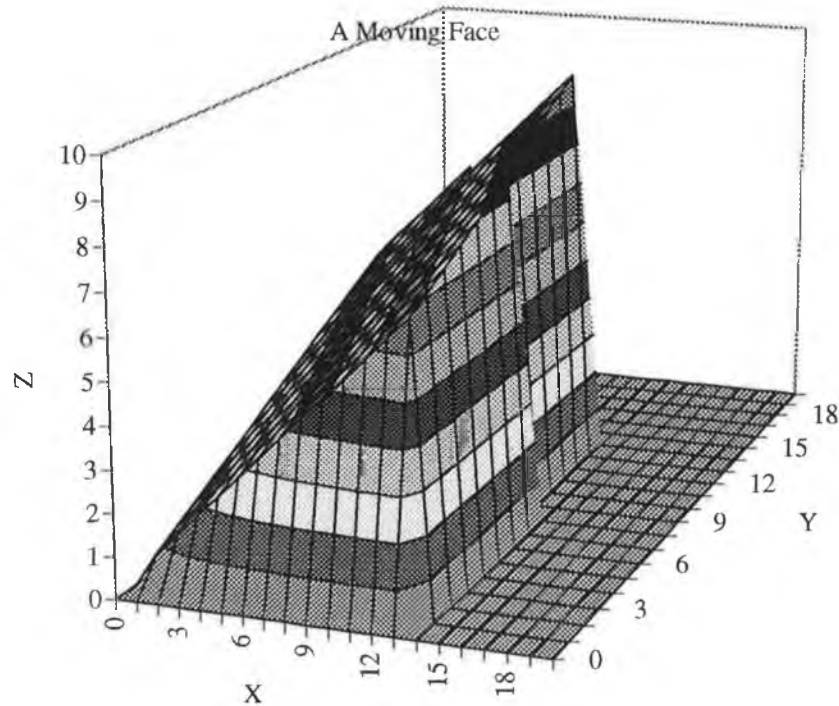


Fig. 6.10: An example of a moving face

Collision Front of a Face

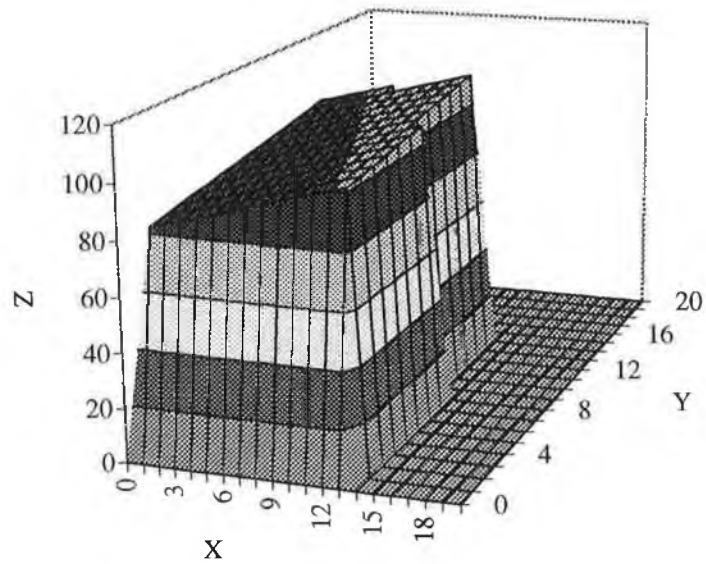


Fig. 6.11: Collision front of the face in Fig. 6.10

Approximation of a collision front by faces

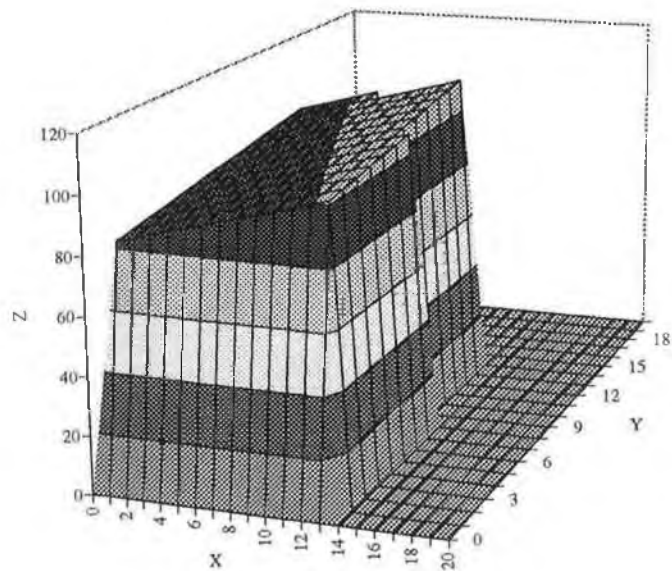


Fig. 6.12: Approximation of the collision front in Fig. 6.11

If the collision front of every face of a moving polyhedral obstacle is approximated by several faces in the above manner, the concept of the accessibility polyhedron of a moving polyhedron can be defined as follows.

Accessibility polyhedron: Suppose a polyhedron moves in direction d_L with speed v_L and the robot is initially at S at time t_0 and the maximum speed of the robot is v . Only the movement of a face of the polyhedron is considered. If v is bigger than v_L , the collision front of every face of the polyhedron can be approximated by several faces as discussed above, and these approximating faces form another polyhedron in the three dimensional Euclidean space. The polyhedron so formed is called the accessibility polyhedron of the moving polyhedron.

From the definition of the accessibility polyhedron, it is obvious that if a line segment AB does not intersect any of the accessibility polyhedra of the polyhedral obstacles in the environment with respect to A as the starting point, then the robot can move from A to B along the line segment path AB without any intersection with the obstacles. In other words, if the point B is visible from A among the accessibility polyhedra of the polyhedral obstacles in the environment with A as the starting point, then it is safe for the robot to move from A to B .

The three dimensional accessibility graph can be defined based on the concept of the accessibility polyhedron.

Accessibility Graph (3D): Let S be the starting point, t_0 be the starting time, v be the speed of the mobile robot and D be the destination point. With each accessible point X , an item of time $t(X)$ is associated to it. The three dimensional accessibility graph can be constructed as follows:

1. Insert the starting point S in the vertex of the accessibility graph and set its default accessible time to t_0 .

2. For every newly added vertex U in the set of vertices in the accessibility graph, calculate all the accessibility polyhedra of the polyhedral obstacles in the environment with U as the initial point. For every accessibility polyhedron, some vertices are added to its edges so that the maximum length of every edge is less than a given value E . Find all the visible vertices of the accessibility polyhedra and insert them and all the line edges from U to these visible vertices into the accessibility graph together with their accessible time items.

The three dimensional accessibility graph defined and constructed above is infinite. In chapter 5, only the accessible points of the vertices of the polygonal obstacles are added to the two dimensional accessibility graph as its vertices, so it is easy to make it definite by adopting the rule that if a vertex appears more than once, the one with the least accessible time is kept. However, for the three dimensional accessibility graph, not only the accessible points of the vertices of the polyhedral obstacles are added to it, but some fictitious vertices on the edges of the accessibility polyhedra are also added to it, so it is difficult to make the three dimensional accessibility graph finite using two dimensional method.

The purpose of constructing the three dimensional accessibility graph is for solving the problem of motion planning in a three dimensional dynamic environment. With this as the only goal, it is easy to make it finite. First, the minimum time motion of the robot only bends either at internal points on obstacle edges or vertices of polyhedra (property 1), that is, the minimum time motion will not bend at points on the edges of any accessibility polyhedron which are created by approximating the collision front by faces (e. g., edge $mAmC$ in Fig. 6.9). The three dimensional accessibility graph can be made finite by adopting the rule that fictitious vertices are added to the edges of the polyhedral obstacles so that the fictitious vertices added to the accessibility polyhedra of these obstacles are the corresponding meeting (accessible) points of these added vertices. When the same vertex (one of the vertex or the added fictitious vertices) on any polyhedral obstacle appears more than once as nodes of the three dimensional accessibility graph, the node with the smallest accessible time will be kept, all the others will be deleted from the graph.

With the definition of the three dimensional accessibility graph, the following procedure can generate the minimum time motion among moving polyhedral obstacles in a three dimensional environment.

1. Insert the starting point S into the queue OPEN.
2. Remove a vertex V whose associated accessible time is the smallest from queue OPEN.
3. If V is the destination point then report the motion and exit with success. Otherwise calculate all the accessibility polyhedra of the polyhedral obstacles in the environment with V as the starting point. For every accessibility polyhedron, some vertices are added to its edges so that the maximum length of every edge is less than a given value E. Find all the visible vertices (including the destination point) of the accessibility polyhedra from V and put them into V's successor set VM. Compare every node in VM with the node in OPEN, if the node does not appear in OPEN then put it into OPEN, else compare the associated accessible time of the node with the associated accessible time of the old node in OPEN, if the associated accessible time of the node in VM is less, then replace the old node in OPEN with that node in VM, else delete the node from VM.

For the motion planning of a robot with mass, that is, it takes time for the robot to accelerate from zero speed to its maximum speed, the accessibility graph algorithm can also be extended to solve this problem. The concept of the improved accessibility polyhedron can be defined in a similar manner as the definition of the improved accessibility polygon. For every face of a polyhedral obstacle, if the angle between its outgoing normal and the its velocity vector is less than 90 degrees, the improved collision front of that face is calculated by the following equations

$$\left\{ \begin{array}{l} t = \frac{\sqrt{x^2 + y^2 + (z - A)^2}}{v_{\max}} + \frac{v_{\max}}{2A_{\max a}} + \frac{v_{\max}}{2A_{\max d}} \\ x = x_0 + v_{Fx} \bullet t \\ y = y_0 + v_{Fy} \bullet t \\ z = z_0 + v_{Fz} \bullet t \end{array} \right. \quad (6.5)$$

where, $v_{\max}$ is the maximum speed of the robot, (x_0, y_0, z_0) is the co-ordinate of a point on the moving face at the starting time (time 0), (v_{Fx}, v_{Fy}, v_{Fz}) are the components of v_F on X, Y and Z axes respectively, A is the Z component of the starting point of the robot, $A_{\max a}$ and $A_{\max d}$ are the maximum acceleration and maximum deceleration respectively; else the collision front is calculated by (6.4). The improved accessible points corresponding to its vertices are also calculated in the same manner by (6.5) or (6.4). The collision front or the improved collision front of a moving face can also be approximated by several faces in a similar manner discussed above. The improved accessibility polyhedron of a moving polyhedral obstacle can also be formed by the these approximating faces and the trajectory of the moving obstacles. A similar procedure can also be used to construct and search the improved accessibility graph to generate the minimum time motion for a mobile robot with mass.

If the number of fictitious vertices added to the edges of a polyhedral obstacle is big, huge memory is required to store the three dimensional accessibility graph or improved accessibility graph, so the computation is very time consuming. On the contrary, if the number of fictitious vertices is small, the minimum time motion thus generated will have a big gap between the real one. So it is important to select the proper number of vertices added on the edges of the polyhedral obstacles for this algorithm. The computation time of the three dimensional accessibility graph or improved accessibility graph algorithm depends on the number of edges of a polyhedron, the number of the polyhedra and the number of the fictitious vertices

added to the edges of the accessibility polyhedra or improved accessibility polyhedra. It is difficult to get the analytical form for the evaluation of the computation time.

6.5 Summary and Discussion

The problem of motion planning in three dimensional static and dynamic environments has been studied in this chapter. The accessibility graph algorithm which is developed in chapter 5 is extended to solve the motion planning problem in three dimensional environments where the space time approach has difficulties, since the space time will transform the three dimensional space into a four dimensional space. The four dimensional space time is difficult to represent in geometrical space and use more memory. The improved accessibility graph algorithm which is proposed in chapter 5 to solve the motion planning problem for mobile robots with mass is also extended to solve the same problem in three dimensional environments. All these suggest that the time complexity for motion planning in a dynamic environment may be the same as for the stationary case. However it is difficult to prove this.

Chapter

7. Motion Planning in Binary Image Representation Environments

7.1 Introduction

Motion planning problems in static and dynamic environments have been dealt with in previous chapters. The obstacles in environments, stationary or moving, are assumed to be polygons (polyhedra in three dimensional environments). For real applications of intelligent mobile robots, the environment is recognised by sensors on the robot. Cameras are widely used as sensors for mobile robots and become more and more popular. In this case, the environment is represented as a binary image where the obstacle region is represented as 1 and the free region is represented as 0. The environment which is represented by a binary image is called a binary image representation environment. In a binary image representation environment, the whole environment is an image which is represented as an array of a certain size, say 512 by 512. The original image is a grey image which may contain noise and it may be difficult to distinguish the obstacle region from the free region. This original image

must be processed to separate the obstacle region (objects in the image) from the free region (background). This chapter will deal with the processing and motion planning in such environments.

In the first section of this chapter, fuzzy logic methods for digital image processing are studied. Fuzzy logic methods can be used to smooth, intensify, threshold an image and to detect edges of objects in an image. They can also be used to recognise objects and analyse the motion of objects in a sequence of images.

As discussed in chapter 2, the visibility graph for a binary image representation environment is not efficient, the approach usually used is the quadtree representation approach. This approach is effective in a static environment for shortest path planning. However, for minimum time path planning problems in static environments and motion planning problems in dynamic environments, there are difficulties with the quadtree approach. The disadvantage with a binary image representation environment is that it is very time consuming to transform a physical obstacle into the robot's configuration space. Approximating such an obstacle by a polygon will overcome this. In the second section of this chapter, a new approach which uses a rectangle to approximate a binary obstacle to plan the motion for a robot in static or dynamic environments is proposed based on the concept of the principal fuzzy index of area coverage. It is suitable for static and dynamic environments. In the last section, the obstacles in a binary image representation environment are polygonized and the algorithms developed in chapters 3, 4, and 5 can be used to plan the motion of mobile robots without any modification.

7.2 Fuzzy Logic Methods for Digital Image Processing

7.2.1 Introduction

Fuzzy set theory was first developed in 1965 by L. Zadeh as an extension to traditional set theory, along with the fuzzy logic to manipulate the fuzzy sets. A fuzzy set allows for degrees of membership in a set. A membership function defines the grade of membership in a fuzzy set for all the possible members. This allows human observation, expression and expert knowledge to be more closely modelled. Fuzzy logic makes tough problems much easier to solve by allowing a more natural representation of the situation being dealt with.

A fuzzy set A in a universe of discourse U is characterised by a membership function μ_A which takes values in the interval $[0, 1]$, namely, $\mu_A: U \rightarrow [0, 1]$. A fuzzy set A in a discrete universe of discourse U with $u_1, u_2, \dots, u_n$ elements can be represented as:

$$A = \sum \mu_A(u_i) / u_i \quad (7.1)$$

The element u in U at which $\mu_A(u)=0.5$ is called the crossover point. A fuzzy set whose member ($\mu_A(u_i) > 0$) is a single point in U is referred as fuzzy singleton. Depending on the applications, a membership function $\mu_A(u)$ can be defined by using the standard S-function, π -function, triangle shape, etc..

An image X of $M \times N$ dimension with L grey levels can be considered as an array of $M \times N$ fuzzy singletons, each with a membership function value denoting the property of the image at that point [Pal, S. 1981]. It has been found that low level digital image processing bears some fuzziness in nature due to the factors such as:

- the lack of the quantitative measurement of image quality.
- the loss of information due to many-to-one features in image grabbing.

- the inherent distortion or noise in the image being processed.
- the subjective selection of the membership functions.

Fuzzy sets have been applied to low level grey tone image processing operations such as grey level thresholding, edge detection, image skeleton, image enhancement and segmentation [Pal, S. 1988] [Pal, S. 1990] [Pal, S. 1989] [Pal, S. 1981] [Murthy, C. 1990] [Goetcheian, V. 1980] [Li, H. 1988] [Yan, J. 1992a]. Recently, fuzzy features such as fuzzy index of compactness are also used to recognise objects in a sequence of images and calculate the motion of the objects [Yan, J. 1992b]. However, the method based on fuzzy index of compactness can only calculate the displacement of an object, it can not calculate the rotation of the object. An approach based on the principal fuzzy index of area coverage is proposed to deal with the recognition of objects in consecutive images in later part of this section. This approach can not only calculate the displacement of an object, but can also calculate the rotation of the object.

7.2.2 Fuzzy Measures

An image X of size $M \times N$ with L grey levels can be considered as an array of fuzzy singletons, each having a value of membership denoting its degree of brightness relative to some brightness level l ($l=0, 1, \dots, L-1$). In the notation of fuzzy sets, the image X can be expressed as

$$X = \{ \mu_X(x_{mn}) = \mu_{mn} / x_{mn} \} \quad (7.2)$$

where $m=0, 1, \dots, M-1$, $n=0, 1, \dots, N-1$. The standard S-function (7.3) is often used to fuzzify the image.

$$S(x_{mn}, a, b, c) = \begin{cases} 0 & x_{mn} < a \\ 2 \bullet \left(\frac{x_{mn} - a}{c - a} \right)^2 & a \leq x_{mn} \leq b \\ 1 - 2 \bullet \left(\frac{c - x_{mn}}{c - a} \right)^2 & b < x_{mn} \leq c \\ 1 & x_{mn} > c \end{cases} \quad (7.3)$$

where $b=(a+c)/2$ is called the cross over point.

Fuzzy measures play a very important role in fuzzy operations of images. The usually used fuzzy measures in digital image processing include index of fuzziness, fuzzy entropy, fuzzy area, fuzzy perimeter, fuzzy index of compactness, fuzzy height, fuzzy width, fuzzy length, fuzzy breadth and fuzzy index of area coverage, etc.. Their definitions will be described below.

7.2.2.1 Index of Fuzziness

The index of fuzziness [Pal, S. 1988] [Yan, J. 1992a] of a fuzzy set is defined as the metric distance between fuzzy set A and its nearest crisp set C. The nearest crisp set C of a fuzzy set A is denoted by

$$\mu_C(x) = \begin{cases} 0 & \text{if } \mu_A(x) \leq 0.5 \\ 1 & \text{if } \mu_A(x) > 0.5 \end{cases} \quad (7.4)$$

So the index of fuzziness of an image X can be expressed as

$$v(X) = \frac{2}{M \bullet N} \left[\sum_{n=0}^{N-1} \sum_{m=0}^{M-1} |\mu_X(x_{mn}) - \mu_{X'}(x_{mn})|^k \right]^{\frac{1}{k}} \quad (7.5)$$

where X' denotes the nearest crisp set of fuzzy set X of an image which is fuzzified by (7.3), k is an integer which is bigger than zero. If $k=1$, the index of fuzziness is called the linear index of fuzziness. It can be expressed as

$$v_l(X) = \frac{2}{M \bullet N} \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} |\mu_X(x_{mn}) - \mu_{X'}(x_{mn})| \quad (7.6)$$

With the definition of the nearest crisp set of a fuzzy set in (7.4), it is obvious that the linear index of fuzziness can be expressed as

$$v_l(X) = \frac{2}{M \bullet N} \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} \min[\mu_X(x_{mn}), 1 - \mu_X(x_{mn})] \quad (7.7)$$

7.2.2.2 Fuzzy Entropy

The term “fuzzy entropy” [Murthy, C. 1990] uses Shannon’s function but its meaning is quite different from classical entropy because no probabilistic concept is needed to define it. The fuzzy entropy of an image X is defined as

$$H(X) = \frac{1}{M \bullet N \bullet \ln 2} \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} S_n(\mu_X(x_{mn})) \quad (7.8)$$

with Shannon’s function

$$S_n(\mu_X(x_{mn})) = -\mu_X(x_{mn}) \bullet \ln(\mu_X(x_{mn})) - (1 - \mu_X(x_{mn})) \bullet \ln(1 - \mu_X(x_{mn})) \quad (7.9)$$

7.2.2.3 Fuzzy Area

The fuzzy area [Pal, S. 1990] of a fuzzy subset μ is defined as

$$a(\mu) = \int \mu \quad (7.10)$$

where the integration is taken over a region outside which $\mu=0$. In the case of an image, the fuzzy membership function μ is piecewise constant, the fuzzy area is defined as

$$a(X) = \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} \mu_X(x_{mn}) \quad (7.11)$$

7.2.2.4 Fuzzy Perimeter

If the membership function $\mu(x)$ of a fuzzy set is piecewise constant, the perimeter [Pal, S. 1990] of μ is defined as

$$p(\mu) = \sum_{i,j,k} |\mu(i) - \mu(j)| \bullet |A(i, j, k)| \quad (7.12)$$

This is just the weighted sum of the lengths of the arcs $A(i, j, k)$ along which the regions having μ values $\mu(i)$ and $\mu(j)$ meet, weighted by the absolute difference of these values. In case of an image, if the pixels are considered as the piecewise constant region, and the common arc length for adjacent pixels is considered as unity, then the perimeter of an image is defined by

$$p(\mu) = \sum_{i,j} |\mu(i) - \mu(j)| \quad (7.13)$$

where $\mu(i)$ and $\mu(j)$ are the membership values of two adjacent pixels [Pal, S. 1990].

7.2.2.5 Fuzzy Index of Compactness

The index of compactness of a fuzzy set μ having an area of $a(\mu)$ and a perimeter $p(\mu)$ is defined as [Pal, S. 1988] [Pal, S. 1990] [Yan, J. 1992a] [Yan, J. 1992b]

$$IOC(\mu) = \frac{a(\mu)}{(p(\mu))^2} \quad (7.14)$$

Physically, the index of compactness means the fraction of maximum area (that can be encircled by the perimeter) actually occupied the object. In the nonfuzzy case, the value of index of compactness is maximum for a circle and this value is $1/4\pi$. In case of a fuzzy disc, where the membership function value is only dependent on its distance from the centre, the index of compactness is bigger than $1/4\pi$ [Pal, S. 1990]. Of all possible fuzzy discs, the index of compactness is, therefore, minimum for its crisp set.

7.2.2.6 Fuzzy Height and Width

The height [Pal, S. 1990] of a fuzzy set μ is defined as

$$h(\mu) = \int \max_x \{\mu(x, y)\} dy \quad (7.15)$$

where the integration is taken over a region outside which $\mu(x, y)=0$.

Similarly, the width of a fuzzy set [Pal, S. 1990] is defined by

$$w(\mu) = \int \max_y \{\mu(x, y)\} dx \quad (7.16)$$

with the same condition over integration as above.

For a digital image, x and y can take only discrete values, and since $\mu=0$ outside the bounded region, the maxes are over a finite set. In this case, the fuzzy height and width of an image take the form

$$h(\mu) = \sum_y \max_x \{\mu(x, y)\} \quad (7.17)$$

and

$$w(\mu) = \sum_x \max_y \{\mu(x, y)\} \quad (7.18)$$

So physically, in case of a digital image, the fuzzy height is the sum of the maximum membership function values of each row. Similarly, the fuzzy width is the sum of the maximum membership function values of each column.

7.2.2.7 Fuzzy Length and Breadth

The length of a fuzzy set μ [Pal, S. 1990] is defined as

$$l(\mu) = \max_x \left\{ \int \mu(x, y) dy \right\} \quad (7.19)$$

where the integration is taken over a region outside which $\mu(x, y)=0$.

In case of a digital image where x and y can take only discrete values, the expression takes the form

$$l(\mu) = \max_x \left\{ \sum_y \mu(x, y) \right\} \quad (7.20)$$

Physically speaking, the length of an image fuzzy subset gives its longest expansion in the Y direction. If μ is crisp, $\mu(x, y)=0$ or 1, for background and object pixels respectively; the length then denotes the maximum number of object pixels in the Y direction.

Comparing equations (7.20) and (7.17) it is obvious that the length is different from the height in the sense that the former takes the summation of the entries in each column first and then maximises over different rows whereas the later maximises the entries in each row and then sums over different column. It is also obvious that

$$l(\mu) / h(\mu) \leq 1 \quad (7.21)$$

The breadth of a fuzzy set μ [Pal, S. 1990] can be defined in a similar way

$$b(\mu) = \max_y \left\{ \int \mu(x, y) dx \right\} \quad (7.22)$$

where the integration is taken over a region outside which $\mu(x, y)=0$.

In case of a digital image, where x and y can take only discrete values the breadth will take the form

$$b(\mu) = \max_y \left\{ \sum_x \mu(x, y) \right\} \quad (7.23)$$

The fuzzy breadth of an image gives its longest expansion in the X direction. If μ is crisp, $\mu(x, y)=0$ or 1; the breadth denotes the maximum number of object pixels in the direction.

From equations (7.23) and (7.18) it is obvious that the difference between breadth and width is the same as that between length and height. The relation between breadth and width of a fuzzy set is also

$$b(\mu) / w(\mu) \leq 1 \quad (7.24)$$

7.2.2.8 Fuzzy Index of Area Coverage (FIOAC)

The index of area coverage of a fuzzy set μ [Pal, S. 1990] is defined as

$$IOAC(\mu) = \frac{area(\mu)}{l(\mu) \bullet b(\mu)} \quad (7.25)$$

In the nonfuzzy case, IOAC has a maximum value of 1 for rectangles placed along the axes of measurement. For a circle this value is $\pi r^2 / (2r \bullet 2r) = \pi/4$. Physically, by the fuzzy index of area coverage (FIOAC) of an image, we mean the fraction of the maximum area (that can be covered by the length and breadth of the image) actually covered by the image.

7.2.3 Fuzzy Operators for Digital Images

The fuzzy operators for a digital image include fuzzy intensification, fuzzy edge detection, fuzzy smoothing and fuzzy thresholding, etc.. They can be described as follows.

7.2.3.1 Contrast Intensification and Enhancement

The contrast intensification operator “INT” on a fuzzy set X of an image generates another fuzzy set $X' = INT(X)$, the membership function of which is

$$\mu_{X'}(x) = INT(\mu_X(x)) = \begin{cases} 2 \bullet [\mu_X(x)]^2, & 0 \leq \mu_X(x) \leq 0.5 \\ 1 - 2 \bullet [1 - \mu_X(x)]^2, & 0.5 < \mu_X(x) \leq 1 \end{cases} \quad (7.26)$$

This operation reduces the fuzziness of a set A by increasing the values of $\mu_X(x)$ which are above 0.5 and decreasing those which are below 0.5. In fact the operator INT defined in (7.26) is a transformation of the membership function and is defined by T_1 .

In general each pixel's membership function of an image X, μ_{mn} defined by (7.2) and (7.3) may be modified to μ'_{mn} to enhance the image in the fuzzy plane by the transformation function T_r [Pal, S. 1981]:

$$\mu'_{mn} = T_r(\mu_{mn}) = \begin{cases} T_r'(\mu_{mn}), & 0 \leq \mu_{mn} \leq 0.5 \\ T_r''(\mu_{mn}), & 0.5 < \mu_{mn} \leq 1 \end{cases} \quad (7.27)$$

where $r=1, 2, \dots$, the transformation function T_r is defined as successive applications of T_1 by the following recursive relationship

$$T_s(\mu_{mn}) = T_1(T_{s-1}(\mu_{mn})) \quad (7.28)$$

and $T_1(\mu_{mn})$ represent the operator INT defined in (7.26).

This can be shown graphically in Fig. 7.1. As r increases, the curve gets steeper because of the successive application of INT. In the limiting case, as $r \rightarrow \infty$, T_r produces a two level (binary) image.

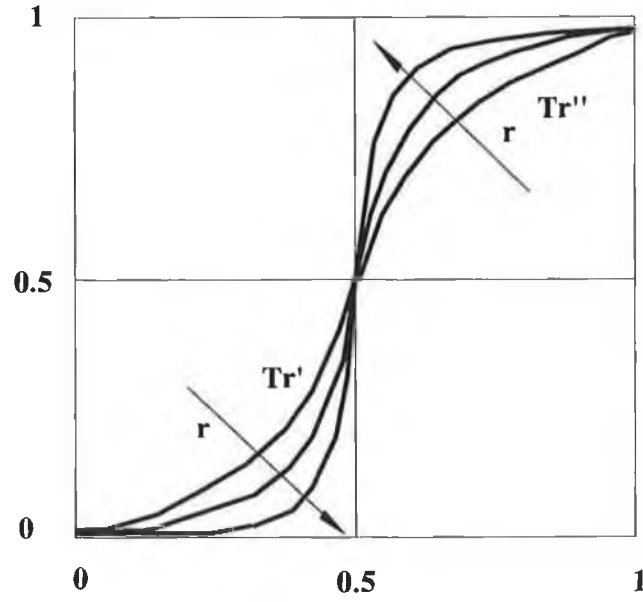


Fig. 7.1: INT transformation function for contrast enhancement

If the fuzzy set of an image X is fuzzified by (7.2) and (7.3), the enhanced image can be obtained by defuzzifying μ'_{mn} [Pal, S. 1981] [Yan, J. 1992a].

$$x'_{mn} = \begin{cases} a + (c - a) \cdot \sqrt{\frac{\mu'_{mn}}{2}}, & 0 \leq \mu'_{mn} \leq 0.5 \\ c - (c - a) \cdot \sqrt{\frac{1 - \mu'_{mn}}{2}}, & 0.5 < \mu'_{mn} \leq 1 \end{cases} \quad (7.29)$$

7.2.3.2 The MIN and MAX operators and their operations

The fuzzy operators of MIN and MAX [Yan, J. 1992a] can be classified into point-wise function (MIN_P , MAX_P) and local functions (MIN_L , MAX_L). The point-wise functions MIN_P and MAX_P known as the fuzzy intersection and union, perform the operations at each point on the image. The local functions MIN_L and MAX_L , known as the minimum and maximum operation replace the digital value of the centre pixel

by the minimum (or maximum) grey value of its neighbours and itself inside a specified window (say 3×3).

The MIN_L and MAX_L operators resemble shrinking and expanding operators. The MIN_L and MAX_L operations can be carried out either on the digital grey level image plane or on the fuzzy property plane of the image.

The edge detection of the grey level image [Yan, J. 1992a] can be achieved by taking the difference of the original image X and its shrunk version $MIN_L(X)$, that is:

$$Fuzzy\ Edges = X - MIN_L(X) \quad (7.30)$$

The strength of the detected edge at any point is proportional to the largest local digital value at that point in the original image.

Image smoothing is often implemented to blur the image by attenuating the high spatial frequency components associated with edges and other abrupt changes in grey levels. This can be performed by employing the shrink operator followed by the expanding operator, that is,

$$Fuzzy\ Smoothing = [MAX_L]^q [MIN_L]^q (X) \quad (7.31)$$

where $q=1, 2, \dots, N$. The higher the q , the greater is the degree of blurring.

7.2.3.3 Fuzzy Thresholding

The index of fuzziness defined in (7.4), (7.5) and (7.6) reflects the average amount of fuzziness in the image by measuring the distance between the extracted fuzzy property plane and its nearest crisp two tone version. Different cross-over points (b) of the fuzzifying S-function (7.3) will generate different indices of fuzziness. Fuzzy thresholding [Yan, J. 1992a] [Pal, S. 1988] is to select a cross-over point, b , in (7.3)

so that the index of fuzziness is a minimum. This cross-over point b is the thresholding grey level, that is, with the thresholding level l as the cross over point b of the fuzzifying S-function, the distance between the extracted fuzzy property plane and its nearest crisp two tone version is a minimum. For a fixed window size ($w=c-a$) of the S-function, the fuzzy thresholding algorithm can be stated as follows [Pal, S. 1988] [Yan, J. 1992a].

1. Construct the membership function $\mu_X(x_{mn})$ of the image X using the S-function (7.3), that is, $\mu_X(x_{mn})=S(x_{mn}, a, b, c)$, where $b=l_i$ (grey level).
2. Compute the linear index of fuzziness $v_l(X)$ of μ_X corresponding to $b= l_i$ by (7.6).
3. Varying l_i from the minimum grey level $l_{\min}$ of the image X to the maximum grey level $l_{\max}$ of the image X . The l_i with which $v_l(X)$ is the smallest is the optimum thresholding grey level for the image X .

The above algorithm is effective for bimodal images. For multimodal images, different window sizes of the S-function and different searching regions $[l_{\min}, l_{\max}]$ will generate different “optimum” thresholding results [Yan, J. 1992a]. An algorithm which can solve the multimodal image thresholding problem based on fuzzy logic is proposed by J. Yan and M. Ryan [Yan, J. 1992a]. The automatic thresholding procedure for a three-modal image developed by J. Yan and M. Ryan can be described as follows.

1. Describe the grey level regions of the image in linguistic terms and find the grey level region boundaries TS_1 and TS_2 for the objects in the image.
2. Determine the local grey level search ranges: $[l_{\min}, \min(TS_1, TS_2)]$ and $[\min(TS_1, TS_2), \max(TS_1, TS_2)]$.
3. Choose the initial window size w starting from small number (say $w=c-a=5$).

4. Specify the initial grey level $l_1 = l_{\min}$ for the first local grey level search range, $l_1 = \min(TS_1, TS_2)$ for the second local grey level search range.
5. Construct the fuzzy property plane $\mu_X(x_{mn})$ using the S-function, that is, $\mu_X(x_{mn}) = S(x_{mn}, a, b, c)$, where $b = l_i$ represents the i th grey level.
6. Compute the linear index of fuzziness $v_l(X)$ by (7.6) corresponding to $b = l_i$.
7. Varying l_i from the lower boundary to the upper boundary in each local grey level region, go back to step 4 until l_i reaches the upper boundary.
8. Find the minimum $v_l(X)$ computed together with the optimum thresholding grey level $l_t = l_i$.
9. If more than one thresholding grey level corresponds to the minimum $v_l(X) = 0$, the selection of the window size has failed. If the selected thresholding level l_t is large, increase the window size w by a large amount. If l_t is small, increase the window size w by a small amount, go back to step 4.
10. In the case that the minimum $v_l(X)$ is not equal to zero and there is only one threshold corresponding to it, select $l_t = l_i$ for which $v_l(X)$ is the minimum. If the selected threshold is not too close to the lower or the upper boundary of the local grey level search range, go to step 11. Otherwise, decrease the window size a little bit and go back to step 4.
11. Try a few nearby window size (given the window size some distance) and choose the thresholding level l_t where $v_l(X)$ is the minimum.

In the above procedure, the threshold is automatically selected by considering the index of fuzziness under a given window size. The window size will be changed in the case that either the computed minimum index of fuzziness is unreasonable or the

selected thresholding level is too close to the boundaries of the local search range [Yan, J. 1992a].

7.2.4 Fuzzy Logic based Image Motion Detection and Analysis

Image motion detection is a process of estimating the perceived change in the scene as a consequence of observed image plane spatiotemporal change, while image motion analysis is a process of yielding significant information on the change and the structure of the 3-D scene under observation. Image motion detection and analysis is required for many motion based guidance applications. For instance, mobile robots with visual sensors rely on the analysis of the detected motions to avoid obstacles in a dynamic environment cluttered with moving obstacles.

In the process of image motion detection and analysis, the following procedures have to be undertaken:

1. Extract features (points, segments or regions) for each object on the image plane.
2. Match the corresponding features (points, segments or regions) across two images.
3. Estimate the movements of the features (points, segments or regions) on the image plane.
4. Find the corresponding motions of the objects in the real world scene under observation.

In the literature, various techniques have been described to perform image motion detection and analysis based on the status of the observer. For a stationary observer, a simple but frequently used approach is to differentiate two consecutive image

frames numerically, and mark the nonzero regions of the resulting image. This strategy has formed the basis for a number of motion detection and analysis systems [Roach, J. 1980] [Tsai, R. 1981] [Fang, J. 1984]. For a moving observer, the problem becomes much harder since everything in the image may be undergoing apparent motion and the overall pattern of motion may be quite complex.

Both numerical and qualitative approaches for image motion detection and analysis have been proposed based on the computation of the optical flow and the identification of corresponding image features in the two consecutive images. The numerical approaches [Thompson, W. 1990] [Roach, J. 1980] [Tsai, R. 1981] [Fang, J. 1984], which involve the computations of linear or non-linear equations, tend to be computationally expensive and noise sensitive. The qualitative approaches [Bhanu, B. 1991] [Nelson, R. 1991] [Bhanu, B. 1992], which either utilise the qualitative information derived from point correspondences or make use of the qualitative knowledge about the motion of the objects in the scene, allow a certain amount of imprecision or noise in the images to be accommodated by manipulating linguistic descriptions.

Whatever approach is used, the central issue is the correspondence problem, that is, how to find and match the corresponding features for each object across two consecutive images.

J. Yan, R. Lawlor and M. Ryan have proposed an approach which uses fuzzy measures to find and match the corresponding features for objects across two consecutive images. The fuzzy index of compactness which is defined by (7.14) and is scale-invariant and rotation-invariant is used to match objects in two consecutive images. The approach includes several steps and can be briefly described as follows.

1. Remove noise from the images by using the fuzzy smoothing operators defined above.

2. Segment the objects from the background using the fuzzy thresholding operation described above.
3. Detect and locate each object on the image plane as a rectangular region which bounds the object.
4. Extract features from the located regions such as IOC.
5. Match the corresponding features across two consecutive images.
6. Estimate the displacements of objects in the images.
7. Find the corresponding motions of the objects in the real world scene under observation.

The algorithm for an object locating and boundary tracing in step 3 can be stated as:

1. Set the starting search point as the origin of the image co-ordinate (0, 0).
2. Search the objects on the segmented image plane along each row of the image.
3. Stop the search if any pixel whose value is different from that of the background is encountered. Register the co-ordinate of the current pixel on the image plane as (x_0, y_0) . Go to step 6.
4. Continue the searching process if there is any pixel whose value does not belong to the background encountered on the image plane.
5. Terminate the searching process if nothing of interest is found.
6. Trace the boundary of the object detected.

- A. Set the initial object boundary pixel co-ordinate (x_0, y_0) as current boundary tracing point co-ordinate (x, y) . Set $(x_{\max}, y_{\max}) = (x_0, y_0)$ and $(x_{\min}, y_{\min}) = (x_0, y_0)$.
- B. Form the 8-connected neighbours of the current pixel, choose the neighbour which satisfies the following conditions as the boundary pixel of the object. The conditions are: neighbour is object pixel, neighbour has 4-connected background pixels, neighbour is inside the region being examined, and neighbour is not the last or the second last current pixel.
- C. Compute the maximum and minimum values of the bounded region co-ordinate by: $x_{\max} = \max(x_{\max}, x)$, $x_{\min} = \min(x_{\min}, x)$, $y_{\max} = \max(y_{\max}, y)$ and $y_{\min} = \min(y_{\min}, y)$.
- D. Set the chosen neighbour (x, y) as current tracing point co-ordinate.
- E. Go to step B if the current tracing point does not go back to the initial point (x_0, y_0) .
- F. Otherwise, check the validity of the searched boundary. Register the values $[(x_{\min}, y_{\min}), (x_{\max}, y_{\max})]$ if they are valid. Delete the image bounded from the image plane (that is, replace the pixel values of the bounded region with the pixel value of the background). Go to step A.

The algorithm can match and find the displacements of objects in two consecutive images. However, it can not find rotations of objects. The match of features in the consecutive images is performed using only the fuzzy index of compactness (IOC), and is not always reliable. Using more scale-invariant and rotation-invariant fuzzy measures to match the objects in consecutive images will make the match more reliable. The fuzzy index of area coverage (IOAC) defined by (7.25) looks to be such a measure. It is scale-invariant. Unfortunately, it is not rotation-invariant. If an object in the real world rotates, the non rotation-invariant measures can not match the real features of the object. However, the IOAC can be made rotation-invariant by a special co-ordinate transformation. This will lead to the definition of the concept of the principal index of area coverage of a fuzzy subset.

In mechanics, the moments of an object in a two dimensional Euclidean space are defined as (see Fig. 7.2)

$$m_{p,q} = \iint (x - x_c)^p \bullet (y - y_c)^q \bullet f(x, y) dx dy \quad (7.32)$$

where $f(x, y)$ is the unit weight function of the object, (x_c, y_c) are the co-ordinates of the centre of gravity of the object:

$$x_c = \frac{\iint x \bullet f(x, y) dx dy}{\iint f(x, y) dx dy} \quad (7.33)$$

$$y_c = \frac{\iint y \bullet f(x, y) dx dy}{\iint f(x, y) dx dy} \quad (7.34)$$

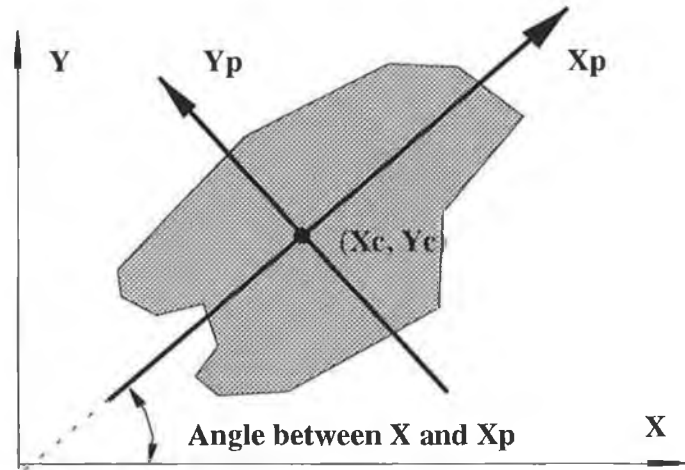


Fig. 7.2: Principal axis of an object

The principal axis of the object is the axis around which the object can be rotated with minimum inertia. The object is most elongated in this direction (see Fig. 7.2). Because the inertia of the object around the principal axis is the minimum, it must pass through the centre of gravity. The angle between the principal axis and the X-axis (see Fig. 7.2) can be calculated by

$$\phi = \frac{1}{2} \arctan\left(\frac{2 \bullet m_{1,1}}{m_{2,0} - m_{0,2}}\right) \quad (7.35)$$

The fuzzy moments of a fuzzified image can be defined in a similar manner

$$m_{p,q} = \iint (x - x_c)^p \bullet (y - y_c)^q \bullet \mu(x, y) dx dy \quad (7.36)$$

where $\mu(x, y)$ is the membership function of the image, (x_c, y_c) is the co-ordinate of the centroid

$$x_c = \frac{\iint x \bullet \mu(x, y) dx dy}{Area(\mu)} \quad (7.38)$$

$$y_c = \frac{\iint y \bullet \mu(x, y) dx dy}{Area(\mu)} \quad (7.39)$$

For an object in a image, if the rectangular region $[(x_{min}, y_{min}), (x_{max}, y_{max})]$ is found by the procedure developed above, this rectangular region can be treated as a fuzzy subset. The moments of this fuzzy subset can be calculated by

$$m_{p,q} = \sum_{x=x_{min}}^{x_{max}} \sum_{y=y_{min}}^{y_{max}} (x - x_c)^p \bullet (y - y_c)^q \bullet \mu(x, y) \quad (7.40)$$

where (x_c, y_c) is calculated by

$$x_c = \frac{\sum_{x=x_{min}}^{x_{max}} \sum_{y=y_{min}}^{y_{max}} x \bullet \mu(x, y)}{\sum_{x=x_{min}}^{x_{max}} \sum_{y=y_{min}}^{y_{max}} \mu(x, y)} \quad (7.41)$$

and

$$y_c = \frac{\sum_{x=x_{\min}}^{x_{\max}} \sum_{y=y_{\min}}^{y_{\max}} y \bullet \mu(x, y)}{\sum_{x=x_{\min}}^{x_{\max}} \sum_{y=y_{\min}}^{y_{\max}} \mu(x, y)} \quad (7.42)$$

With the moments calculated by (7.40), the angle between the principal axis of the object and X-axis can be calculated by (7.35). A new co-ordinate system can be established with the principal axis as the X_p axis, Y_p axis can be obtained by rotating Y axis by the angle calculated by (7.35), the origin of the new co-ordinate is at (x_c, y_c) . This new co-ordinate system is call the object's principal axes co-ordinate system and is shown in Fig. 7.2.

The principal index of area coverage (PIOAC) of an object in an image can be defined as the index of area coverage of the object in its principal axes co-ordinate system, that is,

$$PIOAC(\mu) = \frac{Area(\mu)}{l_p(\mu) \bullet b_p(\mu)} \quad (4.43)$$

where $Area(\mu)$ is the area of the object, $l_p(\mu)$ and $b_p(\mu)$ are the principal length and principal breadth of the object respectively. They are defined by the following equations

$$Area(\mu) = \sum_{x=x_{\min}}^{x_{\max}} \sum_{y=y_{\min}}^{y_{\max}} \mu(x, y) \quad (4.44)$$

$$l_p(\mu) = \max_{x_p} [\int \mu(x_p, y_p) dy_p] = \max_{x_p} [\sum_{y_p} \mu(x_p, y_p)] \quad (4.45)$$

$$b_p(\mu) = \max_{y_p} [\int \mu(x_p, y_p) dx_p] = \max_{y_p} [\sum_{x_p} \mu(x_p, y_p)] \quad (4.46)$$

It is obvious that PIOAC is scale-invariant and rotation-invariant.

Once the objects in an image are segmented from the background and separated from each other by the rectangular regions, it is straightforward to calculate the fuzzy features (index of compactness and principal index of area coverage) of every object.

	Image 1	Image 2	Image 3
Resolution	174×154	114×148	124×157
PIOAC	0.725339	0.72455	0.728036
IOAC	0.625056	0.610835	0.607715
1/IOC	25.69325	23.967495	23.563858

Table 7.1: Results of fuzzy features of the object in Fig. 7.3



Fig. 7.3: Image of an object after thresholding

Table 7.1 shows the results of calculating the above fuzzy features of the object showed in Fig. 7.3 (after thresholding) in three consecutive images, where resolution

is the size of the minimum rectangular region which encloses the object, PIOAC represents the Principal Index Of Area Coverage, IOAC represents the Index Of Area Coverage, $1/IOC$ represents the inverse of the Index of Compactness.

Table 7.2 shows the results of another object shown in Fig. 7.4. Examining these results shows that the PIOAC is best able to recognize the objects. If the IOC above is used to match these two objects, it is difficult to distinguish them, since the IOCs of both objects with different resolution and orientation are very close. If the PIOAC is used, it is easy to distinguish them.



Fig. 7.4: Image of another object after thresholding

	Image 1	Image 2	Image 3
Resolution	172×151	140×134	111×134
PIOAC	0.691719	0.693647	0.695934
IOAC	0.715347	0.692217	0.653557
1/IOC	22.322838	22.455952	23.21006

Table 7.2: Results of fuzzy features of the object in Fig. 7.4

Once the objects are matched on the image plane, the displacement and rotation of each object can be estimated using the following equations:

$$\begin{cases} \Delta x = x_c' - x_c \\ \Delta y = y_c' - y_c \\ \Delta \theta = \theta_2 - \theta_1 \end{cases} \quad (7.47)$$

where (x_c', y_c') are the co-ordinates of the centroid of the object in the second image, (x_c, y_c) are the co-ordinates of the centroid of the object in the first image, θ_2 is the angle between the principal axis of the object and the X axis in the second image, θ_1 is the angle between the principal axis of the object and the X axis in the first image.

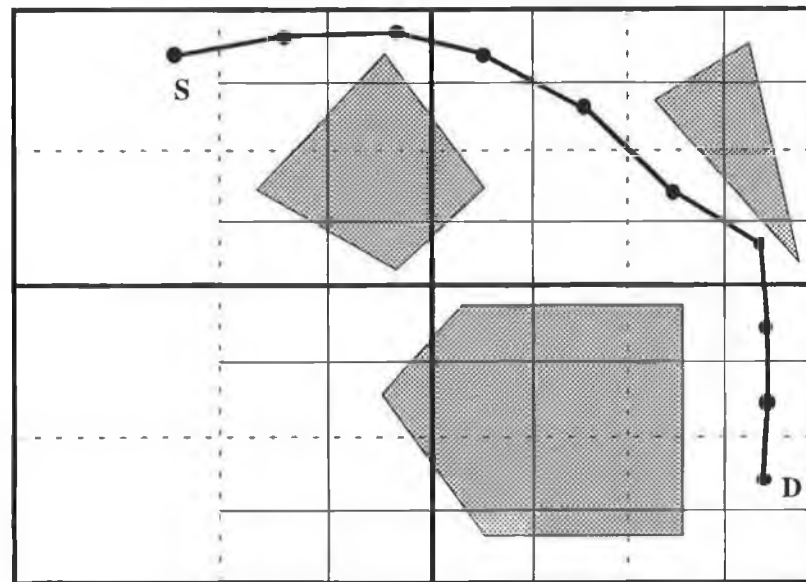
7.3 Motion Planning in Binary Image Representation Environments

7.3.1 Quadtree Approach

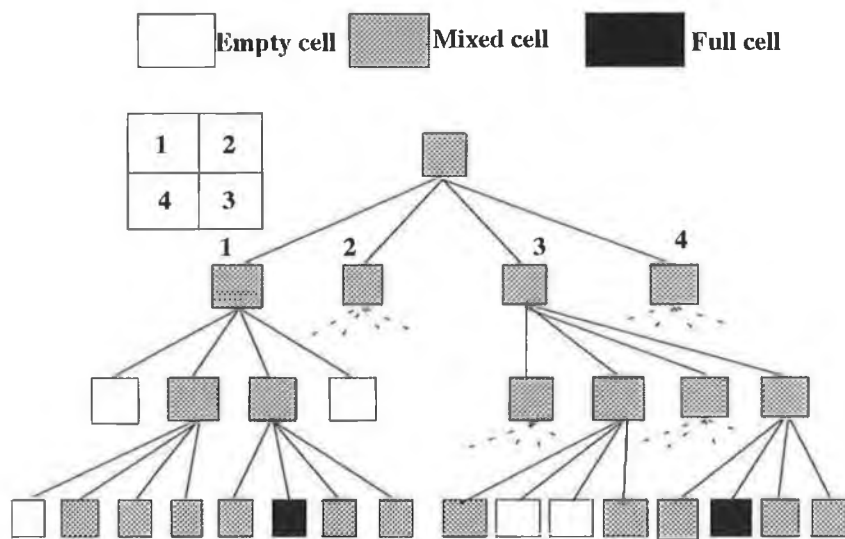
As pointed out in chapter 2, it is very inefficient to construct the visibility graph in a binary image representation environment, since there are so many vertices in an obstacle which is represented by pixels in the image. The approximate cell decomposition approach provides an alternative to solve this problem. The most widely used technique in the approximate cell decomposition is a 2^m -tree decomposition, where m is the dimension of the configuration space. For a binary image representation environment, $m=2$, in this case, the decomposition is called the quadtree decomposition.

The concept of the quadtree decomposition is to divide the rectangular environment (tree root) into four equal size rectangular regions (cells), each region is a node of the tree and is labelled as Empty, Full or Mixed (Empty means that there is no obstacle in the cell; Full means that the whole cell is occupied by obstacles; Mixed means that the

cell is partly occupied by obstacles). The decomposition is applied to each Mixed child node until the size of each mixed cell reaches a resolution value. Fig. 7.5(a) shows a quadtree decomposition of a two dimensional environment, Fig. 7.5(b) displays a portion of the quadtree graph.



(a)



(b)

Fig. 7.5: A quadtree decomposition of an environment

With the environment represented by a quadtree, a graph search algorithm can be used to search the quadtree to generate a path which is a sequence of empty or mixed cells. In other words, a path between the starting point S and the destination point D is a series of empty or mixed cells, C_0, C_1, C_n with $S \in C_0, D \in C_n, C_i$ and C_{i+1} are adjacent ($i=0, 1, \dots, n-1$). A path between the starting point S and the destination point D is shown in Fig. 7.5(a) with thick line segments.

7.3.2 Approximating an Obstacle with Its Principal Rectangle

The quadtree approach is effective for geometrical motion planning (shortest path) in a binary image representation environment. However, it is difficult to extend the quadtree approach to solve dynamic motion planning (minimum time path). For dynamic environments, the space time approach must be used to solve the minimum time motion planning problem in a binary image representation environment, since the accessibility graph algorithm is only valid for polygonal obstacles. If a binary image representation obstacle is considered as a polygon, it will be very inefficient as it has so many vertices. If the binary image representation obstacle is approximated by a polygon, it will be straightforward to use the approaches developed in chapters 3, 4 and 5 to solve various motion planning problems. Another advantage of the approximation is that it is difficult to transform a binary image representing obstacle into the robot's configuration space, while it is easy to transform a polygon into the robot's configuration space. This section will discuss the approach which approximates an obstacle with a rectangle. The approach which approximates an obstacle with a polygon will be discussed in the next section.

Principal Rectangle: the rectangle of an object in a binary representation environment is defined as the smallest rectangle which encloses the object, whose edges are parallel to the principal axes (see Fig. 7.6). Fig. 7.7 shows an example of the principal rectangle of an obstacle in a binary image representation environment.

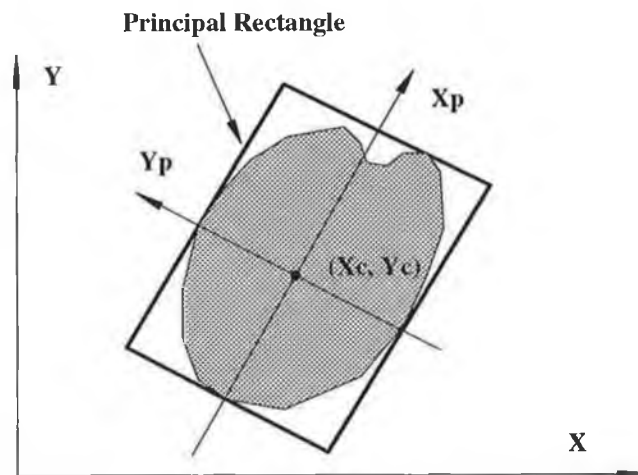


Fig. 7.6: Illustration of the definition of principal rectangle

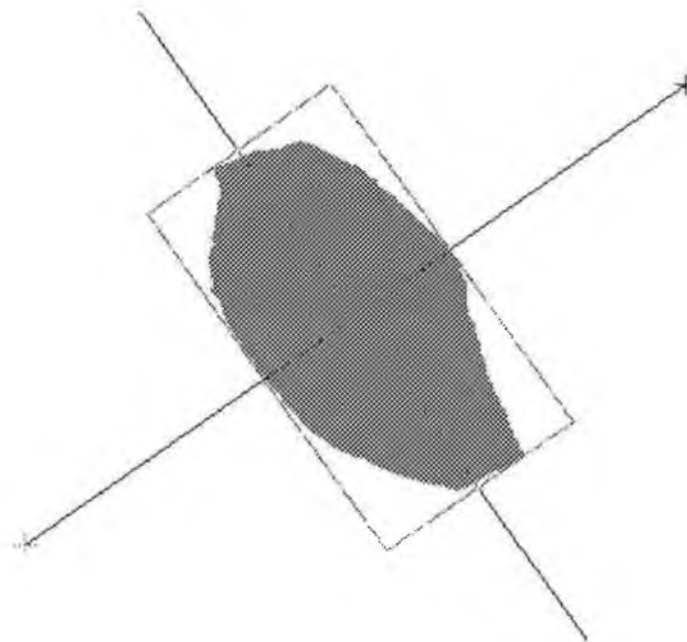


Fig. 7.7: Approximating an obstacle with its principal rectangle

Each obstacle in a binary image representation environment can be approximated by its principal rectangle. The algorithms developed in chapters 3, 4 and 5 can be used to solve various motion planning problems.

7.3.3 Polygonizing an Obstacle

Approximating a binary image representing obstacle by its principal rectangle is an effective approach to solve the motion planning problem in a binary image representation environment. The advantage is that there are only four vertices in a rectangle, so motion planning in an environment with rectangular obstacles will be computationally cheaper. The disadvantage is that the approximation of an obstacle by its principal rectangle is not very accurate. A path which is collision free may not be collision free after the approximation. Approximating the obstacle by a polygon will improve this in most cases.

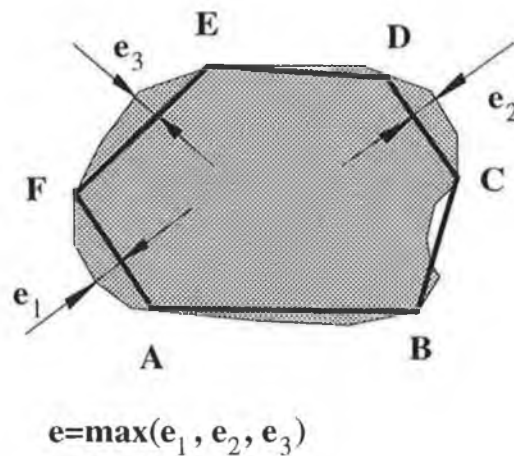


Fig. 7.8: Error of polygonizing an obstacle

The boundary of an obstacle in a binary representation environment is a closed curve. The error of polygonizing such an obstacle is defined as the biggest perpendicular distance from any segment of the resulting polygon to the farthest point on the original boundary curve between the end points of that segments (see Fig. 7.8).

The following procedure can polygonize an obstacle so that the error is within the given error range. First select the intersecting points between the boundary of the obstacle and its principal X-axis as the head point and the tail point. The head point and the tail point separate the boundary into two curve segments. For every curve segment, find the farthest point on the curve segment from the line segment which is

formed by linking the head point and the tail point. This farthest point will separate the curve segment into another two curve segments. This process is continued until the polygonizing error is less than the given error. This is shown in Fig. 7.9, the polygon formed by the thick line segments is the polygon which polygonizes the shaded obstacle.

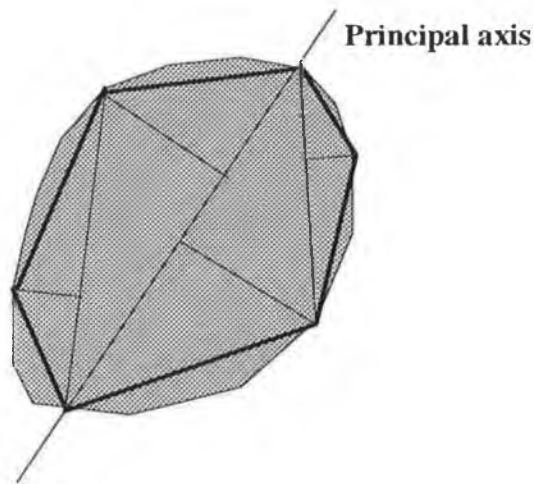
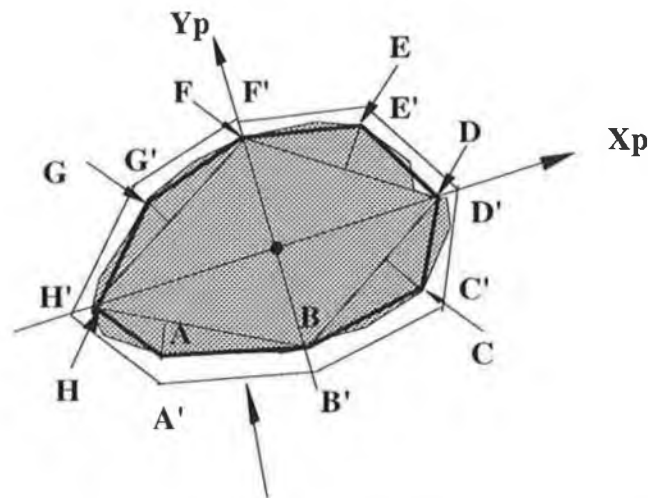


Fig. 7.9: Polygonizing an obstacle within the given error range

For this method, if the given error range is small, there will be too many vertices in the polygon and the computation for motion planning will be time consuming. If however the given error is large, the path which is collision free for such polygonizing obstacles may not be collision free for the real situation, since the polygon can not enclose all the obstacle. For this reason, this method is not used. Instead, a polygon with eight vertices will be used to approximate an obstacle. This method will be introduced below.

First, four points on the boundary of the obstacle are selected as vertices of a polygon. These four points are the intersecting points between the boundary of the obstacle and its principal axes (B, D, F and H in Fig. 7.10). These four points will separate the boundary of the object into four curve segments. There is one farthest point on each curve segment from the line segment which links the starting point and the ending point of the curve segment (e. g., point A is one of these point in Fig. 7.10). These four farthest points together with those four intersecting points form the

vertices of a polygon. This is shown in Fig. 7.10 with thick line segments, ABCDEFGH is this polygon. This polygon is enlarged with a certain value as if it is expanded to the configuration space as shown in Fig. 3.4, so that the polygon is the minimum polygon which can enclose the whole obstacle. This polygon is shown in Fig. 7.10 with thin line segments, A'B'C'D'E'F'G'H' is this polygon. This polygon has the following advantages. First, there are not so many vertices in it, so the computation for the polygonization and the motion planning will not be very time consuming. Next, the polygon encloses the whole obstacle, this guarantees that the motion planned with this approximation will be collision free for the real situation.



This is the last polygon to approximate the object

Fig. 7.10: Polygonizing an obstacle by a eight vertices polygon

Fig. 7.11 is an example of polygonizing the obstacle in Fig. 7.3 with this method.

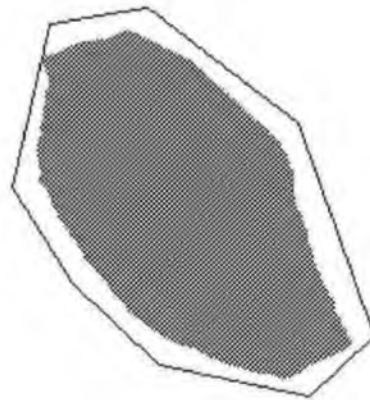


Fig. 7.11: Polygonizing a binary image obstacle

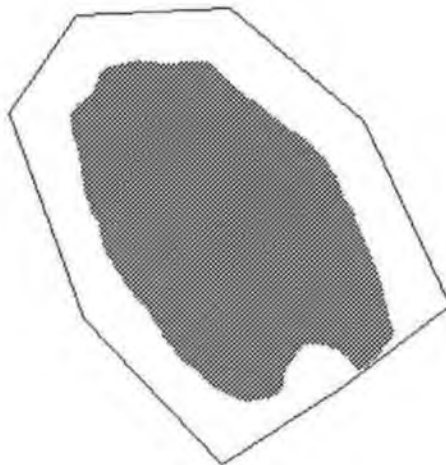


Fig. 7.12: In the case of a sharply changing boundary obstacle

This approach is effective for obstacles whose boundaries do not change sharply. If the boundary of an obstacle changes sharply, as shown in Fig. 7.12, this approach will be very wasteful. The principal rectangle shown in Fig. 7.13 may be better in this case.

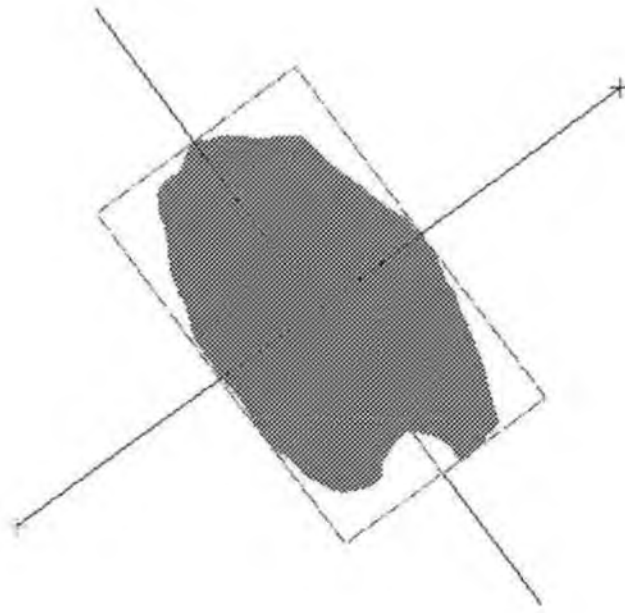


Fig. 7.13: The principal rectangle of the obstacle in Fig. 7.12

7.4 Summary and Discussion

Fuzzy logic based digital image processing and motion planning in binary image representation environments have been studied in this chapter. Experimental results show that the concept of principal index of area coverage proposed in this chapter is a good measure to match objects in consecutive images. It is better than the index of compactness which is used in [Yan, J. 1992] for matching objects in consecutive images. There are cases where the index of compactness fails to match the objects, but the principal index of area coverage can successfully match these cases.

Quadtree decomposition of the environment is an effective approach for geometrical planning in a binary image representation static environment. But it is difficult to extend it to solve dynamic planning in the same environment. Even worse, if the environment is a dynamic one, that is, obstacles in it are moving, space time methods must be used to solve the motion planning problem. As pointed out in the early chapters, the memory required and computation time for the space time approach is

relatively huge. Approximating an obstacle with a polygon is a good alternative in this situation. With obstacles approximated by polygons, the approaches developed in chapters 3, 4 and 5 can be used to solve various motion planning problems in a binary image representation environment without any modification.

Chapter

8. Simulation Programming

8.1 Introduction

Motion planning in various environments has been studied in chapters 3, 4, 5, 6 and 7. This chapter will cover the simulation implementation topics. The simulation is carried out on a personal computer using Borland C++.

There are three simulation programs. The first is the implementation of motion planning (shortest path planning and minimum time path planning) in static environments. This is a DOS application. A window system allows users to edit the simulation environment by using a mouse: adding polygonal obstacles, deleting obstacles, moving obstacles, enlarging obstacles and shrinking obstacles. Users can also select the starting point and destination point by the mouse. After editing the obstacles and selecting the starting and destination points, the path search can be started by selecting the starting button.

The second program is the implementation of motion planning in dynamic environments. This part is programmed as a Microsoft Windows application because animation using BGI (Borland Graphics Interface) is faster with Windows.

The last program performs digital image processing. This part is implemented as a DOS application using Turbo Vision. Digital image processing is programmed based on the commercial software package Mavis.

8.2 Simulation Programming for Static Environments

8.2.1 A Special Window System for DOS Applications

This part will introduce the special window system for DOS applications. The structure of this special window system is designed based on the inheritance property of C++. The overall structure of this system is illustrated in Fig. 8.1. It can be divided into four layers: screen device layer, framing layer, event driver layer and application layer. Every layer is composed of some objects (classes). The base class of the screen device layer is the Rectangle Class: Rect. There are six members and eight member functions in this base class. The six members are the upper left-hand corner co-ordinates, the lower right-hand corner co-ordinates and the width and height of the rectangle respectively.

Class Rso is the Rectangular Screen Object class. It is derived from class Rect and is an abstract class.

To show text on the rectangle, a virtual screen (or text buffer) class must be designed. This class provides low-level screen support for the Text-based Rectangular Screen Object class Trso.

Class Trso is derived from virtual class Rso with class TxBuff as its member. Trso supports the display of text in a rectangle region on the screen.

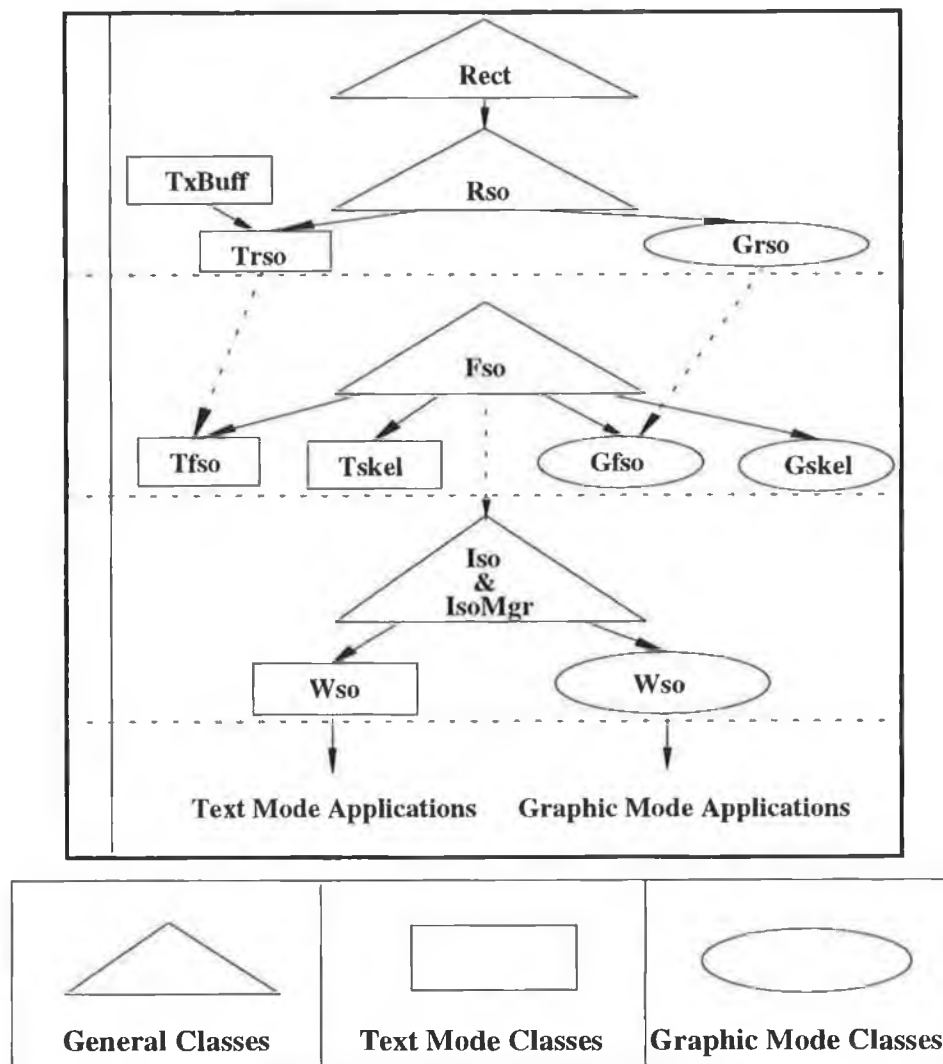


Fig. 8.1: Overall structure of the special window system

Class Trso only supports the display of text on a rectangle for text mode. It does not support text display and graphics for graphics mode. The class which supports graphics mode for screen device layer must be designed. This class is the graphics-based rectangular screen object, class Grso.

The next layer of the window system is the Framed Screen Object, class Fso. A framed screen object consists of three rectangular screen objects: overall, frame and

interior. This is shown in Fig. 8.2. Like class Rso, class Fso is also an abstract class. Defining an abstract class is helpful to define a base protocol which supports text mode and graphics mode. Text-based Frame Screen Object class Tfso and Graphic-based Frame Screen Object class Gfso will be derived from Fso.

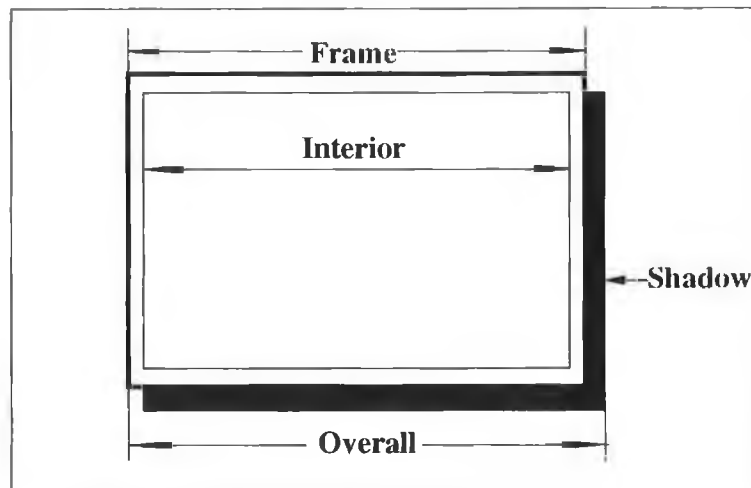


Fig. 8.2: Composition of a framed screen object

The next layer of the window system is the event driver layer. This layer is the most complicated and important layer of the system. First, two abstract classes, interactive screen object class Iso and its manager class IsoMgr are defined. These two classes can be derived to create special classes for text mode and graphics mode applications.

With the text window class or the graphic window class Wso defined in the event driver layer, the menu system classes and button class can be derived from Wso. The overall structure of the menu system classes is shown in Fig. 8.3. The top layer of the system is class Wso. The second layer includes Menu Entry Screen Object class Meso, Menu Screen Object class Mso and Pull-down Menu Screen Object class Pmso. The third layer includes Pull-down Menu Entry Screen Object class Pmeso, Dropped Menu Screen Object class Dmso and Pull Down Bar class PullDnBar. The relations between these classes can be illustrated in Fig. 8.4. These classes can be used in text mode or graphics mode. If graphics mode is used, "GRAPHICS" must be

defined when the source codes are compiled. This is realised by the following compiling statements.

```
#ifdef GRAPHICS
#include "gwsounit.h"
#else
#include "twsounit.h"
#endif
```

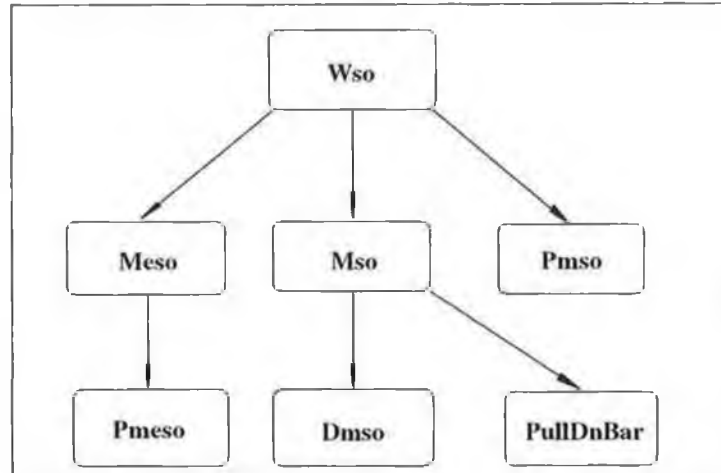


Fig. 8.3: Overall structure of the menu system classes

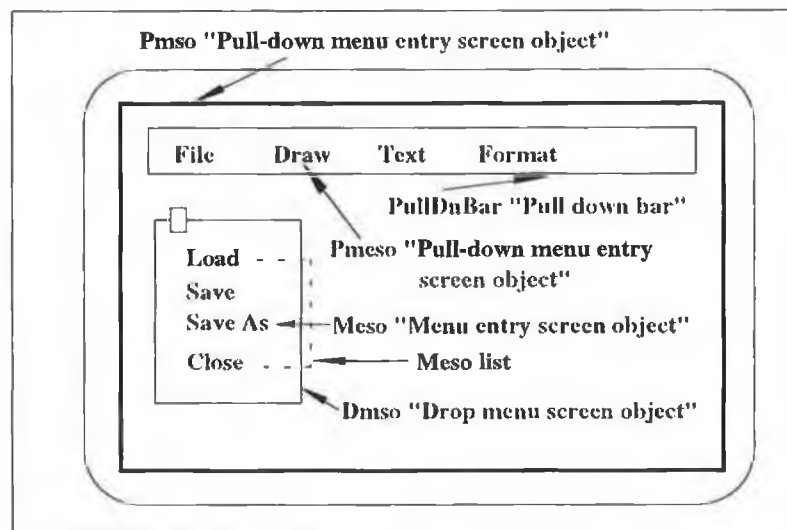


Fig. 8.4: Composition of the pull-down menu system

The button classes can be defined in a similar way. There are two types of buttons. One is the text button, another is the icon button.

The header files for defining these classes are given in Appendix A.

8.2.2 *Simulation System Design*

With the window system and menu system designed above, it is straightforward to design the simulation system for static environments. The menu system of the simulation system is shown in Fig. 8.5, where (a) is the pull-down bar, (b) is the drop menu of pull-down menu entry “Environment”, (c) is the drop menu of pull-down menu entry “Motion”.

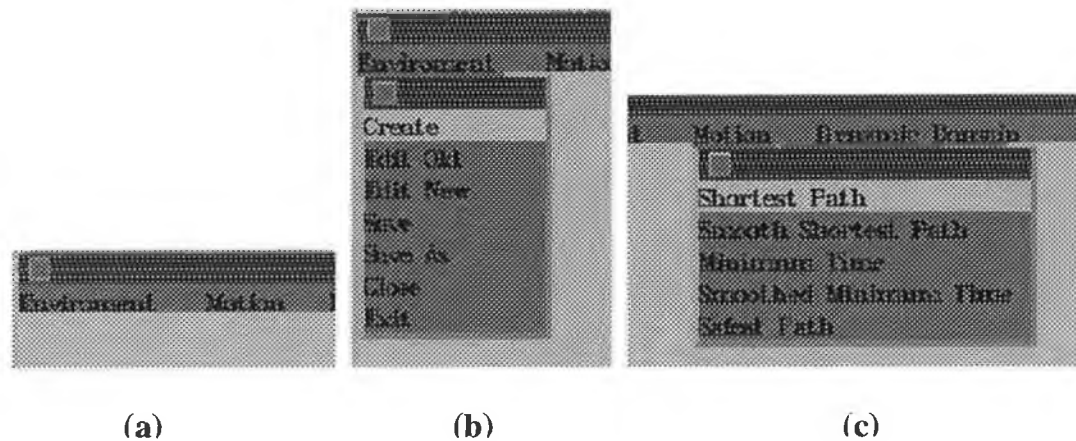


Fig. 8.5: Menu system for the simulation system

When a menu entry of the drop menu is selected, a window with buttons pops up. This is shown in Fig. 8.6, where (a) is the popped window of menu entry “Create”, (b) is the popped window of menu entry “Shortest Path”.

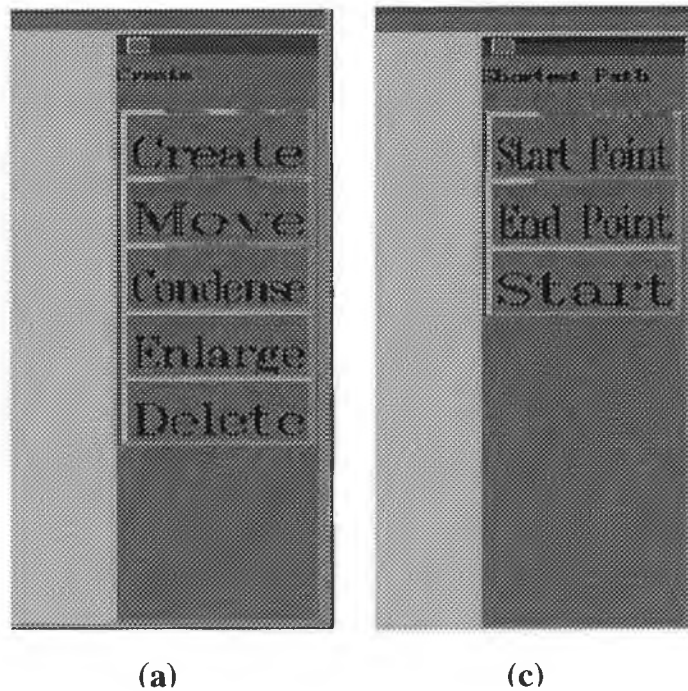


Fig. 8.6: Buttons in the popped windows

“Create” button in Fig. 8.6 (a) is for creating a new polygonal obstacle in the planning environment, “Move” button is for moving a polygonal obstacle to a new position, “Condense” button is for shrinking a polygonal obstacle (because it is too big), “Enlarge” button is for enlarging a polygonal obstacle (because it is too small), “Delete” button is for deleting a polygonal obstacle from the planning environment. “Start Point” button in Fig. 8.5 (b) is for selecting the starting point of the robot in the environment, “End Point” button is for selecting the destination point of the robot in the environment, “Start” button is for starting the search for the path from the starting point to the destination point. The simulation results can be seen in Fig. 3.11 and Fig. 3.16.

The system works in graphics mode. It takes a lot of memory to handle the menu system. When the minimum time path is searched, memory is not enough if there are too many obstacles in the environment. So the minimum time path planning part is immigrated to Windows as a Windows application. The simulation results can be seen in Fig. 4.8 and Fig. 4.12.

8.3 Simulation Programming for Dynamic Environments

The system designed for static environments can be used for simulation in dynamic environments. But there are some disadvantages with the system designed above. The first is that the system in graphics mode requires a lot of memory. The second is that graphic animation in BGI is very slow and does not give high quality animation. For these reasons the simulation system for dynamic environments has been designed as a Windows application, since using bitmaps can generate fast animation.

8.3.1 The Moving Object Class “CSprite”

Class CSprite can generate high quality animation in Windows by using bitmaps. Its definition can be shown in table 8.1 by its header file.

```
//  
//    SPRITE.H:    Header    file    for    the    CSprite    class.  
//  
#ifndef SPRITE_H  
#define SPRITE_H  
  
class CSprite  
{  
    int mHeight;  
    HBITMAP mHImage;  
    HBITMAP mHMask;  
    HBITMAP mHSave;  
    int mWidth;  
    int mX;  
    int mY;  
  
public:  
    CSprite ();  
    ~CSprite ();  
  
    void GetCoord (RECT *Rect);  
    BOOL Hide (HDC HDc);  
    BOOL Initialize (HBITMAP HMask, HBITMAP HImage);  
    BOOL MoveTo (HDC HDc, int X, int Y);  
    BOOL Redraw (HDC HDc);  
    BOOL Start (HDC HDc, int X, int Y);  
};  
#endif
```

Table 8.1: Header file of class CSprite

The moving object is stored as two bitmap files. Fig. 8.16 is an example of such an object, where (a) is the image of the moving object with black backgrounds, (b) is the mask of the moving object with the object being black and the backgrounds being white.

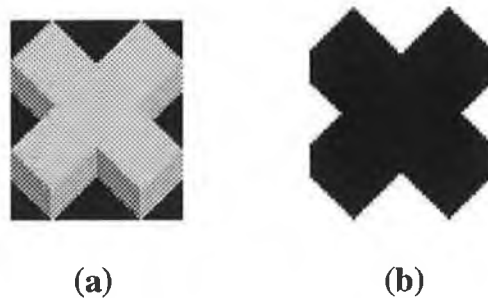
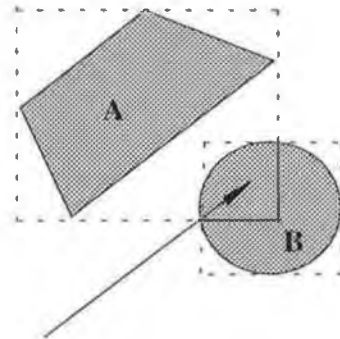


Fig. 8.6: Example of the bitmap files of a moving object

When the object is drawn at a certain location on the screen, the bitmap of the background at that location is stored and the mask is drawn on the screen using boolean “AND” operation between the background and the mask, finally the image is drawn on the screen using boolean “XOR” operation between the background and the image. When the object is drawn at another location, the background of the old location is restored by drawing the saved background bitmap. This process can generate very high quality motion pictures if there is only one moving object on the screen.

If there are more than one moving objects, there will be a problem for the above class CSprite. This can be shown in Fig. 8.7. Suppose there are two moving objects on the screen. Both objects do not collide (overlap) with each other. When the backgrounds of object A at the location shown in Fig. 8.7 is saved, part of object B will be saved as A’s backgrounds, since the backgrounds is saved as a bitmap. When both objects move to new positions, part of object B will be restored as backgrounds. This result looks as if both objects collide (overlap) with each other at that position. This problem is solved by saving the background of an object as two bitmaps (image and mask) as if the background is a moving object as shown in Fig. 8.6. When the background of the object’s old position is restored, the mask bitmap and the image

bitmap are drawn on the screen with the same process of drawing an object. The original source codes and the revised source codes of class CSprite are given in Appendix B.



This part of B will be saved as background of A

Fig. 8.7: The bitmaps of two objects overlap

With this modification of the original class CSprite, if two objects do not really collide each other, the backgrounds can be restored. If two objects do collide each other, part of one object will be restored on the backgrounds. This is good for checking that whether two objects collide or not. The animation result will be shown later in this section.

8.3.2 Obstacle Object Class “Polygon”

The class CSprite discussed above must use bitmaps to display an object on the screen. The straightforward way to get the bitmap is using Borland Workshop or Windows’ Paintbrush to create the object’s bitmap. However, this makes it difficult for the planner to get the obstacle’s exact size information. So the polygonal obstacles must be created in the same Windows application in a similar manner used in the simulation system for static environments. The bitmap of each polygonal obstacle must be created by the same application. The class Polygon is designed for this purpose. Its header file is shown in table 8.2. There are only geometrical parameters about a polygonal obstacle in this definition, so this class will serve as the base class

for the planner's programming where motion parameters will be added to derive a moving polygonal obstacle class.

```
#ifndef OWNPOLYGON_H
#define OWNPOLYGON_H
#ifndef __PTRECT_H
# include "PTRECT.H"
#endif
class OwnPolygon
{
private:
    int vertNumber;
    POINT *vertCoordinate;
public:
    OwnPolygon(int N);
    OwnPolygon(int N, POINT *point);
    OwnPolygon(OwnPolygon const &);
    virtual ~OwnPolygon();
    OwnPolygon const & operator=(OwnPolygon const &);
    virtual int getvertNumber() const;
    virtual POINT getvertCoord(int index);
    virtual int calbitmapWidth();
    virtual int calbitmapHeight();
    virtual int ptsrslttoPolygon(POINT point);
    virtual BOOL ifptinPolygon(POINT point);
    virtual int calbmpbyteSize();
    virtual void calimgBitmap(BYTE * imgBitmap, BYTE * mskBitmap,
int color);
    virtual RECT calRect();
    virtual BOOL CrossWithTheLine(float X1, float Y1, float X2,
float Y2);
#ifndef __WINDOWS_H /* check if windows.h is included */
    BOOL BGIShow;
    virtual void BGIDraw();
    virtual void BGIHide();
#else
    virtual void GDIDraw(HDC hdc);
#endif
    POINT orgCoordinate;
    POINT *relativevertCoord;
};
#endif
```

Table 8.2: Header file of class Polygon

Member function `calimgBitmap( BYTE *imgBitmap, BYTE *mskBitmap, int color)` will generate two pointers to two BYTE arrays, `imgBitmap` and `mskBitmap`. Arrays `imgBitmap` and `mskBitmap` will be used to create the bitmaps of the image and the mask of a polygonal obstacle respectively. Member functions `calbitmapWidth()` and `calbitmapHeight()` return the width and height of the bitmap respectively. Member function `calbmpbyteSize()` returns the size of arrays `imgBitmap` and `mskBitmap` in bytes.

8.3.3 Creating the Bitmap of a Polygonal Obstacle

The function call:

hBitmap=CreateDIBitmap(hdc, &bmih, CBM_INIT, lpBits, &bmi, wUsage)

creates an initialised bitmap. The ***lpBits*** parameter is a pointer to the array of bits which is imgBitmap or mskBitmap created by class Polygon. ***&bmih*** is a pointer to a BITMAPINFOHEADER structure, ***&bmi*** is a pointer to an initialised BITMAPINFO structure. The BITMAPINFOHEADER and BITMAPINFO structures are defined as follows:

```
typedef struct tagBITMAPINFOHEADER
{
    DWORD    biSize;
    LONG     biWidth;
    LONG     biHeight;
    WORD     biPlanes;
    WORD     biBitCount;
    DWORD    biCompression;
    DWORD    biSizeImage;
    LONG     biXPelsPerMeter;
    LONG     biYPelsPerMeter;
    DWORD    biClrUsed;
    DWORD    biClrImportant;
} BITMAPINFOHEADER;

typedef struct tagBITMAPINFO
{
    BITMAPINFOHEADER bmiHeader;
    RGBQUAD          bmiColors[1];
} BITMAPINFO;
```

The first field of structure BITMAPINFO is an initialised BITMAPINFOHEADER structure, the second field is an array of initialised RGBQUAD structures that define the color table. Structure RGBQUAD is defined as follows:

```
typedef struct tagRGBQUAD
{
    BYTE    rgbBlue;
    BYTE    rgbGreen;
    BYTE    rgbRed;
    BYTE    rgbReserved;
} RGBQUAD;
```

All these three structures are defined in Borland C++ system included header file "windows.h". Since there is only one element in the RGBQUAD array, we define the full bitmap information structure, BITMAP16, for 16 colors. BITMAP16 is defined as follows:

```
struct BITMAP16 {
    BITMAPINFO bmi;
    RGBQUAD OTHER15[15];
};
```

The C++ program shown in table 8.3 is the program used to generate the bitmaps of all the polygonal obstacles in the environment.

```
BITMAP16 bMiFull=
{
    {
        {0x28, 0x08, 0x08,
         0x01, 0x04, 0x00,
         0x00, 0x00, 0x00,
         0x00, 0x10},
        //RGBQUAD bmiColor[]=
        { {0x00, 0x00, 0x00, 0x00} }
    },
    {
        {0x00, 0x00, 0x80, 0x00},
        {0x00, 0x80, 0x00, 0x00},
        {0x00, 0x80, 0x80, 0x00},
        {0x80, 0x00, 0x00, 0x00},
        {0x80, 0x00, 0x80, 0x00},
        {0x80, 0x80, 0x00, 0x00},
        {0x80, 0x80, 0x80, 0x00},
        {0xC0, 0xC0, 0xC0, 0x00},
        {0x00, 0x00, 0xFF, 0x00},
        {0x00, 0xFF, 0x00, 0x00},
        {0x00, 0xFF, 0xFF, 0x00},
        {0xFF, 0x00, 0x00, 0x00},
        {0xFF, 0x00, 0xFF, 0x00},
        {0xFF, 0xFF, 0x00, 0x00},
        {0xFF, 0xFF, 0xFF, 0x00}
    }
};

for(int i=0; i < polygonNumber; i++)
{
    OwnPolygon *po;
    po=new OwnPolygon(polygonVertices[i], polygonPoint[i]);

    int w=po->calbitmapWidth()+1;
    int h=po->calbitmapHeight()+1;
    if((w % 2) !=0)
        w=w+1;
    int w1=w/2+4-((w/2) % 4);
    w=2*w1;

    bMiFull.bmi.bmiHeader.biWidth=w;
    bMiFull.bmi.bmiHeader.biHeight=h;

    int N=po->calbmpbyteSize();
    byMask=new BYTE[N];
    byBits=new BYTE[N];
```

```

int clr=3;
po->calingBitmap(byBits, byMask, clr+i);

delete po;
Hmask[i] = CreateDIBitmap(HWinDC, &bMiFull.bMi.bmiHeader,
CBM_INIT, byMask, &bMiFull.bMi, DIB_RGB_COLORS);
if(HMask[i]==NULL)
    MessageBox(wHandle, "Can not create bitmap", "WM_COMMAND",
                MB_OK);
Himage[i] = CreateDIBitmap(HWinDC, &bMiFull.bMi.bmiHeader,
CBM_INIT, byBits, &bMiFull.bMi, DIB_RGB_COLORS);

if(HImage[i]==NULL)
    MessageBox(wHandle, "Can not create bitmap", "WM_COMMAND",
                MB_OK);

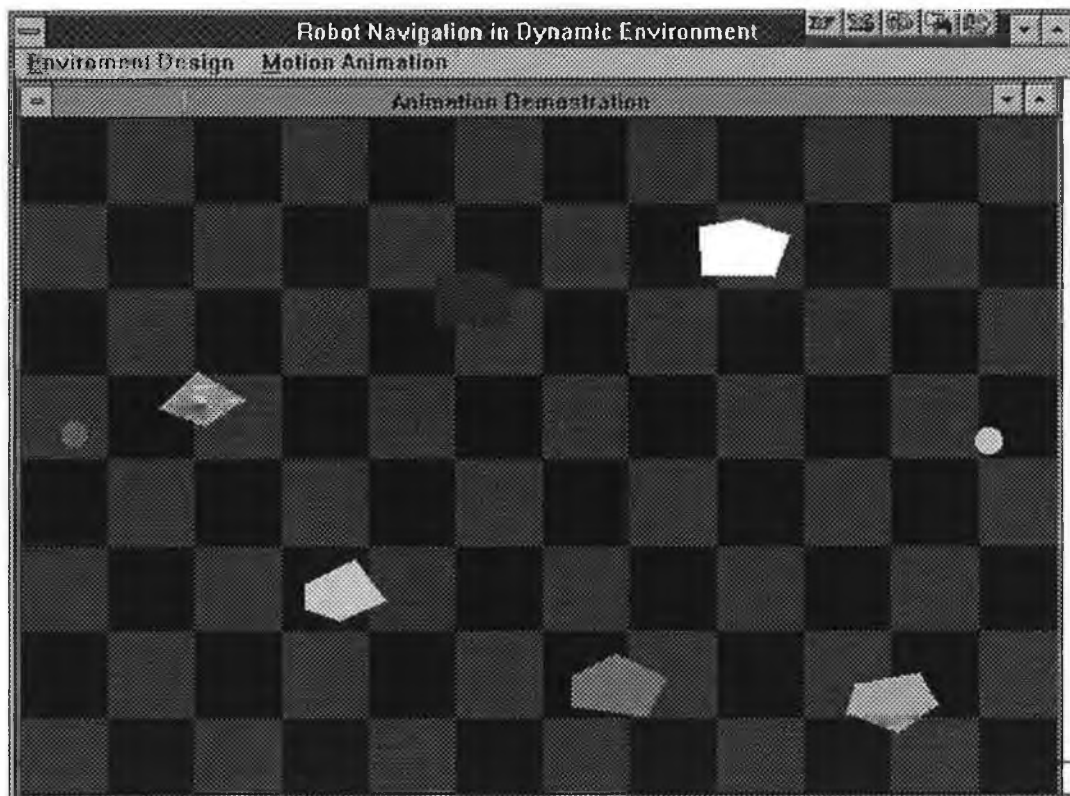
delete byMask;
delete byBits;
}
ReleaseDC (wHandle, HWinDC);

```

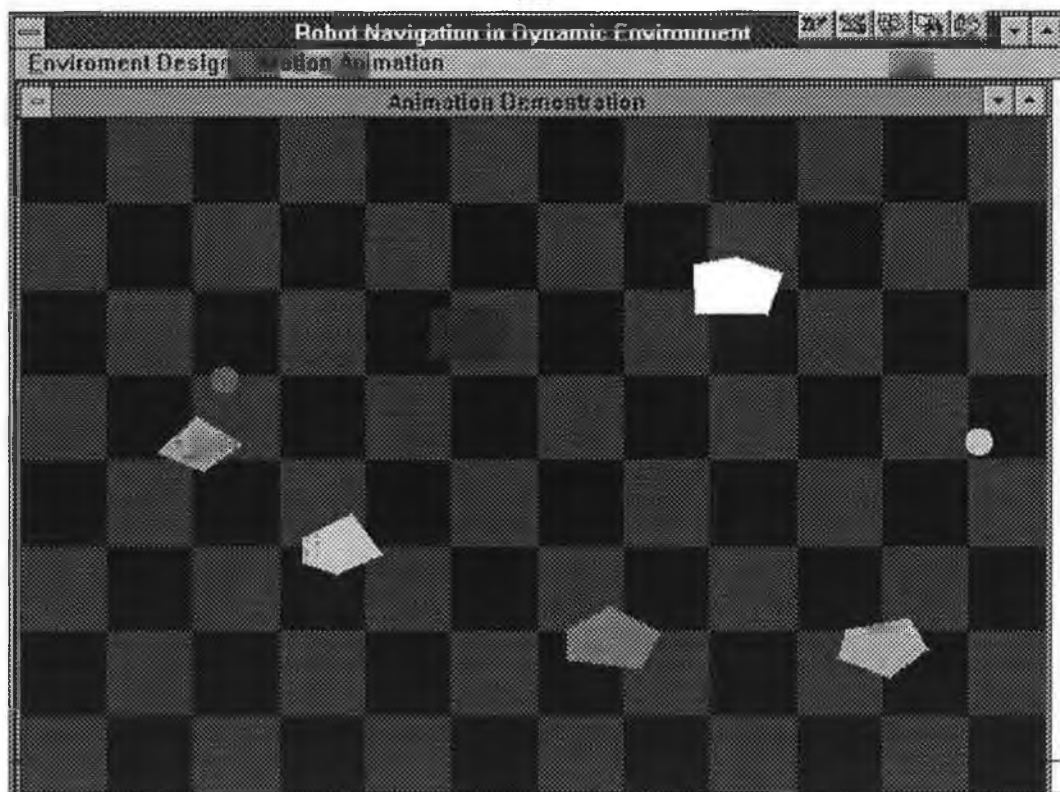
Table 8.3: C++ program which creates the bitmaps of obstacles

8.3.4 Simulation System Design

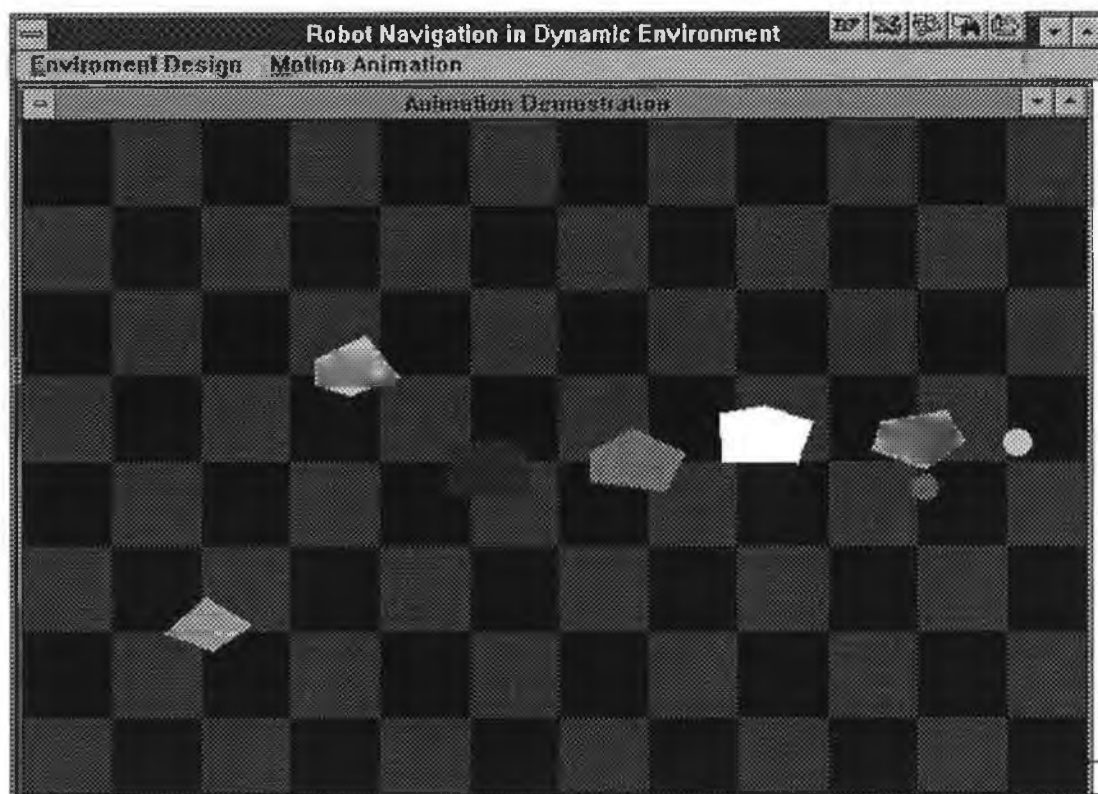
The simulation system for dynamic environments is designed as a Windows application. The menu of the system is similar to that of the system for static environments. When the “Position Setup” item is selected, the obstacles and the starting position and destination position of the robot can be moved using the mouse. When the “Speed Setup” item is selected, a window pops up, allowing the speed of each obstacle or the robot to be changed.



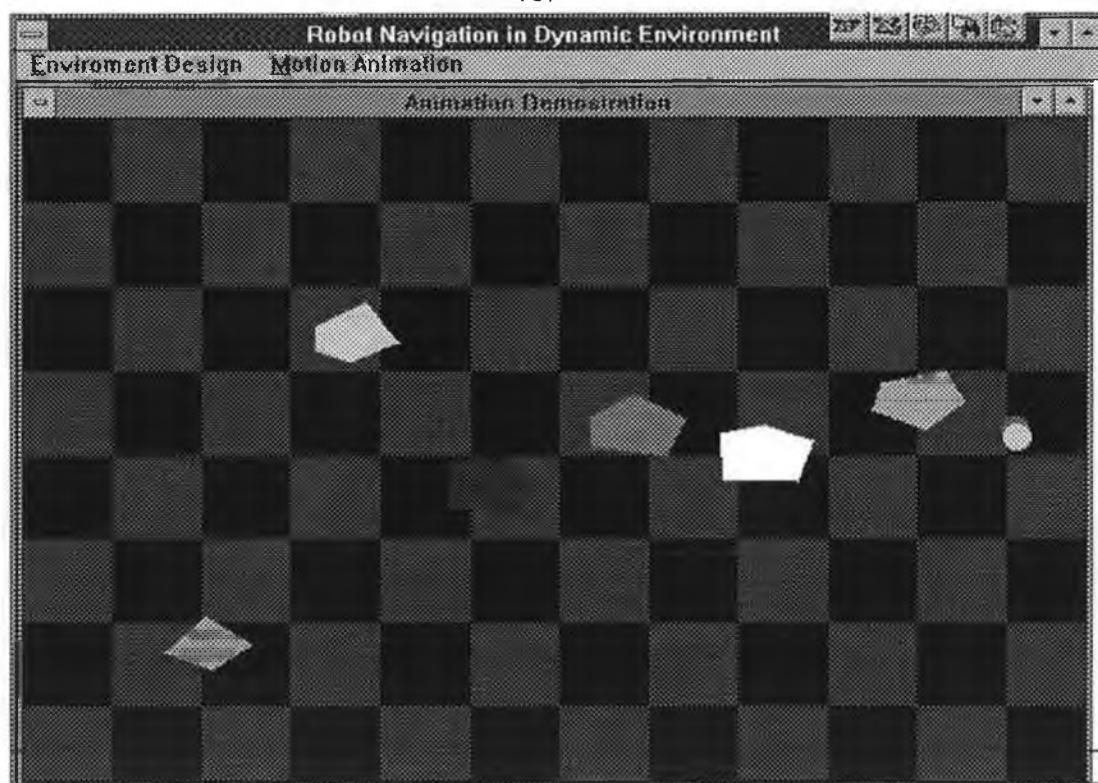
(a)



(b)



(c)



(d)

Fig. 8.8: Animation of the result shown in Fig. 5.7

The simulation results of the system can be seen in Fig. 5.2 and Fig. 5.7. Fig 8.8 is the animation of the result shown in Fig. 5.7.

8.4 Digital Image Processing Programming

This section is simple. Turbo Vision of Borland C++ is used to design the menu system for digital image processing. The menu system can be shown in Fig. 8.9, where (a) is the menu items for picture grabbing and saving, (b) is the menu items for image processing.

Picture	Processing
Init_Camera Alt-I	Smooth Alt-M
Clear_Monitor Alt-C	Histogram Alt-H
Grab_Picture Alt-G	Edge Detection Alt-E
Snap_Picture Alt-S	Threshold Alt-T
Save_Picture Alt-S	IOAC_Calculate Alt-O
Load_Picture Alt-L	Draw On Screen Alt-D
Exit Alt-X	Polygonize Alt-y

(a)
(b)

Fig. 8.9: Menu system for digital image processing

The image processing programming is based on the commercial software package Mavis. The results can be seen in Fig. 7.3, Fig. 7.4, Fig. 7.7 and Fig. 7.11.

Chapter

9. Conclusions

9.1 Conclusions

Motion planning for mobile robots in various environments has been studied in this thesis. The results of the research can be summarised as follows.

1. Configuration space and visibility graph are the basic tools for shortest path and minimum time path planning in static environments. They are also the basis for motion planning in dynamic environments and binary image representation environments. The robot can be considered as a point in the configuration space. This simplifies the problems.
2. For dynamic planning in static environments, there is no analytical solution to the problem. But the approximate minimum time path can be planned based on the Minimum Time Path at Visibility Nodes (MTPVN). With the results of optimum speed trajectory along a line segment path, the production rules for generating

speed trajectories for MTPVN are constructed. The introduction of the point of invariance (POI) of a node on the visibility graph solves the inefficient depth first search problem. Search in the visibility graph can generate the minimum time path at visibility nodes.

3. A new approach which is more practical for car-like mobile robots has been proposed for planning the minimum time path based on smoothing the path at each turning corner (a node on the visibility graph). The minimum time path is composed of line segments and arc segments. The speed for the robot to move along an arc is determined by the radius of the arc and the clearance between the robot and the obstacles in the environment. There is no need to introduce the concept of the point of invariance (POI) of a node in this approach. This approach provides an alternative for minimum time path planning in static environments and can generate the safer path.
4. The accessibility graph plays an important role in motion planning in dynamic environments. It is an extension of the visibility graph to dynamic environments. It has advantages over the space time approaches in that it solves the “when” and “where” problems at the same time. So it can generate the minimum time motion for massless robots. It takes $O(n^2 \cdot \log(n))$ time for the accessibility graph approach to generate a minimum time motion for a massless mobile robot, where n is the number of total vertices of the polygonal obstacles in the environment.
5. The concept of the accessibility polygon is proposed in this thesis. Based on this concept, the accessibility graph can be constructed in a similar way as the visibility graph is constructed. The concepts of the improved accessibility polygon and the improved accessibility graph are proposed to solve the motion planning problems for mobile robots with mass (that is, it takes time for the robot to accelerate from zero speed to its maximum speed). The space time approach has been used up until now. This converts a two dimensional problem into a three dimensional problem with time added as another dimension. This requires exponential memory and exponential time. While the improved

accessibility graph approach developed in this research can generate a minimum time motion in $O((n+k)^2 \cdot \log(n+k))$ time, where n is the number of total vertices of polygonal obstacles in the environment and k is the number of obstacles in the environment. This is a great improvement on the space time approach. The improved accessibility graph approach can be extended to solve some special cases in two dimensional environments. It can also be extended to solve the motion planning problems in three dimensional dynamic environments where space time will become four dimensions and will be difficult to represent in geometrical space.

6. Motion planning in three dimensional environments has also been studied in this thesis. Based on the concept of three dimensional visibility graph and the properties of a minimum time motion in three dimensional dynamic environments, the concepts of the accessibility polyhedron and the three dimensional accessibility graph are proposed. An approach based on the three dimensional accessibility graph is proposed for motion planning in three dimensional dynamic environments for massless robots. This approach can also be extended to solve motion planning problems for mobile robots with mass in three dimensional dynamic environments.
7. When cameras are used as sensors for mobile robots, a motion planning environment will be represented as a digital image. The image can be processed to represent the obstacle region as 1 and the background region as 0. Such an environment is called a binary image representation environment. Fuzzy logic based image processing has been studied in this thesis. The concept of fuzzy Principal Index Of Area Coverage (PIOAC) of an object in an image is proposed to recognise and match objects in consecutive images. Experimental results show that PIOAC is a good measure to recognise and match objects in consecutive images.
8. Visibility graph approach in binary image representation environments is very inefficient, since there are so many vertices in an obstacle. Motion planning

approach usually used in binary image representation environments is the quadtree approach. It has disadvantages in that it is difficult to transform the obstacles into the robot's configuration space and the space time approach must be used in dynamic environments. Polygonizing the obstacles is proposed in this research so that all the approaches developed can be used to solve various motion planning problems in binary image representation environments without any modification. Two methods of polygonizing an obstacle are implemented. One is using the principal rectangle of an obstacle to approximate the obstacle. Another is using a 8 vertices polygon to approximate an obstacle.

9.2 Recommendations for Further Research

The research work in this thesis suggests a number of problems for future investigation:

1. Finding more efficient algorithms for minimum time path planning in static environments and studying the error between the planned path and the real minimum time path. We have shown that the minimum time path planned is only approximate. Applying dynamic programming seems to be a good research direction. The problem is to make it efficient.
2. Heuristics for searching for a path in accessibility graph for dynamic environments. The density of the obstacles might be used to determine the choice of a path.
3. Motion planning in dynamic environments with rotating moving obstacles. The research in this thesis assumes that the obstacles do not rotate. The problem seems to be more difficult when obstacles can rotate.

4. Motion planning when the orientation of the robot must be taken into consideration in dynamic environments, that is, minimum time motion planning in higher dimensional configuration space.
5. Planning safe motion in the presence of moving obstacles. In dynamic environments, a motion may have to be time-minimal to be safe. However, the environment may not be so critical to the robot. In such a situation, the robot may want some clearance between the obstacles, since the minimum time motion planned is not very safe in general in the sense that it brings the robot close to the obstacles. Minimum safety may be guaranteed by adding some safe time to the improved accessibility polygon. It would be useful to generate safer motion without sacrificing too much time.
6. Planning shortest path in dynamic environments. There is little research on shortest path in dynamic environments up to date. Sometimes it is useful. For example, for a person who carries heavy suitcases, a shortest path may be more important than a minimum time path.
7. Planning amidst programmed obstacles. This is useful for environments with more than one robot. In such an environment, other mobile robots can be considered as programmed automata - "intelligent obstacles".
8. Implementing the accessibility graph algorithm for three dimensional dynamic environments. It might require very hard work to do this.
9. Finding more efficient algorithms for motion planning in binary image representation environments. This is important since more intelligent robots will use cameras as sensors.

References

[Aggarwal, J. 1988]

J. K. Aggarwal and N. Nandhakuwar, On the Computation of Motion from Sequences of Images - A Review, Proceedings of the IEEE, Vol. 76, No. 8, August, 1988.

[Aggarwal, J. 1990]

J. K. Aggarwal and Bikash Sabata, Motion from a Sequence of Range Images, Proceedings of the IEEE International Workshop on Intelligent Motion Control, Istanbul, 20-22 August 1990.

[Akishita, Sadao 1990]

Sadao Akishita, Sadao Kawamura and Kei-ichi Hayashi, New Navigation Function Utilising Hydrodynamic Potential for Mobile Robot, Proceedings of the IEEE International Workshop on Intelligent Motion Control, pp. 289-294, Istanbul, 20-22 August 1990.

[Akman, V. 1987]

V. Akman, Unobstructed shortest paths in polyhedral environments, Lecture Note in Computer Science 251, Springer-Verlag, 1987.

[Anderson, Tracy 1990]

Tracy L. Anderson and Max Donath, Autonomous Robots and Emergent Behaviour: A Set of Primitive Behaviours for Mobile Robot Control, Proceedings of IEEE International Workshop on Intelligent Robots and System IROS'90, 1990.

[Arkin Ronald 1990a]

Ronald C. Arkin, Integrating Behavioural, Perceptual, and World Knowledge in Reactive Navigation, Robotics and Autonomous Systems, Vol. 6, pp. 105-122, 1990.

[Arkin, Ronald 1990b]

Ronald C. Arkin and Daryl T. Lawton, Reactive Behavioural Support for Qualitative Visual Navigation, Proceedings of the IEEE International Workshop on Intelligent Motion Control, Istanbul, 20-22 August 1990.

- [Asano, Takao 1986]
Takao Asano, Tetsuo Asano, Leonidas Guibas and John Hershberger, Visibility of Disjoint Polygons, *Algorithmica* 1, pp. 49-63, 1986.
- [Asano, Takao 1987]
Takao Asano, Tetsuo Asano and Hiroshi Imai, Shortest Path between two Simple Polygons, *Information processing Letters* 24, 285-288, 1987.
- [Avnaim, F. 1988]
F. Avnaim, J. D. Boissonnat and B. Faverjon, A Practical Exact Motion Planning Algorithm for Polygonal Objects Amidst Polygonal Obstacles, Technical Report No. 890, INRIA, Sophia-Antipolis, France, 1988.
- [Ayala, D. 1985]
D. Ayala, P. Brunet, R. Juan and I. Navazo, Object Representation by Means of Nonminimal Division Quadrees and Octrees, *ACM Transactions on Graphics*, Vol. 4, No. 1, pp. 41-59, January 1985.
- [Bajai, C. 1990]
C. Bajai and M. S. Kim, Generation of Configuration Space Obstacles: The Case of Moving Algebraic Surfaces, *International Journal of Robotics Research*, Vol. 9, pp. 92-112, 1990.
- [Baltsan, Avikan 1988]
Avikan Baltsan and Micha Sharir, On the Shortest Paths between Two Convex Polygons, *Journal of the Association for Computing Machinery*, Vol. 35, No. 2, pp. 267-287, April 1988.
- [Barraquand, Jerome 1990]
Jerome Barraquand and Jean-Claude Latombe, A Monte-Carlo Algorithm for Path Planning With Many Degrees of Freedom, *Proceedings of the IEEE International Conference on Robotics and Automation*, Cincinnati (Ohio) 1990.
- [Bhanu, B. 1991]
B. Bhanu and W. Burger, A Qualitative Approach to Dynamic Scene Understanding, *CVGIP: Image Understanding*, Vol. 54, No. 2, Sept. 1991, pp. 184-205.
- [Bhanu, B. 1992]
B. Bhanu, Dynamic-scene and Motion Analysis Using Passive Sensors, *IEEE Expert*, Feb. 1992, pp. 45-64.
- [Bhattacharya, B. 1988]
B. K. Bhattacharya and J. Zorbas, Solving the Two-Dimensional Findpath Problem Using a Line-Triangle Representation of the Robot, *Journal of Algorithms* 9, pp. 449-469, 1988.
- [Blake, A. 1991]

A. Blake et al. Visual Navigation around Curved Obstacles, Proceedings of the 1991 IEEE International Conference on Robotics and Automation, Sacramento, California, April 1991.

[Bolles, Robert 1985]

Robert C. Bolles and H. Harlyn Baker, Epipolar-Plane Image Analysis: A Technique for Analysing Motion Sequences, Proceedings of the 3rd workshop on Computer Vision Representation and Control, Bellaire, 1985.

[Borenstein, Johann 1988]

Johann Borenstein and Yoram Koren, Obstacle Avoidance with Ultrasonic Sensors, IEEE Journal of Robotics and Automation, Vol. 4, No. 2, April 1988.

[Briggs, Amy 1992]

Amy J. Briggs, An Efficient Algorithm for one-step Planner Compliant Motion Planning with Uncertainty, Algorithmica 8, pp. 195-208, 1992.

[Brost, Randy 1991]

Randy C. Brost, Computing the Possible Rest Configurations of Two Interacting Polygons, Proceedings of the 1991 IEEE International Conference on Robotics and Automation, Sacramento, California, April 1991.

[Cameron, S. 1985]

S. A. Cameron, A Study of the Clash Detection Problem in Robotics, Proceedings of the IEEE International Conference on Robotics and Automation, pp. 488-493, 1985.

[Cameron, S. 1990]

S. A. Cameron, Collision Detection by Four Dimensional Intersection Testing, IEEE Transactions on Robotics and Automation, Vol. 6, pp. 291-302, 1990.

[Canny, J. 1988a]

J. F. Canny and B. R. Donald, Simplified Voronoi Diagrams, Discrete and Computational Geometry, Vol. 3, pp. 219-236, Springer-Verlag, 1988.

[Canny, J. 1988b]

J. F. Canny, The Complexity of Robot Motion Planning, MIT Press, 1988.

[Canny, John 1987]

John Canny and John Reif, New Lower Bound Techniques for Robot Motion Planning Problems, Proceedings of the 27th Annual IEEE Symposium on the Foundations of Computer Science, Los Angeles, 1987.

[Canny, John 1988]

John Canny, Some Algebraic and Geometric Computations in PSPACE, Proceedings of the 20th Annual ACM Symposium on Theory of Computing, pp. 460-467, Chicago, 1988.

[Canudas, C. 1991]

C. Canudas de Wit and R. Roskam, Path Following of a 2-DOF Wheeled Mobile Robot under Path and Input Torque Constraints, Proceedings of the 1991 IEEE International Conference on Robotics and Automation, Sacramento, California, April 1991.

[Casals, Alicia 1990]

Alicia Casals, Josep Amat and Antonio B. Martinez, High Speed Processor for Autonomous Vehicles Guidance, Proceedings of the IEEE International Workshop on Intelligent Motion Control, Istanbul, 20-22 August 1990.

[Chan, R. 1991]

R. H. T. Chan, P. K. S. Tam and D. N. K. Leung, Robot Navigation in Unknown Terrains via Multi-resolution Grid Maps, Proceedings of IECON'91: 1991 International Conference on Industrial Electronics, Control and Instruments, Japan, 1991.

[Chen, Pang 1991]

Pang C. Chen and Yong K. Hwang, Practical Path Planning among Movable Obstacles, Proceedings of the 1991 IEEE International Conference on Robotics and Automation, Sacramento, California, April 1991.

[Cheng, R. 1992]

R. M. H. Cheng and Ramesh Rajagopalan, Kinematics of Automatic Guided Vehicles with an Inclined Steering Column and an Offset Distance: Criterion for Existence of Inverse Kinematic Solution, Journal of Robotic Systems 9(8), pp. 1059-1081, 1992.

[Choo, Chang 1991]

Chang Y. Choo, John M. Smith and Nasser M. Nasrabadi, An Effective Terrain Acquisition Algorithm for a Mobile Robot, Proceedings of the 1991 IEEE International Conference on Robotics and Automation, Sacramento, California, April 1991.

[Chuang, Jen-Hui 1991]

Jen-Hui Chuang and Narendra Ahuja, Path Planning Using the Newtonian Potential, Proceedings of the 1991 IEEE International Conference on Robotics and Automation, Sacramento, California, April 1991.

[Ciliz, Kemal 1989]

M. Kemal Ciliz and Can Isik, Fuzzy Rule-based Motion Controller for an Autonomous Mobile Robot, Robotics, Vol. 7, pp. 37-42, 1989.

- [Coffman, E. 1992]
 E. G. Coffman Jr. and E. N. Gilbert, Path through a Maze of Rectangles, Networks, Vol. 22, pp. 349-367, 1992.

- [Conan, Vania 1992]
 Vania Conan, Christopher Leroux and Pascal Aubry, Finding the Safe Stepping Areas for a Walking Mobile Robot with 3-D Vision, Proceedings of ICARCV'92: Second International Conference on Automation, Robotics and Computer Vision, Singapore, 1992.

- [d'Andrea-Novel, B. 1991]
 B. d'Anrea-Novel, G. Bastin and G. Campion, Modelling and Control of Non Holonomic Wheeled Mobile Robots, Proceedings of the 1991 IEEE International Conference on Robotics and Automation, Sacramento, California, April 1991.

- [Dean, Thomas 1990]
 Thomas Dean el. Coping with Uncertainty in a Control System for Navigation and Exploration, Proceedings of the 8th National Conference on Artificial Intelligence, London, July 1990.

- [DeMuth, Gordon 1990]
 Gordon DeMuth and Steve Springsteen, Obstacle Avoidances Using Neural Networks, Proceedings of the Symposium on Autonomous Underwater Vehicle Technology AUV 90, Washington, 5-6 June 1990.

- [Dodds, David 1991]
 David R. Dodds, Coping with Complexity in the Navigation of an Autonomous Mobile Robot, Proceedings of the SPIE - The International Society for Optical Engineering, Vol. 1388, pp. 448-452, Mobile Robot V, 1991.

- [Donald, Bruce 1989]
 Bruce Donald and Patrick Xavier, A Provably Good Approximation Algorithm for Optimal-Time Trajectory Planning, Proceedings of the IEEE International Conference on Robotics and Automation, Scottsdale, pp. 958-963, 1989.

- [Donald, Bruce 1991]
 Bruce R. Donal and Patrick G. Xavier, Time-Safety Trade-offs and a Bang-Bang Algorithm for Kinodynamic Planning, Proceedings of the 1991 IEEE International Conference on Robotics and Automation, Sacramento, California, April 1991.

- [Erdmann, M. 1987]
 M. Erdmann and T. Lozano-Perez, On Multiple Moving Objects, Algorithmica, Vol. 2, pp. 477-522, 1987.

- [Esterling, D. 1983]
D. M. Esterling and Van J. Rosendale, An Intersection Algorithm for Moving Parts, Proceedings of the NASA Symposium on Computer Aided Geometry Modelling, Hampton, pp. 119-123, 1983.
- [Fang, J. 1984]
J. Q. Fang and T. S. Hunag, Some Experiments on Estimating the 3-D Motion Parameters of Rigid Body from Two Consecutive Frames, IEEE Transactions on PAMI, Sept. 1984, pp. 545-554.
- [Faverjon, B. 1984]
B. Faverjon, Obstacle Avoidance Using an Octree in the Configuration Space of a Manipulator, Proceedings of the IEEE International Conference on Robotics and Automation, Atlanta, pp. 504-507, 1984.
- [Feng, Dai 1991]
Dai Feng and Bruce H. Krogh, Dynamic Steering Control of Conventionally Steered Mobile Robot, Journal of Robotic Systems, Vol. 8, No. 5, pp. 699-721, 1991.
- [Feng, J. 1991]
Jie Feng, Wei-Chung Lin and Chin-Tu Chen, Epicardial Boundary Detection Using Fuzzy Reasoning, IEEE Transactions on Medical Imaging, Vol. 10, No. 2, June, 1991.
- [Fennema, Claude 1990]
Claude Fennema and Allen R. Hanson, Experiments in Autonomous Navigation, Proceedings of the IEEE International Workshop on International Motion Control, Istanbul, 20-22 April 1990.
- [Fernandes, C. 1991]
C. Fernandes, L. Gurvits and Z. X. Li, A Variational Approach to Optimal Nonholonomic Motion Planning, Proceedings of the 1991 IEEE International Conference on Robotics and Automation, Sacramento, California, April 1991.
- [Fraichard, Th. 1990]
Th. Fraichard, C. Laugier and G. Lievin, Robot Motion Planning: The Case of Non Holonomic Mobiles in a Dynamic World, Proceedings of IEEE International Workshop on Intelligent Robots and Systems IROS'90, 1990.
- [Fraichard, Th. 1991a]
Th. Fraichard and C. Laugier, On Line Reactive Planning for a Non Holonomic Mobile in a Dynamic World, Proceedings of the 1991 IEEE International Conference on Robotics and Automation, Sacramento, California, April 1991.
- [Fraichard, Th. 1991b]

Th. Fraichard and Lilia Imag-Cnrs, Smooth Trajectory Planning for a Car in Structured World, proceedings of the 1991 IEEE International Conference on Robotics and Automation, Sacramento, California, April 1991.

[Fraichard, Th. 1993a]

Th. Fraichard, Dynamic Trajectory Planning with Dynamic Constraints: a 'State-Time Space' Approach, Proc. of the IEEE/RSJ International Conference on Intelligent Robots and System, Yokohama, Japan. pp. 1393-1400, Jul. 26-30, 1993.

[Fraichard, Th. 1993b]

Th. Fraichard and C. Laugier, Kinodynamic Planning in a Structured and Time-Varying Workspace, Geometric Reasoning for Perception and Action, Lecture Notes in Computer Science 708, Springer-Verlag, 1993.

[Fraichard, Th. 1993c]

Th. Fraichard and C. Laugier, Dynamic Trajectory Planning, Path-Velocity Decomposition and Adjacent Path, Proc. of the 1993 International Joint Conference on Artificial Intelligence, Chambéry, France, Volume 2, pp. 1592-1597, Aug. 29-Sept. 3, 1993.

[Fryxell, R. 1987]

R. C. Fryxell, Navigation Planning Using Quadrees, Oak Ridge National Laboratory TM-10481, Oak Ridge, 1987.

[Fujimura, Kikuo 1991]

Kikuo Fujimura, A Model of Reactive Planning for Multiple Mobile Agents, Proceedings of the 1991 IEEE International Conference on Robotics and Automation, Sacramento, California, April 1991.

[Fujimura, Kikuo 1991]

Kikuo Fujimura, Motion Planning in Dynamic Environments, Springer-Verlag, 1991.

[Fukuda, Toshio 1990]

Toshio Fukuda and Takanori Shibata, Neural Network Applications for Robotic Motion Control, Proceedings of the IEEE International Workshop on Intelligent Motion Control, Istanbul, 20-22 August 1990.

[Gachet, D. 1992]

D. Gachet, M. A. Salichs and L. Moreno, Contour Following Algorithms for a Mobile Robot, Parallel and Distributed Computing in Engineering Systems, 1992.

[Gasos, J. 1990]

J. Gasos, M. C. Garcia-Alegre and R. Garcia Rosa, Fuzzy Local Navigation of a Simulated Real Robot in Unknown Environments, Proceedings of the IEEE

International Workshop on Intelligent Motion Control, Istanbul, 20-22, August 1990.

[Gelin, R. 1992]

R. Galin, ADA: The Ideal Tool for Experimental Robots, Proceedings of ICARCV'92: Second International Conference on Automation, Robotics and Computer Vision, Singapore, 1992.

[Gewali, L. 1990]

L. P. Gewali, S. Natfos and I. G. Tollis, Path Planning in the Presence of Vertical Obstacles, IEEE Transactions on Robotics and Automation, Vol. 6, No. 3, pp. 331-341, June, 1990.

[Ghosh, S. 1987]

S. K. Ghosh and D. M. Mount, An Output-sensitive Algorithm for Computing Visibility Graphs, Proceedings of the 28th Annual IEEE Symposium on the Foundations of Computer Science, Los Angeles, pp. 11-19, 1987.

[Gilbert, E. 1985]

E. G. Gilbert and D. W. Johnson, Distance Functions and their Application to Robot Path Planning in the Presence of Obstacles, IEEE Transactions of Robotics and Automation, RA-1(1), pp. 21-30, 1985.

[Goetcherian, V. 1980]

Vartkes Goetcherian, From Binary to Grey Tone Image Processing Using Fuzzy Logic Concepts, Pattern Recognition, Vol. 12, pp. 7-15, 1980.

[Griswold, N. 1990]

N. C. Griswold and J. Eem, Control for Mobile Robots in the Presence of Moving Obstacles, IEEE Transactions on Robotics and Automation, Vol. 6, pp. 263-268, 1990.

[Hague, Tony 1990]

Tony Hague and Stephen Cameron, Motion Planning for the Oxford AGV, Proceedings of the IEEE International Workshop on Intelligent Motion Control, Istanbul, 20-22 August 1990.

[Hague, Tony 1990]

Tony Hague, Mike Brady and Stephen Cameron, Using Moments to Plan Paths for the Oxford AGV, Proceedings of IEEE International Conference on Robotics and Automation, Ohio, 1990.

[Han, Xian-Guang 1992]

Xian-Guang Han, Yu-Geng Xi and Zhong-Jun Zhang, Irregular Obstacle Avoidance for Conventionally-Steered Mobile Robots, Proceedings of ICARCV'92: Second International Conference on Automation, Robotics and Computer Vision, Singapore, 1992.

- [Harashima, Fumio 1990]
Fumio Harashima, Hideki Hashimoto and Takashi Kubota, Sensor Based Robot Control Systems, Proceedings of the IEEE International Workshop on Intelligent Motion Control, Istanbul, 20-22 August 1990.
- [Harris, C 1989]
C. J. Harris and C. G. Moore, Intelligent Identification and Control for Autonomous Guided Vehicles Using Adaptive Fuzzy-based Algorithms, Eng. Applications of AI, Vol. 2, December, 1989.
- [Hayashi, Akira 1991]
Akira Hayashi and Benjamin J. Kuipers, Planning a Smooth Path for Autonomous Vehicles Using Primary Convex Regions, Knowledge Based Systems and Neural Networks, 1991.
- [Hejase, Ale 1990]
Ale Jose Hejase, Optimal Motion Synthesis of a Mobile Robot, Proceedings of the IEEE International Workshop on Intelligent Motion Control, Istanbul, 20-22, August 1990.
- [Herman, M. 1986]
M. Herman, Fast, Three Dimensional, Collision Free Motion Planning, Proceedings of the IEEE International Conference on Robotics and Automation, Cincinnati, Ohio, pp. 210-215, 1986.
- [Hirzinger, G. 1990]
G. Hirzinger, J. Dietrich and B. Brunner, Multisensory Telerobotic Concepts, Proceedings of the IEEE International Workshop on Intelligent Motion Control, Istanbul, 20-22 August 1990.
- [Ho, Alex 1990]
Alex W. Ho and Geoffrey C. Fox, Neural Network Near-Optimal Motion Planning for A Mobile Robot on Binary and Varied Terrains, Proceedings of IEEE International Workshop on Intelligent Robots and Systems, IROS'90, 1990.
- [Holenstein, Alois 1991]
Alois A. Holenstein and Essam Badreddin, Collision Avoidance in a Behaviour-based Mobile Robot Design, Proceedings of the 1991 IEEE International Conference on Robotics and Automation, Sacramento, California, April 1991.
- [Hopcroft, J. 1984]
J. E. Hopcroft and J. T. Schwartz, On the Complexity of Motion Planning for Multiple Independent Objects: PSPACE Hardness of the "Warehouseman's Problem", International Journal of Robotics Research, Vol. 3, pp. 76-88, 1984.

[Hwang, Y. 1988]

Y. K. Hwang and N. Ahuja, Path Planning Using a Potential Field Representation, Proceedings of the IEEE International Conference on Robotics and Automation,, Philadelphia, PA, pp. 648-649, 1988.

[Hwang, Yong 1992]

Yong K. Hwang and Narendra Ahuja, A Potential Field Approach to Path Planning, IEEE Transactions on Robotics and Automation, Vol. 8, No. 1, February 1992.

[Iida, Shigeki 1990]

Shigeki Iida and Shinichi Yuta, Control of a Vehicle Subsystem for an Autonomous Mobile Robot with Power Wheeled Steering, Proceedings of the IEEE International Workshop on Intelligent Motion Control, Istanbul, 20-22 April 1990.

[Ilari, J. 1990]

J. Ilari and C. Torras, 2D Path Planning: A Configuration Space Heuristic Approach, International Journal of Robotics Research, Vol. 9, pp. 75-91, 1990.

[Jahne, B. 1992]

B. Jahne, Digital Image Processing: Concepts, Algorithms and Scientific Applications, Springer-Verlag, 1992.

[Jarvis, Ray 1992]

Ray Jarvis, An Autonomous Mobile Robot in a Rangepic World, Proceedings of ICARCV'92: Second International Conference on Automation, Robotics and Computer Vision, Singapore, 1992.

[Kamada, Hiroshi 1990]

Hiroshi Kamada, Satoshi Naoi and Toshiyuki Gotoh, A Compact Navigation System Using Image processing and Fuzzy Control, IEEE Proceedings of 1990 South-eastern, 1990.

[Kamada, Hiroshi 1992]

Hiroshi Kamada, Toshiyuki Gotoh and Masumi Yoshida, A Visual Control System Using Image processing and Fuzzy Inference, Systems and Computers in Japan, Vol. 23, No. 3, 1992.

[Kambhampati, S. 1986]

S. Kambhampati and L. S. Davis, Multiresolution Path Planning for Mobile Robots, IEEE Journal of Robotics and Automation, Vol. 2, pp. 46-49, 1986.

[Kanayama, Y. 1988]

Y. Kanayama and G. R. DeHaan, A Mathematical Theory of Safe Path Planning, TRCS88-16, Department of Computer Science, University of California, 1988.

[Kanayama, Y. 1989]

Y. Kanayama, A Mathematical Theory of Object Centred Path Planning, TRCS89-07, Department of Computer Science, University of California, 1989.

[Kant, K. 1986]

K. Kant and S. W. Zucker, Toward Efficient Planning: The Path Velocity Decomposition, International Journal of Robotics Research, Vol. 5, pp. 72-89, 1986.

[Kapoor, S. 1988]

S. Kapoor and S. N. Maheshwari, Efficient Algorithms for Euclidean Shortest Path and Visibility Problem with Polygonal Obstacles, Proceeding of the First ACM Symposium on Computer Geometry, pp. 75-80, Baltimore, 1988.

[Kehtarnavaz N. 1988]

N. Kehtarnavaz and S. Li, A Collision-free Navigation Scheme in the Presence of Moving Obstacles, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 808-813, 1988.

[Kehtarnavaz, N. 1990]

N. Kehtarnavaz and N. Griswold, Establishing Collision Zones for Obstacles Moving with Uncertainty, Computer Vision, Graphics, and Image Processing, Vol. 49, pp. 95-103, 1990.

[Khatib, Oussama 1986]

Oussama Khatib, Real-Time Obstacle Avoidance for Manipulators and Mobile Robots, The International Journal of Robotics Research, Vol. 5, No. 1, 1986.

[Kim, Myung-Soo, 1991]

Myung-Soo Kim, San-Ryong Moon and Kwan-Hee Lee, Motion Planning with Planner Geometric Models, Proceedings of the 1991 IEEE International Conference on Robotics and Automation, Sacramento, California, April 1991.

[Kobayashi, Hisato 1990]

Hisato P. Kobayashi, Operations of Locomotive Robot by R.C.E. Network, Proceedings of the IEEE International Workshop on Intelligent Motion Control, Istanbul, 20-22 August 1990.

[Koren, Y. 1990]

Y. Koren, Analysis of Control Methods for Mobile Robot Obstacle Avoidance, Proceedings of the IEEE International Workshop on Intelligent Motion Control, 20-22 August 1990.

[Koren, Yoram 1991]

Yoram Koren and Johann Borenstein, Potential Field Methods and Their Inherent Limitations for Mobile Robot Navigation, Proceedings of the 1991 IEEE

International Conference on Robotics and Automation, Sacramento, California, April 1991.

[Koskinen K. 1992]

K. Koskinen, H. Makela, K. Rintanen and A. Penttinen, A Piloting System for Autonomous Robots in Outdoor Applications, Proceedings of ICARCV'92: Second International Conference on Automation, Robotics and Computer Vision, Singapore, 1992.

[Koyama, Hiroshi 1991]

Hiroshi Koyama, Yoichiro Maeda, Satoru Fukami and Tomohiro Takagi, Study of Obstacle Avoidance Problem for Mobile Robot Using Fuzzy Production System, Journal of the Robotics Society of Japan 9, 76-78, 1991.

[Kyriakopoulos, K. 1991]

K. J. Kyriakopoulos and G. N. Saridis, Collision Avoidance of Mobile Robots in Non-stationary Environments, Proceedings of the 1991 IEEE International Conference on Robotics and Automation, Sacramento, California, pp. 904-909, April 1991.

[Lafferriere, G. 1991]

G. Lafferriere and H. Sussmann, Motion Planning for Controllable Systems without Drift, Proceedings of the 1991 IEEE International Conference on Robotics and Automation, Sacramento, California, April 1991.

[Laloni, C. 1992]

C. Laloni, R. Gutsche and F. M. Wahl, A Factory-Floor Monitoring System for Mobile Robot Guidance, Proceedings of ICARCV'92: Second International conference on Automation, Robotics and Computer Vision, Singapore, 1992.

[Lamadrid, J. 1986]

J. G. Lamadrid, Avoidance System for Moving Obstacles, Proceedings of the SPIE Symposium on Mobile Robots, pp. 304-311, 1986.

[Larson, R. 1982a]

R. E. Larson and J. L. Casti, Principles of Dynamic Programming Part I: Basic Analytic and Computational Methods, Marcel Dekker Inc., 1982.

[Larson, R. 1982b]

R. E. Larson and J. L. Casti, Principles of Dynamic Programming Part II: Advanced Theory and Applications, Marcel Dekker Inc., 1982.

[Latombe, Jean-Claude 1990a]

Robot Motion Planning, Kluwer Academic Publishers, 1990.

[Laumond, Jean-Paul 1990b]

Jean-Paul Laumond, Michel Taix and Paul Jacobs, A Motion Planner for Car-like Robots Based on a Mixed Global/Local Approach, Proceedings of IEEE International Workshop on Intelligent Robots and Systems IROS'90, 1990.

[Lee, B. 1987]

B. Lee and C. S. G. Lee, Collision Free Motion Planning of Two Robots, IEEE Transactions on Systems, Man and Cybernetics, Vol. 17, pp. 21-32, 1987.

[Lee, D. 1984]

D. T. Lee and F. P. Preparata, Euclidean Shortest Paths in the Presence of Rectilinear Barriers, Networks, Vol. 1, pp. 393-410, 1984.

[Leonard, John 1992]

John J. Leonard, Hugh F. Durrant-Whyte and Ingemar J. Cox, Dynamic Map Building for an Autonomous Mobile Robot, The International Journal of Robotics Research, Vol. 11, No. 4, August 1992.

[Leou Jin-Jang 1992]

Jin-Jang Leou, Yung-Liu Chang and Jiann-Shing Wu, Robot Operation Monitoring for Collision Avoidance by Image Sequence Analysis, Pattern Recognition, Vol. 25, No. 8, pp. 855-867, 1992.

[Leven, Daniel 1987]

Daniel Leven and Micha Sharir, Intersection and Proximity Problems and Voronoi Diagrams, Algorithmic and Geometric Aspects of Robotics, Vol. 1, pp. 187-228, Lawrence Erlbaum Associates Publisher, 1987.

[Li, H. 1988]

Hua Li and Hyun S. Yang, Fast and Reliable Image Enhancement Using Fuzzy Relaxation Technique, Proceedings of the 4th International Conference on Pattern Recognition, UK, 1988.

[Lin, Ming 1991]

Ming C. Lin and John F. Canny, A Fast Algorithm for Incremental Distance Calculation, Proceedings of the 1991 IEEE International Conference on Robotics and Automation, Sacramento, California, April 1991.

[Liu, Yun-Hui 1990]

Yun-Hui Liu and Suguru Arimoto, A Flexible Algorithm for Planning Local Shortest Path of Mobile Robots Based on Reachability Graph, Proceedings of IEEE International Workshop on Intelligent Robots and Systems IROS'90, 1990.

[Liu, Yun-Hui 1991]

Yun-Hui Liu and Suguru Arimoto, Proposal of Tangent Graph and Extended Tangent Graph for Path Planning of Mobile Robots, Proceedings of the 1991

IEEE International Conference on Robotics and Automation, Sacramento, California, April 1991.

[Liu, Yun-Hui 1992]

Yun-Hui Liu and Suguru Arimoto, Path Planning Using a Tangent Graph for Mobile Robots Among Polygonal and Curved Obstacles, The International Journal of Robotics research, Vol. 11, No. 4, August 1992.

[Lozano-Perez, T. 1979]

T. Lozano-Perez and M. A. Wesley, An Algorithm for Planning Collision Free Paths among Polyhedral Obstacles, Communications of the ACM, Vol. 22, pp. 560-570, 1979.

[Lozano-Perez, Tomas 1991]

Tomas Lozano-Perez and Patrick A. O'Donnell, Parallel Robot Motion Planning, Proceedings of the 1991 IEEE International Conference on Robotics and Automation, Sacramento, California, April 1991.

[Luo, Ren 1990]

Ren C. Luo, T.-J Pan and C. M. Aras, Intelligent Motion Control System for Autonomous Vehicles Operating in a Factory Floor, Proceedings of the IEEE International Workshop on Intelligent Motion Control, Istanbul, 20-22, August 1990.

[Ma, Peijun 1992]

Peijun Ma, Kenji Ohshima and Satoshi Yamane, Application of Neural Network and Fuzzy Control to 2 Dimensional Orbit Tracking, Proceedings of ICARCV'92: Second International Conference on Automation, Robotics and Computer Vision, Singapore, 1992.

[Maeda, Yoichiro 1988]

Yoichiro Maeda and Morikazu Takegaki, Collision Avoidance Control among Moving Obstacles for a Mobile Robot on the Fuzzy Reasoning, Journal of the Robotics Society of Japan 6, 50-54, 1988.

[Mahesh, H. 1991]

H. Mahesh, J. Yuh and R. Lakshmi, A Co-ordinated Control of an Underwater Vehicle and Robotic Manipulator, Journal of Robotic Systems 8(3), pp. 339-370, 1991.

[Meagher, D. 1982]

D. Meagher, Geometrical Modelling Using Octree Encoding, Computer Graphics and Image Processing, Vol. 19, pp. 129-147, 1982.

[Meystel, A. 1986a]

A. Meystel, A. Guez and G. Hillel, Minimum Time Path Planning for Robot Motion in Obstacle Strewn Environment, Proceedings of 1986 Annual Computer Science Conference, Cincinnati, OH. 1986.

[Meystel, A. 1986b]

A. Meystel, A. Guez and H. Hillel, Minimum Time Path Planning for a Robot, Proceedings of the IEEE International Conference on Robotics and Automation, San Francisco, CA. 1986.

[Meystel, A. 1990]

A. Meystel, Autonomous Mobile Robots, World Scientific, 1990.

[Mitchell, J. 1990a]

J. S. B. Mitchell, Shortest Paths among Obstacles in the plane, Technical Reports, Cornell University, 1990.

[Mitchell, J. 1990b]

J. S. B. Mitchell and C. H. Papadimitriou, The Weighted Region Problem: Finding Shortest Paths through a Weighted Planar Subdivision, TR-885, School of Operations Research and Industrial Engineering, Cornell University, 1990.

[Mitchell, J. 1990c]

J. S. B. Mitchell, A New Algorithm for Shortest Paths Among Obstacles in the Plane, TR-832, School of Operations Research and Industrial Engineering, Cornell University, January, 1990.

[Mobasser, B. 1990]

B. Mobasser, Path Planning under Uncertainty from a Decision Analytic Perspective, Proceedings of the IEEE International Symposium on Intelligent Control, pp. 556-560, Albany, 1990.

[Murthy, C. 1990]

Chivukula A. Murthy and Sankar K. Pal, Fuzzy Thresholding: Mathematical Framework, Bound Functions and Weighted Moving Average Technique, Pattern Recognition Letters, Vol. 11, pp. 197-206, 1990.

[Nagata, Shigemi 1990]

Shigemi Nagata, Nobuo Watanabe and Kazuo Asakawa, Mobile Robot Control by Neural Networks, Systems and Computers in Japan, Vol. 21, No. 7, pp. 96-106, 1990.

[Nakamura, K. 1982]

K. Nakamura, S. Iwai and T. Sawaragi, Decision Support Using Causation Knowledge Base, IEEE Transactions on Systems, Man and Cybernetics, Vol. 12, 1982.

[Nelson, R. 1991]

R. C. Nelson, Qualitative Detection of Motion by a Moving Observer, *Journal of Computer Vision*, Vol. 7, No. 1, 1991, pp. 33-46.

[Nilson, N. 1969]

N. J. Nilson, A Mobile Automaton: An Application of Artificial Intelligence Techniques, *Proceedings of the First International Joint Conference on Artificial Intelligence*, pp. 509-520, Washington DC, 1969.

[Noborio, H. 1990a]

H. Noborio, T. Naniwa and S. Arimoto, A Quadtree Based Path Planning Algorithm for a Mobile Robot, *Journal of Robotics Systems*, Vol. 7, pp. 555-574, 1990.

[Noborio, Hiroshi 1990b]

Hiroshi Noborio, Several Path-Planning Algorithms of a Mobile Robot for an Uncertain Workspace and Their Evaluation, *Proceedings of the IEEE International Workshop on Intelligent Motion Control*, Istanbul, 20-22 August 1990.

[Noreils, Fabrice 1991a]

Fabrice R. Noreils and Roland Prajoux, From Planning to Execution Monitoring Control for Indoor Mobile Robots, *Proceedings of the 1991 IEEE International Conference on Robotics and Automation*, Sacramento, California, April 1991.

[Noreils, Fabrice 1991b]

Fabrice R. Noreils, Matthieu Herrb and Pierrick Grandjean, A Robot 3D Vision Tracking Module Integrated in a Mobile-Robot Control System, *Proceedings of the IEEE International Workshop on Intelligent Motion Control*, Istanbul, 20-22 August 1990.

[O'Duinn, John 1993]

John O'Duinn and Micheal O'hEigeartaigh, Navigation through Dynamic Environments Using Graph Theory, Working Paper: CA-1693, School of Computer Applications, Dublin City University, Dublin 9, Ireland, 1993.

[O'Dunlaing, C. 1983]

C. O'Dunlaing, M. Sharir and C. K. Yap, Retraction: A New Approach to Motion Planning, *proceedings of the 15th ACM Symposium on the Theory of Computing*, Boston, pp. 107-220, 1983.

[O'Dunlaing, C. 1984]

C. O'Dunlaing, M. Sharir and C. K. Yap, Generalised Voronoi Diagrams for Moving a Ladder: II. Efficient Construction of the Diagram, Technical Report No. 32, Robotics Laboratory, Courant Institute, New York University, 1984.

[O'Dunlaing, C. 1984]

C. O'Dunlaing, M. Sharir and C. K. Yap, Generalised Voronoi Diagrams for Moving a Ladder: I. Topological Analysis, Technical Report No. 32, Robotics Laboratory, Courant Institute, New York University, 1984.

[O'Dunlaing, Colm 1985]

Colm O'Dunlaing and Chee K. Yap, A "Retraction" Method for Planning the Motion of a Disc, Journal of Algorithms 6, pp. 104-111, 1985.

[O'Dunlaing, Colm 1987]

Colm O'Dunlaing, Motion Planning with Inertial Constraints, Algorithmica 2, pp. 431-475, 1987

[Occello, Michel 1992]

Michel Occello and Marie-Claude Thomas, Real Time Distributed Blackboards for Intelligent Control in Mobile Robotics, Proceedings of ICARCV'92: Second International Conference on Automation, Robotics and Computer Vision, Singapore, 1992.

[Ohmori, Yoshihiro 1992]

Yoshihiro Ohmori, Yasushi Nakauchi, Yoshikuni Itoh and Yuichiro Anzai, A Task Allocation Algorithm for Multiple Mobile Robot Environment, Proceedings of ICARCV'92: Second International Conference on Automation, Robotics and Computer Vision, Singapore, 1992.

[Pal, S. 1981]

Sankar K. Pal and Robert A. King, Image Enhancement Using Smoothing with Fuzzy Sets, IEEE Transactions on Systems, Man and Cybernetics, Vol. SMC-11, No. 7, July, 1981.

[Pal, S. 1988]

Sankar, K. Pal and Azriel Rosenfeld, Image Enhancement and Thresholding by Optimisation of Fuzzy Compactness, Pattern Recognition Letters, Vol. 7, pp. 77-86, 1988.

[Pal, S. 1989]

Sankar K. Pal, Fuzzy Skeletonization of an Image, Pattern Recognition Letters, Vol. 10, pp. 17-23, 1989.

[Pal, S. 1990]

Sankar K. Pal and Ashish Ghosh, Index of Area Coverage of Fuzzy Image Subsets and Object Extraction, Pattern Recognition Letters, Vol. 11, pp. 831-841, 1990.

[Pan, T. 1990]

T. Pan and R. C. Luo, Motion Planning for Mobile Robots in a Dynamic Environment with Moving Obstacles, Proceedings of the IEEE International Conference on Robotics and Automation, Cincinnati, pp. 578-583, 1990.

[Pin, Francois 1990]

Francois G. Pin and Hubert A. Vasseur, Autonomous Trajectory Generation for Mobile Robots with Non-Holonomic and Steering Angle Constraints, Proceedings of the IEEE International Workshop on Intelligent Motion Control, Istanbul, 20-22, August 1990.

[Porto, Vincent 1991]

Vincent W. Porto and David B. Fogel, Neural Network Techniques for Navigation of AUVs, Proceedings of the Symposium on Autonomous Underwater Vehicle Technology AUV 90, Washiton, 5-6 June 1990.

[Pourboghra, F. 1991]

F. Pourboghra, Neural Networks Application in Autonomous Path Generation for Mobile Robots, SPIE - The International Society for Optical Engineering, Vol. 1396 Applications of Optical Engineering: Proceedings of SPIE, Vol. 1396, pp. 243-251, 1991.

[Pruski, A. 1991]

A. Pruski and S. Rohmer, Multivalued Coding: Application to Autonomous Robot Path Planning with Rotations, Proceedings of the 1991 IEEE International Conference on Robotics and Automation, Sacramento, California, April 1991.

[Reif, J. 1985]

J. H. Reif and J. A. Storer, Shortest Paths in Euclidean Space with Polyhedral Obstacles, Technical Report CS-85-121, Computer Science Department, Brandeis University, 1985.

[Reif, John 1985]

John Reif and Micha Sharir, Motion Planning in the Presence of Moving Obstacles, Proceedings of the 26th Annual IEEE Symposium on the Foundations of Computer Science, Portland (Oreg.), pp. 144-154, 1985.

[Rimon, Elon 1991]

Elon Rimon, A Navigation Function for a Simple Rigid Body, Proceedings of the 1991 IEEE International Conference on Robotics and Automation, Sacramento, California, April 1991.

[Roach, J. 1980]

J. W. Roach and J. K. Aggarwal, Determining the Movement of Objects from a Sequence of Images", IEEE Transactions on PAMI, Nov. 1980, pp. 554-562.

[Rosa, Garcia 1989]

R Garcia Rosa, D. Kumpel and M C Garcia-Alegre, A Local Navigation System Based on Fuzzy Control, Proceedings of 5th CIM Europe Conference, May 17-19, 1989.

[Rosa, Ricardo 1990]

Ricardo Garcia Rosa and Maria C. Garcia-Alegre, Fuzzy Logic Strategies to Control an Autonomous Mobile Robot, Cybernetics and Systems: An International Journal, 21, pp. 267-276, 1990.

[Rowe, N. 1990]

N. C. Rowe and R. F. Richbourg, An Efficient Snell's Law Method for Optimal Path Planning Across Multiple Two Dimensional, Irregular, Homogeneous Cost Regions, International Journal of Robotics Research, Vol. 9, pp. 48-66, 1990.

[Samet, H. 1985a]

H. Samet and R. E. Webber, Storing a Collection of Polygons Using Quadrees, ACM Transactions on Graphics, Vol. 2, pp. 182-222, 1985.

[Samet, H. 1985b]

H. Samet and M. Tamminen, Bintree, CSG trees, and Time, Proceedings of the SIGGRAPH'85 Conference, pp. 121-130, 1985.

[Samson, C. 1991]

C. Samson and K. Ait-Abderrahim, Feedback Control of a Nonholomic Wheeled Cart in Cartesian Space, Proceedings of the 1991 IEEE International Conference on Robotics and Automation, Sacramento, California, April 1991.

[Samson, C. 1992]

C. Samson, Path Following and Time-Varying Feedback Stabilisation of a Wheeled Mobile Robot, Proceedings of ICARCV'92: Second International Conference on Automation, Robotics and Computer Vision, Singapore, 1992.

[Sankaranarayanan, A. 1991]

A. Sankaranarayanan and M. Vidyasagar, Path Planning for Moving a Point Object amidst Unknown Obstacles in a Plane: the Universal Lower Bound on Worst Case Path Lengths and a Classification of Algorithms, Proceedings of the 1991 IEEE International Conference on Robotics and Automation, Sacramento, California, April 1991.

[Schwartz, J. 1983a]

J. T. Schwartz and M. Sharir, On the Piano Movers' Problem: I. The Case if a Two Dimensional Rigid Body Moving Amidst Polygonal Barriers, Communications on Pure and Applied Mathematics, 36, pp. 345-398, 1983.

[Schwartz, J. 1983b]

J. T. Schwartz and M. Sharir, On the Piano Movers' Problem: II. General Techniques for Computing Topological Properties of Real Algebraic Manifolds, *Advances in Applied Mathematics*, 4, pp. 298-351, 1983.

[Sharir, M. 1986]

M. Sharir and A. Schorr, On shortest paths in polyhedral spaces, *SIAM Journal on Computing*, Vol. 15, pp. 193-215, 1986.

[Sharir, M. 1987]

M. Sharir, On shortest paths amidst convex polyhedra, *SIAM Journal on Computing*, Vol. 16, pp. 561-572, 1987.

[Sharma, Rajeev 1991]

Rajeev Sharma, Safe Motion Planning for a Robot in a Dynamic, Uncertain Environment, *Proceedings of the 1991 IEEE International Conference on Robotics and Automation*, Sacramento, California, April 1991.

[Shih, C. 1990]

C. Shih, T. Lee and W. A. Gruver, A Unified Approach for Robot Motion Planning with Moving Polyhedral Obstacles, *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 20, pp. 903-915, 1990.

[Shih, Ching-Long 1991]

Ching-Long Shih, J. Peter Sadler and William A. Gruver, Collision Avoidance for Two SCARA Robots, *Proceedings of the 1991 IEEE International Conference on Robotics and Automation*, Sacramento, California, April 1991.

[Shihe, Xiong 1992]

Xiong Shihe and Hu Tianyou, Intelligent Robot "TRESTU" Control System, *Proceedings of ICARCV'92: Second International Conference on Automation, Robotics and Computer Vision*, Singapore, 1992.

[Shin, Dong 1991]

Dong Hun Shin, Sanjiv Singh and Wenfan Shi, A Partitioned Control Scheme for Mobile Robot Path Tracking, *Proceedings of IEEE International Conference on Systems Engineering*, USA, August 1991.

[Shin, Kang 1992]

Kang G. Shin and Qin Zheng, Minimum-Time Collision-Free Trajectory Planning for Dual-Robot Systems, *IEEE Transactions on Robotics and Automation*, Vol. 8, No. 5, October 1992.

[Shirai, Y. 1990]

Y. Shirai, Flexible Path Selection and Obstacle Detection for AGV, *Proceedings of the IEEE International Workshop on Intelligent Motion Control*, Istanbul, 20-22 August 1990.

[Sorouchyari, Esfandiar 1989]

Esfandiar Sorouchyari, Mobile Robot Navigation: A Neural Network Approach, Proceedings of the Journess D'Electroige 1989 Artificial Neural Networks, pp. 159-160, Lausanne, Switzerland, 10-12 October 1989.

[Steer, Barry 1991]

Barry Steer and Madeline Larcombe, A Goal Seeking and Obstacles Avoiding Algorithm for Autonomous Mobile Robots, Proceedings of the 1991 IEEE International Conference on Robotics and Automation, Sacramento, California, April 1991.

[Stella, E. 1992]

E. Stella, M. Ianigro, T. D'Orazio, F. P. Lovergine and A. Distanto, Mobile Robot Navigation on a Planned Path, Proceedings of ICARCP'92: Second International Conference on Automation, Robotics and Computer Vision, Singapore, 1992.

[Suh, S. 1988]

S. H. Suh and K. G. Shin, A Variational Dynamic Programming Approach to Robot Path Planning with a Distance Safety Criterion, IEEE Transactions on Robotics and Automation, Vol. 4, pp. 334-349, 1988.

[Sutner, K. 1988]

K. Sutner and W. Maass, Motion Planning among Time Dependent Obstacles, Acta Informatica, Vol. 26, pp. 93-122, 1988.

[Suzuki, H. 1990]

H. Suzuki and A. Arimoto, A Recursive Method of Trajectory Planning for a Point-like Mobile Robot in Transient Environment Utilising Paint Procedure, Journal of the Robotics Society of Japan, Vol. 8, pp. 9-19, 1990.

[Tai-Jee, Pan 1991]

Tai-Jee Pan and Ren C. Luo, A Feasible Collision-Detection Algorithm for Mobile Robot Motion Planning with Moving Obstacles, IECON'91.

[Takahashi, Osamu 1989]

Osamu Takahashi and R. J. Schilling, Motion Planning in a Plane Using Generalised Voronoi Diagrams, IEEE Transactions on Robotics and Automation, Vol. 5, No. 2, April 1989.

[Takeuchi, T. 1988]

T. Takeuchi and Y. Nagai, Fuzzy Control of a Mobile Robot for Obstacle Avoidance, Information Science, Vol. 45, pp. 231-248, 1988.

[Tang, Kai 1992]

Kai Wing Tang and Ray Jarvis, Collision-Free Path Finding Amongst Polygonal Obstacles Using Efficient Free Space Triangulation, Proceedings of ICARCV'92:

Second International Conference on Automation, Robotics and Computer Vision, Singapore, 1992.

[Tarassenko, L. 1991]

L. Tarassenko and A. Blake, Analogue Computation of Collision-Free Paths, Proceedings of the 1991 IEEE International Conference on Robotics and Automation, Sacramento, California, April 1991.

[Tarassenko, Lionel 1991]

Lionel Tarassenko, Gillian Marshall and Felipe Gomez-Castaneda, Parallel Analogue Computation for Real-Time Path Planning, VLSI for Artificial Intelligence and Neural Networks, Plenum Press, 1991.

[Thompson, W. 1990]

W. B. Thompson and T. C. Pong, Detecting Moving Objects, Journal of Computer Vision, Vol. 4, 1990, pp. 39-57.

[Tominaga, Henry 1991]

Henry Tominaga and Behnam Bavarian, Solving the Moving Obstacle Path Planning Problem Using Embedded Variational Methods, Proceedings of the 1991 IEEE International Conference on Robotics and Automation, Sacramento, California, April 1991.

[Tournassoud, P. 1986]

P. Tournassoud, A Strategy for Obstacle Avoidance and its Application to Multi-robot systems, Proceedings of the IEEE International Conference on Robotics and Automation, San Francisco, pp. 1224-1229, 1986.

[Tsai, R. 1981]

R. Y. Tsai and T. S. Huang, Estimating Three-Dimensional Motion Parameters of a Rigid Planar Patch, IEEE Transactions on ASSP, Dec. 1981, pp. 1147-1152.

[Tsukagoshi, Tsuneo 1990]

Tsuneo Tsukagoshi and Hiroo Wakaumi, Highly-Reliable Semi-Autonomous Vehicle Control on Lattice Lane, Proceedings of IEEE International Workshop on Intelligent Robots and Systems IROS'90, 1990.

[Udupa, S. 1977]

S. Udupa, Collision Detection and Avoidance in Computer Controlled Manipulators, Proceedings of the 5th International Joint Conference on Artificial Intelligence, pp. 737-748, 1977.

[Vasseur, Hubert 1991]

Hubert A. Vasseur, Francois G. Pin and Jack R. Taylor, Navigation of a Car-Like Mobile Robot Using a Decomposition of the Environment in Convex Cells, Proceedings of the 1991 IEEE International Conference on Robotics and Automation, Sacramento, California, April 1991.

[Vercelli, G. 1991]

G. Vercelli, R. Zaccaria and P. Morasso, A Theory of Sensor-Based Robot Navigation Using Local Information, Lecture Notes in Artificial Intelligence 549, Lecture Notes in Computer Science, 1991.

[Wang, B. 1993b]

B. Wang, C. Daly, M. Scott and M. Ryan, An Approach to Motion Planning of a Mobile Robot, 6th Irish Conference on Artificial Intelligence and Cognitive Science, Queen's University, Belfast, Sept. 1993.

[Wang, B. 1993a]

B. Wang, C. Daly, M. Ryan and M. Scott, Fuzzy Logic Methods for Image Motion Detection and Analysis, IEEE International Conference on Systems, Man and Cybernetics, Le Touquet, France, Oct., 1993.

[Wang, B. 1994]

B. Wang, C. Daly and M. Ryan, Minimum Time Navigation of a Mobile Robot, EURISCON'94: European Robotics and Intelligent Systems Conference, Malaga, Spain, August, 1994.

[Wang, Ling-Ling 1991]

Ling-Ling Wang and Wen-Hsiang Tsai, Scheme for Indoor Autonomous Land Vehicle Navigation, Journal of Robotic Systems 8(5), pp. 677-698, 1991.

[Wang, P. 1989]

P. K. C. Wang, Interaction Dynamics of Multiple Mobile Robots with Simple Navigation Strategies, Journal of Robotic Systems, Vol. 6, No. 1, pp. 77-101, 1989.

[Wang, P. 1991]

P. K. C. Wang, Navigation Strategies for Multiple Autonomous Mobile Robots Moving in Formation, Journal of Robotic Systems, Vol. 8, No. 2, pp. 177-195, 1991.

[Warren C. 1990]

C. W. Warren, Multiple Robot Path Co-ordination Using Artificial Potential Fields, Proceedings of the IEEE International Conference on Robotics and Automation, Cincinnati, pp. 500-505, 1990.

[Warren Charles 1991]

Charles W. Warren, A Vector Based Approach to Robot Path Planning, Proceedings of the 1991 IEEE International Conference on Robotics and Automation, Sacramento, California, April 1991.

[Weiczer, Ference 1992]

Ference Weiczner, Path Planning and Following for Mobile Robots with Dynamic Constraints Considered, Proceedings of ICARCV'92: Second International Conference on Automation, Robotics and Computer Vision, Singapore, 1992.

[Wilfong, Gordon 1991]

Gordon Wilfong, Motion Planning in the Presence of Movable Obstacles, *Annals of Mathematics and Artificial Intelligence*, Vol. 3, No. 1, pp. 131-150, January, 1991.

[Wong, E. 1986]

E. K. Wong and K. S. Fu, A Hierarchical Orthogonal Space Approach to Three-Dimensional Path Planning, *IEEE Journal of Robotics and Automation*, Vol. RA-2, No. 1, March 1986.

[Yagi, Y. 1991]

Y. Yagi, S. Kawato and S. Tsuji, Collision Avoidance Using Omnidirectional Image Sensor (COPIS), Proceedings of the IEEE International Conference on Robotics and Automation, Sacramento, pp. 910-915, 1991.

[Yagi, Yasushi 1992]

Yasushi Yagi, Yoshimitsu Nishizawa and Masahiko Yachida, Obstacle Avoidance for Mobile Robot Integrating Omnidirectional Image Sensor COPIS and Ultrasonic Sensor, Proceedings of ICARCV'92: Second International Conference on Automation, Robotics and Computer Vision, Singapore, 1992.

[Yan, J. 1992a]

J. Yan, R. Lawlor and M. Ryan, Image Motion Detection and Analysis Using Fuzzy Logic, Proceedings of the second IEEE International Conference on Automation, Robotics and Computer Vision (ICARCV'92), Singapore, Sept. 1992.

[Yan, J. 1992b]

J. Yan and M. Ryan, Fuzzy Logic Based Digital Image Processing, Proceedings of the International Conference on Factory Automation and Information Management (FIAM'92), USA, pp. 821-832, June, 1992.

[Yap, C. 1984]

C. Yap, Co-ordinating the Motion of Several Discs, TR-105, Courant Institute of Mathematical Sciences, New York University, 1984.

[Yeung, Dit-Yan 1987]

Dit-Yan Yeung and George A. Bekey, A Decentralised Approach to the Motion Planning Problem for Multiple Mobile Robots, Proceedings of the IEEE International Conference on Robotics and Automation, Raleigh (Ga.), pp. 1779-1784, 1987.

[Yoerger, Dana 1991]

Dana R. Yoerger, Hagen Schempf and David M. DiPietro, Design and Performance Evaluation of an Active Compliant Underwater Manipulator for Full-Ocean Depth, *Journal of Robotic Systems* 8(3), pp. 371-392, 1991.

[Zhang, Zhengyou 1992]

Zhengyou Zhang and Olivier Faugeras, A 3D World Model Builder with a Mobile Robot, *The International Journal of Robotics Research*, Vol. 11, No. 4, August 1992.

[Zhu, D. 1991]

D. Zhu and J. Latombe, New Heuristic Algorithms for Efficient Hierarchical Path Planning, *IEEE Transactions on Robotics and Automation*, Vol. 7, pp. 9-20, 1991.

Appendix A. Some Header Files

A.1 Header File of class Rect and Rso

```
//rectangle classes: rsounit.h
#ifndef H_RESOUNIT
#define H_RESOUNIT

#include "scrnsty.h"

enum {False=0, True=1};

//-----Rectangle Class-----

class Rect
{
public:
    int Xul, Yul;           // upper left-hand corner
    int Xlr, Ylr;           // lower right-hand corner
    int Wd, Ht;             // overall size
    Rect(int X, int Y, int W, int H);
    virtual ~Rect(void) { ; }
    virtual void SetSize(int W, int H);
    virtual void SetLocn(int Xl, int Yl);
    virtual void ClipSize(int &W, int &H);
    virtual void ClipDim(int &X, int &Y, int &W, int &H);
    virtual int HzClip(int &X, int &Y, int &N);
    virtual int VtClip(int &X, int &Y, int &N);
    virtual int Contains(int X, int Y);
    virtual int Touches(Rect *R);
};

//-----Rectangular Screen Object Class-----

class Rso: public Rect
{
public:
    Rso(int X, int Y, int W, int H) : Rect(X, Y, W, H) { ; }
    virtual void HzWrt(int, int, char *, char)=0;
    virtual void HzWrtB(int, int, char *)=0;
    virtual void Fill(int, int, int, int, char, char)=0;
    virtual void FillB(int, int, int, int, char, char)=0;
    virtual void Box(int, int, int, int, char, char)=0;
    virtual void Scroll(ScrollDir, int) { ; }
};

#endif
```

A.2 Header File for Screen Style Definitions

```
//      scrnsty.h
//      Screen style definitions
#ifndef H_SCRNSTY
#define H_SCRNSTY

inline unsigned char ForeGround(unsigned char attr)
{
    return attr & 0x0f;
}
inline unsigned char BackGround(unsigned char attr)
{
    return (attr >> 4) & 0x0f;
}

enum LineChar
{
    Ulc, Llc, Urc, Lrc, LTee, RTee, UpTee, DnTee, HzBar, VtBar
};
extern char LineChars[4][10];
enum ScrollDir
{
    UpScroll, DnScroll, LeftScroll, RightScroll
};
struct ColorPak
{
    unsigned char Bc, Wc, Tc, Pc, Fc;
};
//      Border style

const int NoBorder=0x00;
const int Single  =0x10;
const int Double  =0x20;
const int Dashed  =0x30;
const int Solid   =0x10;
const int Recessed=0x20;
const int Relief  =0x30;

//      Frame style

const int Swappable  =0x01;
const int Closeable  =0x02;
const int SShadow    =0x04;
const int NShadow    =0x08;
const int NWShadow   =0x10;
const int SWShadow   =0x20;
const int OutlineMove=0x40;
const int BorderPrompt=0x80;
const int Stretchable=0x100;

//      A shadow bit mask

const int AnyShadow = SShadow+NShadow+NWShadow+SWShadow;

//      A set of predefined color packs and styles

extern ColorPak BlueColors;
extern ColorPak CyanColors;
extern ColorPak RedColors;
extern ColorPak InvColors;
extern ColorPak MonoColors;
extern ColorPak ErrColors;
```

```

extern ColorPak MsgColors;
extern ColorPak GrphColors;
extern ColorPak DefColors;
extern int WithShadow;
extern int WindowStyle;
extern int ButtonStyle;
#endif

```

A.3 Header File of Class TxBuff

```

//texunit.h
//Provide low-level screen support through the TxBuff object.

#ifndef H_TXUNIT
#define H_TXUNIT

#include "scrnsty.h"
#include <string.h>

struct Texel { unsigned char Ch, Attr; };
typedef struct Texel far *TexelPtr;
class TxBuff
{
public:
    int Wd, Ht;
    TexelPtr TxRam;
    TexelPtr OrgPtr;
    char Aliased;
    TxBuff(TxelPtr T);
    TxBuff(int W, int H, TexelPtr T);
    ~TxBuff(void);
    void SetRamPtr(TxelPtr T);
    TexelPtr TxRamAddr(int X, int Y);
    TexelPtr RelAddr(int X, int Y);
    void SetSize(int W, int H);
    void SetLocn(int X, int Y);
    void Swap(int X, int Y, int W, int H, TxBuff *Other, int Xs,
int Ys);
    void Xfr(int X, int Y, int W, int H, TxBuff *Other, int Xs,
int Ys);
    void Scroll(int X, int Y, int W, int H, ScrollDir Sd, int
Amt);
    void Fill(int X, int Y, int W, int H, char Ch, char Attr);
    void FillB(int X, int Y, int W, int H, char Ch, char Opt);
    void HZWrt(int X, int Y, char *Str, char Attr, unsigned Cnt);
    void HZWrt(int X, int Y, char *Str, char Attr)
    {
        HZWrt(X, Y, Str, Attr, strlen(Str));
    }
    void HZWrtB(int X, int Y, char *Str, unsigned Cnt);
    void HZWrtB(int X, int Y, char *Str)
    {
        HZWrtB(X, Y, Str, strlen(Str));
    }
    void HZFill(int X, int Y, char Ch, char Attr, unsigned Cnt)
    {
        Fill(X, Y, Cnt, 1, Ch, Attr);
    }
    void HZFillB(int X, int Y, char Ch, char Opt, unsigned Cnt)
    {
        FillB(X, Y, Cnt, 1, Ch, Opt);
    }
};

```

```

TexelPtr VideoPtr(unsigned &Vmode, unsigned &Vpage );
TexelPtr VideoPtr(void);

#endif

```

A.4 Header File of Class Trso

```

//trsounit.h: Text-based Rectangular Screen Object (Trso) class
#ifndef H_TRSOUNIT
#define H_TRSOUNIT
#include "msmouse.h"
#include "txunit.h"
#include "rsounit.h"
class Trso : public Rso
{
public:
    TxBuff *Pic;
    Trso(TxBuff *T);
    virtual ~Trso(void);
    virtual void SetSize(int W, int H);
    virtual void SetLocn(int Xl, int Yl);
    virtual void HzWrt(int X, int Y, char *Str, char Attr);
    virtual void HzWrtB(int X, int Y, char *Str);
    virtual void Fill(int X, int Y, int W, int H, char Ch, char
Attr);
    virtual void FillB(int X, int Y, int W, int H, char Ch, char
Opt);
    virtual void Box(int X, int Y, int W, int H, char Ba, char
Attr);
    virtual void Scroll(ScrollDir Sd, int Amt);
    virtual void Swap(int X, int Y, int W, int H, Trso *Other, int
Xs, int Ys);
    virtual void Xfr(int X, int Y, int W, int H, Trso *Sre, int
Xs, int Ys);
};
#endif

```

A.5 Header File of Class Grso

```

//grsounit.h: graphics-based rectangular screen object (Grso)
class
#ifndef H_GRSOUNIT
#define H_GRSOUNIT
#include <graphics.h>
#include "msmouse.h"
#include "scrnsty.h"
#include "rsounit.h"
class Grso : public Rso
{
public:
    int TxtHt;
    Grso(void);
    virtual void HzWrt(int X, int Y, char *Str, char Attr);
    virtual void HzWrtB(int X, int Y, char *Str);
    virtual void Fill(int X, int Y, int W, int H, char Ch,
char Attr);
    virtual void FillB(int X, int Y, int W, int H, char Ch,
char Attr);
    virtual void Box(int X, int Y, int W, int H, char Ba,

```

```

char Attr);
        virtual int TextWidth(char *Str);
        virtual int TextHeight(int N);
    };
#endif

```

A.6 Header File of Class Fso

```

//fsounit.h: Framed Screen Object (FSO) General Class
#ifndef H_FSOUNIT
#define H_FSOUNIT

#include <stdlib.h>
#include "rsounit.h"

enum CoordType {Relc, Absc};
enum XfrDirn {GetIm, PutIm};

class Fso
{
    public:
        Rso *Overall, *Frame, *Interior;
        int Bwd;
        int Bstyle;
        int Fattr;
        ColorPak Colors;
        Fso(int Ba, int Fa, ColorPak &Cp);
        virtual ~Fso(void) { ; }
        virtual int IsSwappable(void);
        virtual int IsCloseable(void);
        virtual int IsStretchable(void);
        virtual int HasShadow(void);
        virtual void SetSize(int W, int H);
        virtual void SetLocn(int Xl, int Yl);
        virtual int OnFrame(int X, int Y);
        virtual int OnInterior(int X, int Y);
        virtual int OnBorder(int X, int Y);
        virtual int OnCloseButton(int X, int Y);
        virtual int Touches(Fso *F);
        virtual void Clear(char, char) { ; }
        virtual void DrawFrame(char, char) { ; }
        virtual void GetImage(Rect *) { ; }
        virtual void PutImage(Rect *) { ; }
        virtual void ShadowXfr(Rect *, XfrDirn, int) { ; }
        virtual void DrawShadows(Rect *, XfrDirn, int) { ; }
        virtual void Swap(Rect *, XfrDirn) { ; }
        virtual void Scroll(ScrollDir Sd, int Amt);
        virtual void HZWrt(int X, int Y, char *Str, char Attr);
        virtual void HZWrtB(int X, int Y, char *Str);
        virtual void Fill(int X, int Y, int W, int H, char Ch,
char Attr);
        virtual void FillB(int X, int Y, int W, int H, char Ch,
char Opt);
        virtual void Box(int X, int Y, int W, int H, char Ba,
char Attr);
        virtual int TextWidth(char *) { return 0; }
        virtual int TextHeight(int) { return 0; }
    };
#endif

```

A.7 Header File of Class Tfso

```
//tfsounit.h: Text-Based Framed Screen Object (Tfso) Class

#ifndef H_TFSOUNIT
#define H_TFSOUNIT

#include <string.h>
#include "trsounit.h"
#include "fsounit.h"

class Tfso: public Fso
{
    public:
        Trso *SaveBuff;
        Tfso *HzShadow, *VtShadow;
        Tfso (int Ba, int Fa, ColorPak &Cp);
        virtual ~Tfso(void);
        virtual void SetSize(int W, int H);
        virtual void SetLocn(int Xl, int Yl);
        virtual int OnCloseButton(int X, int Y);
        virtual void DrawFrame(char Ba, char Attr);
        virtual void Clear(char Ch, char Attr);
        virtual void GetImage(Rect *C);
        virtual void PutImage(Rect *C);
        virtual void ShadowXfr(Rect *C, XfrDirn Xd, int DrawIt);
        virtual void DrawShadows(Rect *C, XfrDirn Xd, int
DrawIt);
        virtual void Swap(Rect *C, XfrDirn Xd);
        virtual int TextWidth(char *Str) { return strlen(Str); }
        virtual int TextHeight(int N) {return N; }

};

//----Text based "skeleton" frame screen objects----

class Tskel : public Fso
{
    public:
        Trso *Sides[4];           //Top, bottom, left, right
        Tskel(ColorPak &Cp);
        virtual ~Tskel(void);
        virtual void SetSize(int W, int H);
        virtual void DrawFrame(char Ba, char Attr);
        virtual void GetImage(Rect *C);
        virtual void PutImage(Rect *C);
        virtual void Swap(Rect *C, XfrDirn Xd);

};

extern TxBuff ScrnBuff;           //The text buffer representing the
screen

#endif
```

A.8 Header File of Class Gfso

```
//gfsounit.h: Provides graphics mode "frames" that are used in
//the Iso screen objects. A frame consists of an interior, panel
//and overall object.

#ifndef H_GFSOUNIT
```

```

#define H_GFSOUNIT

#include "fsounit.h"
#include "grsounit.h"

class Gfso: public Fso
{
    public:
        char *SaveBuff, *SwapBuff;
        unsigned SaveSize;
        Grso *CloseBox;
        int HeaderHt;

        Gfso(int Ba, int Fa, ColorPak &Cp);
        virtual ~Gfso(void);
        virtual void SetSize(int W, int H);
        virtual void SetLocn(int Xl, int Yl);
        virtual void DrawFrame(char Ba, char Attr);
        virtual void Clear(char Ch, char Attr);
        virtual void GetImage(Rect *C);
        virtual void PutImage(Rect *C);
        virtual void Swap(Rect *C, XfrDirn Xd);
        virtual int OnCloseButton(int X, int Y);
        virtual int TextWidth(char *Str);
        virtual int TextHeight(int N);
};

class Gskel: public Fso
{
    public:
        Gskel(ColorPak &Cp);
        virtual void DrawFrame(char Ba, char Attr);
        virtual void Swap(Rect *, XfrDirn Xd);
};

#endif

```

A.9 Header File of Class Iso and Class IsoMgr

```

// isounit.h: interactive screen object (ISO) class

#ifndef H_ISOUNIT
#define H_ISOUNIT

#include <stdlib.h>
#include "fsounit.h"
#include "keybrd.h"

//Some event codes (The msmouse.h and keybrd.h files provide others)
const int Close = 0x11b; //same as the Esc key
const int ShutDown = 0x2d00; //same as the Alt-X key
const int Cycle = 0x0f09; //same as the Tab key
const int StrMsg = 0xffff; //indicates there is a string message

class Iso;
class IsoMgr;

struct MsgPkt //message packet
{
    Iso *Focus; //holds current focus
    unsigned Code; //the event code
    unsigned RtnCode; //return event code
};

```

```

        int Mx, My;                //mouse co-ordinates
        char Str[80];              //the message
    };

extern MsgPkt NullMsg;            //defined in isounit.cpp

class Iso
{
public:
    Fso *Panel;                   //can be a Tfso or Gfso pointer
    char Active;                  //true if iso is selected
    char Visible;                 //true if iso is showing
    char IsClosed;                //true if iso is closed
    char TouchFlag;               //true if touchint another iso
    char ClipToFrame;             //used for scroll bar
    Iso *Base;                    //the base for this iso
    Iso *Under, *Over;            //Iso stack pointer
    IsoMgr *SubMgr;               //the child iso stack
    Iso(Fso *P);
    virtual ~Iso(void);
    virtual Rso *ClippingRect(void);
    virtual void SetLocn(int Xl, int Yl, CoordType Ctype);
    virtual void SetSize(int W, int H);
    virtual void Move(int X, int Y);
    virtual void DeltaMove(int Dx, int Dy);
    virtual void MoveLoop(MsgPkt &M);
    virtual void Stretch(int W, int H);
    virtual void DeltaStretch(int Dw, int Dh);
    virtual void StretchLoop(MsgPkt &M);
    virtual int Obscured(void);
    virtual void Draw(void) { ; }
    virtual void DrawPanel(void);
    virtual void Redraw(void) { DrawPanel(); }
    virtual void Open(Iso *B, int X, int Y);
    virtual void Reopen(int X, int Y);
    virtual void Swap(void);
    virtual void Hide(void);
    virtual void Show(void);
    virtual void SetVisibleFlag(int F);
    virtual void Select(void);
    virtual void Remove(void);
    virtual void Prompt(void);
    virtual void UnPrompt(void);
    virtual void Enter(MsgPkt &M);
    virtual void Leave(MsgPkt &M);
    virtual void SwitchFocus(MsgPkt &M);
    virtual void Activate(MsgPkt &M) { ; }
    virtual void OnMouseEnter(MsgPkt &M);
    virtual void OnMouseLeave(MsgPkt &M);
    virtual void OnMouseWithin(MsgPkt &M) { ; }
    virtual void OnClose(MsgPkt &M);
    virtual void OnMouseUp(MsgPkt &M) { Activate(M); }
    virtual void OnMouseDown(MsgPkt &M);
    virtual void BorderHandler(MsgPkt &M);
    virtual void OnMouseStillDown(MsgPkt &M) { ; }
    virtual void OnKeyStroke(MsgPkt &M);
    virtual void OnShiftArrow(MsgPkt &M);
    virtual void Dispatch(MsgPkt &M);
};

class IsoMgr
{
public:
    Iso *Top, *Bottom;            //Top and Bottom of Iso stack
    Iso *Base;                    //whom this stack is belongs to
};

```

```

Iso *Hot;                                //whom the mouse is currently on
Iso *Marker;                             //a marker used when cycling
IsoMgr(Iso *B);
virtual ~IsoMgr(void);
virtual void Push(Iso *Ip);
virtual void MoveToFront(Iso *Me, int Keep);
virtual void ResetTouchFlags(Iso *Me);
virtual void SetTouchFlags(Iso *Me);
virtual void OnIso(int Mx, int My, Iso **I);
virtual Iso *CycleForw(Iso *Curr);
virtual void ProcessCycle(MsgPkt &M);
virtual void EventLoop(MsgPkt &M);
virtual void EventStep(MsgPkt &M);
};

#endif

```

A.10 Header File of Text Window Object Class Wso

```

// twsounit.h: text window object (Twso) class

#ifndef H_TWSOUNIT
#define H_TWSOUNIT

#include "isounit.h"
#include "tfsounit.h"

class Wso: public Iso
{
public:
    Wso(int Ba, int Fa, ColorPak &Cp);
    virtual void MoveLoop(MsgPkt &M);
    virtual void StretchLoop(MsgPkt &M);
    virtual void Prompt(void);
    virtual void UnPrompt(void);
};

#endif

```

A.11 Header File of Graphic Window Object Class Wso

```

// gwsounit.h: graphics-based window screen object class

#ifndef H_GWSOUNIT
#define H_GWSOUNIT

#include "isounit.h"
#include "gfsounit.h"

class Wso: public Iso
{
public:
    Wso(int Ba, int Fa, ColorPak &Cp);
    virtual void MoveLoop(MsgPkt &M);
    virtual void Prompt(void);
    virtual void UnPrompt(void);
};

#endif

```

A.12 Header File of the Menu System

```
//      msounit.h: menu screen object (mso) class definition

#ifndef H_MSOUNIT
#define H_MSOUNIT

//      Conditional directives used to determine which files
//      should be included

#ifdef GRAPHICS
#include "gwsounit.h"
#else
#include "twsounit.h"
#endif

typedef void (*ActionProc) (Wso *Src, MsgPkt &M);

//-----Generic menu entry screen object type-----

class Meso : public Wso
{
public:
    char Name[40];           //menu entry label
    ActionProc Action;       //The action assigned to the entry
    Meso *Next;              //reference the next menu entry
    Meso(char *N, ActionProc A);
    virtual void Draw(void);
    virtual void Prompt(void);
    virtual void UnPrompt(void);
    virtual void OnKeyStroke(MsgPkt &M);
    virtual void OnClose(MsgPkt &M);
    virtual void Activate(MsgPkt &M);
};

class MesoList              //a circular list for storing menu entries
{
public:
    Meso *Last;              //references last menu entry on list
    MesoList(void);
    virtual ~MesoList(void);
    virtual void Append(Meso *Me);
};

class Mso : public Wso      //the menu screen object type
{
public:
    MesoList *Entries;       //the list of menu entries
    Meso *CurrSeln;          //the selected menu entry
    int EntriesDrawn;        //"Entries have been drawn" flag
    int Nrows, Ncols, Spacing; //menu dimension
    Mso(MesoList *El, int Nc, int Nr, int Sp, int W, int H,
        int Bd, int Fa, ColorPak &Cp);
    virtual void SetupDim(int &Nc, int &Nr, int &W, int &H);
    virtual int EntryWidth(Meso *Me);
    virtual void SetupEntries(void);
    virtual int IsHz(void) {return Nrows==1;}
    virtual int IsVt(void) {return Ncols==1;}
    virtual void Open(Iso *B, int X, int Y);
    virtual void MoveLoop(MsgPkt &M);
};
```

```

        virtual void Activate(MsgPkt &M);
        virtual void Leave(MsgPkt &M);
        virtual void OnKeyStroke(MsgPkt &M);
        virtual void Forw(MsgPkt &M);
        virtual void Back(MsgPkt &M);
    };

//-----Types used to create a pull-down menu bar-----

class PullDnBar : public Mso
{
    public:
        Mso *SubMso;
        PullDnBar(MesoList *Item, int W, int Sp, ColorPak &Cp);
        virtual void OnKeyStroke(MsgPkt &M);
};

// This type is used to create a pull down menu system

class Pmso : public Wso
{
    public:
        PullDnBar *Bar;           //the pull down menu bar
        Wso *Inner;               //the window region of the menu system
        Pmso(MesoList *Items, int W, int H, int Sp,
            int Ba, int Fa, ColorPak &Cp);
        virtual void Open(Iso *B, int X, int Y);
};

//This type is used to create an entry for the pull down menu bar

class Dmso;                      //forward reference
class Pmeso : public Meso
{
    public:
        Dmso *Vm;                //the drop menu assigned to the menu entry
        Pmeso(char *N, Dmso *D);
        virtual void SwitchFocus(MsgPkt &M);
        virtual void OnKeyStroke(MsgPkt &M);
};

// This type is used to create a drop menu

class Dmso : public Mso
{
    public:
        Pmeso *Parent;           //the top-level menu that the drop menu
                                //is assigned to
        Dmso(MesoList *Items, int W, int Ba, int Fa, ColorPak &Cp);
        virtual void OnClose(MsgPkt &M);
        virtual void OnKeyStroke(MsgPkt &M);
};
#endif

```

A.13 Header File of Text Button Class and Icon Button Class

```

//gbutton.h: graphics button class definitions

#ifndef H_GBUTTON
#define H_GBUTTON

#include "grphscrn.h"

typedef void (*ActionProc) (Wso *Src, MsgPkt &M);
typedef void (*DrawProc) (Wso *Src);

```

```

class TextButton : public Wso
{
    public:
        ActionProc Action;
        char Str[80];
        char Font[80];
        TextButton(char *S, char *F, int Ba, int Fa,
                    ColorPak &Cp, ActionProc A);
        virtual void Draw(void);
        virtual void Activate(MsgPkt &M);
        virtual void ChangeText(char *S, char *F);
};

class IconButton : public Wso
{
    public:
        ActionProc Action;
        DrawProc DrawIcon;
        IconButton(DrawProc D, int Ba, int Fa, ColorPak &Cp,
ActionProc A);
        virtual void Draw(void);
        virtual void Activate(MsgPkt &M);
};

//-----Some common button action-----

void NoOp(Wso *Src, MsgPkt &M);

#endif

```

Appendix B. Source Codes of Class CSprite

B.1 Original Source Codes

```
/////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////
//
//
//    SPRITE.CPP:    Member    functions    for    the    CSprite    class.
//
//
//
/////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////

#define STRICT
#include <windows.h>
#include "sprite.h"

CSprite::CSprite ()
{
    mHImage = 0;
    mHMask = 0;
    mHSave = 0;
    mX = mY = 0;
    mWidth = mHeight = 0;
}

CSprite::~CSprite ()
{
    {
        if (mHSave != 0)
            DeleteObject (mHSave);
    }
}

void CSprite::GetCoord (RECT *Rect)
{
    {
        Rect->left = mX;
        Rect->top = mY;
        Rect->right = mX + mWidth;
        Rect->bottom = mY + mHeight;
    }
}

BOOL CSprite::Hide (HDC Hdc)
{
    {
        HDC HMemDC;
        int Result;

        if (mHSave == 0)
            return (FALSE);

        HMemDC = CreateCompatibleDC (Hdc);
        if (HMemDC == NULL)
```

```

        return (FALSE);

// restore window graphics at prior position:
SelectObject (HMemDC, mHSave);
Result = BitBlt
    (Hdc,
     mX,
     mY,
     mWidth,
     mHeight,
     HMemDC,
     0,
     0,
     SRCCOPY);

DeleteDC (HMemDC);

if (Result == 0)
    return (FALSE);

return (TRUE);
}

BOOL CSprite::Initialize (HBITMAP HMask, HBITMAP HImage)
{
    HDC HDCScreen;
    HBITMAP HSave;
    BITMAP ImageBM;
    BITMAP MaskBM;
    int Result;

    Result = GetObject (HMask, sizeof (BITMAP), &MaskBM);
    if (Result == 0)
        return (FALSE);

    Result = GetObject (HImage, sizeof (BITMAP), &ImageBM);
    if (Result == 0)
        return (FALSE);

    if (MaskBM.bmWidth != ImageBM.bmWidth ||
        MaskBM.bmHeight != ImageBM.bmHeight)
        return (FALSE);

    HDCScreen = GetDC (NULL);
    HSave = CreateCompatibleBitmap
        (HDCScreen,
         MaskBM.bmWidth,
         MaskBM.bmHeight);
    ReleaseDC (NULL, HDCScreen);
    if (HSave == NULL)
        return (FALSE);

    if (mHSave != 0)
        DeleteObject (mHSave);
    mHSave = HSave;
    mHMask = HMask;
    mHImage = HImage;
    mWidth = MaskBM.bmWidth;
    mHeight = MaskBM.bmHeight;
    return (TRUE);
}

BOOL CSprite::MoveTo (HDC Hdc, int X, int Y)
{
    HBITMAP HBMHold;

```

```

HDC HMemDC;
HDC HMemDCHold;
RECT RectNew;
RECT RectOld;
RECT RectUnion;

// copy entire affected area of window to working bitmap:
RectOld.left = mX;
RectOld.top = mY;
RectOld.right = mX + mWidth;
RectOld.bottom = mY + mHeight;

RectNew.left = X;
RectNew.top = Y;
RectNew.right = X + mWidth;
RectNew.bottom = Y + mHeight;

UnionRect (&RectUnion, &RectOld, &RectNew);
RectUnion.left -= RectUnion.left % 8;

HBMHold = CreateCompatibleBitmap
    (Hdc,
     RectUnion.right - RectUnion.left,
     RectUnion.bottom - RectUnion.top);
if (HBMHold == NULL)
    return (FALSE);

HMemDCHold = CreateCompatibleDC (Hdc);
SelectObject (HMemDCHold, HBMHold);

BitBlt
    (HMemDCHold,
     0,
     0,
     RectUnion.right - RectUnion.left,
     RectUnion.bottom - RectUnion.top,
     Hdc,
     RectUnion.left,
     RectUnion.top,
     SRCCOPY);

// create another memory device context:
HMemDC = CreateCompatibleDC (Hdc);

// restore window graphics at prior position:
SelectObject (HMemDC, mHSave);
BitBlt
    (HMemDCHold,
     mX - RectUnion.left,
     mY - RectUnion.top,
     mWidth,
     mHeight,
     HMemDC,
     0,
     0,
     SRCCOPY);

// save current window graphics:
BitBlt
    (HMemDC,
     0,
     0,
     mWidth,
     mHeight,
     HMemDCHold,
     X - RectUnion.left,
     Y - RectUnion.top,

```

```

        SRCCOPY);

// display mask bitmap:
SelectObject (HMemDC, mHMask);
BitBlt
    (HMemDCHold,
     X - RectUnion.left,
     Y - RectUnion.top,
     mWidth,
     mHeight,
     HMemDC,
     0,
     0,
     SRCAND);

// display image bitmap:
SelectObject (HMemDC, mHImage);
BitBlt
    (HMemDCHold,
     X - RectUnion.left,
     Y - RectUnion.top,
     mWidth,
     mHeight,
     HMemDC,
     0,
     0,
     SRCINVERT);

// copy working bitmap back to window:
BitBlt
    (HDC,
     RectUnion.left,
     RectUnion.top,
     RectUnion.right - RectUnion.left,
     RectUnion.bottom - RectUnion.top,
     HMemDCHold,
     0,
     0,
     SRCCOPY);

// delete the memory device contexts:
DeleteDC (HMemDCHold);
DeleteDC (HMemDC);

// delete working bitmap:
DeleteObject (HBMHold);

mX = X;
mY = Y;

return (TRUE);
}

BOOL CSprite::Redraw (HDC HDC)
{
    HDC HMemDC;
    register int Result;

    if (mHSave == 0)
        return (FALSE);

    HMemDC = CreateCompatibleDC (HDC);
    if (HMemDC == NULL)
        return (FALSE);

```

```

// display mask bitmap:
SelectObject (HMemDC, mHMask);
Result = BitBlt
    (Hdc,
     mX,
     mY,
     mWidth,
     mHeight,
     HMemDC,
     0,
     0,
     SRCAND);
if (Result == 0)
{
    DeleteDC (HMemDC);
    return (FALSE);
}

// display image bitmap:
SelectObject (HMemDC, mHImage);
Result = BitBlt
    (Hdc,
     mX,
     mY,
     mWidth,
     mHeight,
     HMemDC,
     0,
     0,
     SRCINVERT);

DeleteDC (HMemDC);

if (Result == 0)
    return (FALSE);

return (TRUE);
}

BOOL CSprite::Start (HDC Hdc, int X, int Y)
{
    HDC HMemDC;
    int Result;

    if (mHSave == 0)
        return (FALSE);

    HMemDC = CreateCompatibleDC (Hdc);
    if (HMemDC == NULL)
        return (FALSE);

    // save current window graphics:
    SelectObject (HMemDC, mHSave);
    Result = BitBlt
        (HMemDC,
         0,
         0,
         mWidth,
         mHeight,
         Hdc,
         X,
         Y,
         SRCCOPY);
    if (Result == 0)
    {
        DeleteDC (HMemDC);
        return (FALSE);
    }
}

```

```

    }

    // display mask bitmap:
    SelectObject (HMemDC, mHMask);
    Result = BitBlt
        (Hdc,
         X,
         Y,
         mWidth,
         mHeight,
         HMemDC,
         0,
         0,
         SRCAND);
    if (Result == 0)
    {
        DeleteDC (HMemDC);
        return (FALSE);
    }

    // display image bitmap:
    SelectObject (HMemDC, mHImage);
    Result = BitBlt
        (Hdc,
         X,
         Y,
         mWidth,
         mHeight,
         HMemDC,
         0,
         0,
         SRCINVERT);

    DeleteDC (HMemDC);

    if (Result == 0)
        return (FALSE);

    mX = X;
    mY = Y;

    return (TRUE);
}

```

B.2 Source Codes after Revision

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////
//
//
//  SPRITE.CPP:  Member  functions  for  the  CSprite  class.
//
//
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////

#define STRICT
#include <windows.h>
#include "sprite.h"

```

```

CSprite::CSprite ()
{
    mHImage = 0;
    mHMask = 0;
    mHSave = 0;
    mX = mY = 0;
    mWidth = mHeight = 0;
}

CSprite::~CSprite ()
{
    if (mHSave != 0)
        DeleteObject (mHSave);
}

void CSprite::GetCoord (RECT *Rect)
{
    Rect->left = mX;
    Rect->top = mY;
    Rect->right = mX + mWidth;
    Rect->bottom = mY + mHeight;
}

BOOL CSprite::Hide (HDC HDc)
{
    HDC HMemDC;
    int Result;

    if (mHSave == 0)
        return (FALSE);

    HMemDC = CreateCompatibleDC (HDc);
    if (HMemDC == NULL)
        return (FALSE);

    SelectObject (HMemDC, mHMask);
    Result = BitBlt
        (HDc,
         mX,
         mY,
         mWidth,
         mHeight,
         HMemDC,
         0,
         0,
         SRCAND);
    if (Result == 0)
    {
        DeleteDC (HMemDC);
        return (FALSE);
    }

    // restore window graphics at prior position:
    SelectObject (HMemDC, mHSave);
    Result = BitBlt
        (HDc,
         mX,
         mY,
         mWidth,
         mHeight,
         HMemDC,
         0,
         0,
         SRCINVERT);

    DeleteDC (HMemDC);
}

```

```

        if (Result == 0)
            return (FALSE);

        return (TRUE);
    }

    BOOL CSprite::Initialize (HBITMAP HMask, HBITMAP HImage)
    {
        HDC HDCScreen;
        HBITMAP HSave;
        BITMAP ImageBM;
        BITMAP MaskBM;
        int Result;

        Result = GetObject (HMask, sizeof (BITMAP), &MaskBM);
        if (Result == 0)
            return (FALSE);

        Result = GetObject (HImage, sizeof (BITMAP), &ImageBM);
        if (Result == 0)
            return (FALSE);

        if (MaskBM.bmWidth != ImageBM.bmWidth ||
            MaskBM.bmHeight != ImageBM.bmHeight)
            return (FALSE);

        HDCScreen = GetDC (NULL);
        HSave = CreateCompatibleBitmap
            (HDCScreen,
             MaskBM.bmWidth,
             MaskBM.bmHeight);
        ReleaseDC (NULL, HDCScreen);
        if (HSave == NULL)
            return (FALSE);

        if (mHSave != 0)
            DeleteObject (mHSave);
        mHSave = HSave;
        mHMask = HMask;
        mHImage = HImage;
        mWidth = MaskBM.bmWidth;
        mHeight = MaskBM.bmHeight;
        return (TRUE);
    }

    BOOL CSprite::MoveTo (HDC HDC, int X, int Y)
    {
        HBITMAP HBMHold;
        HDC HMemDC;
        HDC HMemDC1;
        HDC HMemDCHold;
        RECT RectNew;
        RECT RectOld;
        RECT RectUnion;

        // copy entire affected area of window to working bitmap:
        RectOld.left = mX;
        RectOld.top = mY;
        RectOld.right = mX + mWidth;
        RectOld.bottom = mY + mHeight;

        RectNew.left = X;
        RectNew.top = Y;
        RectNew.right = X + mWidth;
        RectNew.bottom = Y + mHeight;
    }

```

```

UnionRect (&RectUnion, &RectOld, &RectNew);
RectUnion.left -= RectUnion.left % 8;

HBMHold = CreateCompatibleBitmap
    (Hdc,
     RectUnion.right - RectUnion.left,
     RectUnion.bottom - RectUnion.top);
if (HBMHold == NULL)
    return (FALSE);

HMemDCHold = CreateCompatibleDC (Hdc);
SelectObject (HMemDCHold, HBMHold);

BitBlt
    (HMemDCHold,
     0,
     0,
     RectUnion.right - RectUnion.left,
     RectUnion.bottom - RectUnion.top,
     Hdc,
     RectUnion.left,
     RectUnion.top,
     SRCCOPY);

// create another memory device context:
HMemDC = CreateCompatibleDC (Hdc);
HMemDC1 = CreateCompatibleDC (Hdc);

// restore window graphics at prior position:
SelectObject (HMemDC, mHMask);
BitBlt
    (HMemDCHold,
     mX - RectUnion.left,
     mY - RectUnion.top,
     mWidth,
     mHeight,
     HMemDC,
     0,
     0,
     SRCAND);

// restore window graphics at prior position:
SelectObject (HMemDC, mHSave);
BitBlt
    (HMemDCHold,
     mX - RectUnion.left,
     mY - RectUnion.top,
     mWidth,
     mHeight,
     HMemDC,
     0,
     0,
     SRCINVERT);

SelectObject(HMemDC1, mHMask);
// save current window graphics:
BitBlt
    (HMemDC,
     0,
     0,
     mWidth,
     mHeight,
     HMemDC1,
     0,
     0,
     NOTSRCCOPY);

```

```

DeletedDC (HMemDC1);

BitBlt
    (HMemDC,
     0,
     0,
     mWidth,
     mHeight,
     HMemDCHold,
     X - RectUnion.left,
     Y - RectUnion.top,
     SRCAND);
// display mask bitmap:
SelectObject (HMemDC, mHMask);
BitBlt
    (HMemDCHold,
     X - RectUnion.left,
     Y - RectUnion.top,
     mWidth,
     mHeight,
     HMemDC,
     0,
     0,
     SRCAND);

// display image bitmap:
SelectObject (HMemDC, mHImage);
BitBlt
    (HMemDCHold,
     X - RectUnion.left,
     Y - RectUnion.top,
     mWidth,
     mHeight,
     HMemDC,
     0,
     0,
     SRCINVERT);

// copy working bitmap back to window:
BitBlt
    (HDC,
     RectUnion.left,
     RectUnion.top,
     RectUnion.right - RectUnion.left,
     RectUnion.bottom - RectUnion.top,
     HMemDCHold,
     0,
     0,
     SRCCOPY);

// delete the memory device contexts:
DeleteDC (HMemDCHold);
DeleteDC (HMemDC);

// delete working bitmap:
DeleteObject (HBMHold);

mX = X;
mY = Y;

return (TRUE);
}

BOOL CSprite::Redraw (HDC HDC)
{

```

```

HDC HMemDC;
register int Result;

if (mHSave == 0)
    return (FALSE);

HMemDC = CreateCompatibleDC (Hdc);
if (HMemDC == NULL)
    return (FALSE);

// display mask bitmap:
SelectObject (HMemDC, mHMask);
Result = BitBlt
    (Hdc,
     mX,
     mY,
     mWidth,
     mHeight,
     HMemDC,
     0,
     0,
     SRCAND);
if (Result == 0)
{
    DeleteDC (HMemDC);
    return (FALSE);
}

// display image bitmap:
SelectObject (HMemDC, mHImage);
Result = BitBlt
    (Hdc,
     mX,
     mY,
     mWidth,
     mHeight,
     HMemDC,
     0,
     0,
     SRCINVERT);

DeleteDC (HMemDC);

if (Result == 0)
    return (FALSE);

return (TRUE);
}

BOOL CSprite::Start (HDC Hdc, int X, int Y)
{
    HDC HMemDC;
    HDC HMemDC1;
    int Result;

    if (mHSave == 0)
        return (FALSE);

    HMemDC = CreateCompatibleDC (Hdc);
    if (HMemDC == NULL)
        return (FALSE);

    HMemDC1=CreateCompatibleDC(Hdc);
    if (HMemDC1 == NULL)
        return (FALSE);

    // save current window graphics:

```

```

SelectObject (HMemDC, mHSave);
SelectObject (HMemDC1, mHMask);

Result = BitBlt
    (HMemDC,
    0,
    0,
    mWidth,
    mHeight,
    HMemDC1,
    0,
    0,
    NOTSRCCOPY);

DeleteDC (HMemDC1);
if (Result == 0)
{
    DeleteDC (HMemDC);
    return (FALSE);
}

Result = BitBlt
    (HMemDC,
    0,
    0,
    mWidth,
    mHeight,
    HDc,
    X,
    Y,
    SRCAND);
if (Result == 0)
{
    DeleteDC (HMemDC);
    return (FALSE);
}

// display mask bitmap:
SelectObject (HMemDC, mHMask);
Result = BitBlt
    (HDc,
    X,
    Y,
    mWidth,
    mHeight,
    HMemDC,
    0,
    0,
    SRCAND);
if (Result == 0)
{
    DeleteDC (HMemDC);
    return (FALSE);
}

// display image bitmap:
SelectObject (HMemDC, mHImage);
Result = BitBlt
    (HDc,
    X,
    Y,
    mWidth,
    mHeight,
    HMemDC,
    0,

```

```
    0,  
    SRCINVERT);  
  
DeleteDC (HMemDC);  
  
if (Result == 0)  
    return (FALSE);  
  
mX = X;  
mY = Y;  
  
return (TRUE);  
}
```

Appendix C. Results of Fuzzy Features of Some Objects



Fig. C.1: Image of an object after thresholding

	Image 1	Image 2	Image 3
Resolution	100×140	117×169	164×165
PIOAC	0.711981	0.720012	0.709528
IOAC	0.594143	0.588833	0.62694
1/IOC	25.108078	25.698618	24.219334

Table C.1: Results of fuzzy features of the object in Fig. C.1



Fig. C.2: Image of an object after thresholding

	Image 1	Image 2	Image 3
Resolution	194×181	164×1167	122×146
PIOAC	0.750616	0.754512	0.756311
IOAC	0.738676	0.61275	0.595217
1/IOC	26.589933	26.43062	27.300886

Table C.2: Results of fuzzy features of the object in Fig. C.2



Fig. C.3: Image of an object after thresholding

	Image 1	Image 2	Image 3
Resolution	117×135	114×165	178×151
PIOAC	0.701025	0.696752	0.700745
IOAC	0.690155	0.709144	0.707307
1/IOC	22.659298	21.297632	21.68029

Table C.3: Results of fuzzy features of the object in Fig. C.3



Fig. C.4: Image of an object after thresholding

	Image 1	Image 2	Image 3
Resolution	97×79	115×102	174×134
PIOAC	0.685907	0.683685	0.687476
IOAC	0.690852	0.72191	0.729327
1/IOC	22.223083	21.030231	18.772419

Table C.4: Results of fuzzy features of the object in Fig. C.4