LABORATORY FOR CONCURRENT COMPUTING SYSTEMS

COMPUTER SYSTEMS ENGINEERING
School of Electrical Engineering
Swinburne Institute of Technology
John Street, Hawthorn 3122, Victoria, Australia.

Some Approaches to Path Planning for Autonomous Vehicles

W. Shang
G.K. Egan

Technical Report 31-030

Abstract

Among the various approaches to optimal path planning, the Hypothesis and Test Method, the Penalty Functions Method, and the Explicit Free Space Method are believed to be the most influential. They are compared briefly in this report in order to identify a preferable alternative in terms of execution and applicability. It is believed that the Explicit Free Space Method has certain advantages over the others in this respect. The algorithm presented here is derived from the Lozano-Perez Method, and consists of three steps: Constructing the Cspace Obstacles, Representing the Free Space, and Searching for a Collision-Free Path. In constructing two-dimensional workspace, we use the Lozano-Perez algorithm: $CO_A(B) = B\text{-}A_o$. An "approximate algorithm" is adopted as an alternative to facilitate the computation of three-dimensional workspace.It is based on the modelling of 3-D obstacles B_i as pyramids, and the moving object A as any shape convex polyhedron. A " visibility graph " is used to establish a configuration space that can be mapped intoa graph of vertices between which the object can move on a straight line. The graph is traversed by way of Nilssion's Artificial Intelligence Method.

SOME APPROACHES TO PATH PLANNING FOR AUTONOMOUS VEHICLES

W. Shang and G.K. Egan

1. Introduction

The use of the first and some of the second generation robots have posed enormous difficulty because of the low level techniques available for programming them. Joint Level programming involves teach-by-show methods, and Robot Level language is based on the use of textual languages. Neither of these methods is efficient or convenient since a very large number of robot instructions have to be provided to the robot controller. This difficulty raises the need for higher level languages which enable a robot to be programmed in a more simplistic manner. Two higher level programming systems that have been gaining momentum recently include Task Level and Object Level methods. Task Level defines the objects with the help of design data, from which together with knowledge of the robot, the system works out a plan to perform the task and necessary information to enable the robot to carry out the task. Object Level results from simplifying the programming process and aims to specify the desired task in terms of operations on the objects instead of specifying the motion of the robot.

The world model, which contains the objects and robot, is partially within the knowledge of an object level programming system. The sequence of tasks, may be, therefore, defined more easily, and collision-free trajectories may also be computed by the system. The principal problems involved at this level are related to world modelling, automatic grasping and collision free path planning. The purpose of this paper is to discuss problems in collision-free path planning.

2. Path Planning

The aim of path planning is to determine a continuous collision-free path between the initial position and the goal position given an initial position, a goal position and a set of obstacles in the workspace. Extensive research has been taking place since the 1970's [Fu 85], with varying degrees of success, but the planning of the optimal path remains to be explored. Methods used for this purpose fall into three categories:

1. Hypothesis and Test Method;
2. Penalty Functions Method;
3. Explicit Free Space Method.

The Hypothesis and Test Method starts with a hypothetical path, and explores possibilities of collision of the robot with other obstacles. The path is modified in case of collision until it becomes collision-free. Simplicity is a chief advantage of this method, as most of the tools needed are already available in the geometric modelling system. Its main disadvantage is the difficulty involved in generating the correction, especially when the workspace is cluttered with obstacles.

W. Shang is a postgraduate student in the School of Electrical Engineering and a Researcher in the Laboratory for Concurrent Computing Systems at the Swinburne Institute of Technology, John Street, Hawthorn 3122, Australia, Phone:+61 3 819 85
E-mail: wss@stan.xx.swin.oz.au.

G.K. Egan is Professor of Computer Systems Engineering and Director of the Laboratory for Concurrent Computing Systems at the Swinburne Institute of Technology, John Street, Hawthorn 3122, Australia, Phone:+61 3 819 8516, E-mail: gke@stan.xx.swin.oz.au.

In the Penalty Function Method, penalty functions whose values derive from the proximity of the obstacles need to be defined. The values of these functions increase as the robot moves closer to the obstacles. This method makes it easy to deal with a larger number of obstacles and constraints, but the penalty functions are usually difficult to specify.

The Explicit Free Space Method has proved to be the most popular one. A few algorithms [BE91] have been proposed in this class; amongst them the most influential are the Brooks [BR83] and Lozano-Perez [LP81],[LP83] algorithms. Brooks' method represents the free space as overlapping generalized cones and the volume swept by the moving object as a function of its orientation. Conceptually, finding a collision-free path is equivalent to comparing the swept volume of the object with the sweepable volume of the free space. See Fig.1. In a relatively uncluttered workspace, Brooks' method is fast and efficient. Its major drawback is that paths can only follow the spines of the generalized cones used to represent free space. It does not perform well in cluttered environments, for there are not sufficient generalized cones to allow for a rich choice of path. The applicability of this method has been compromised due to this major drawback.

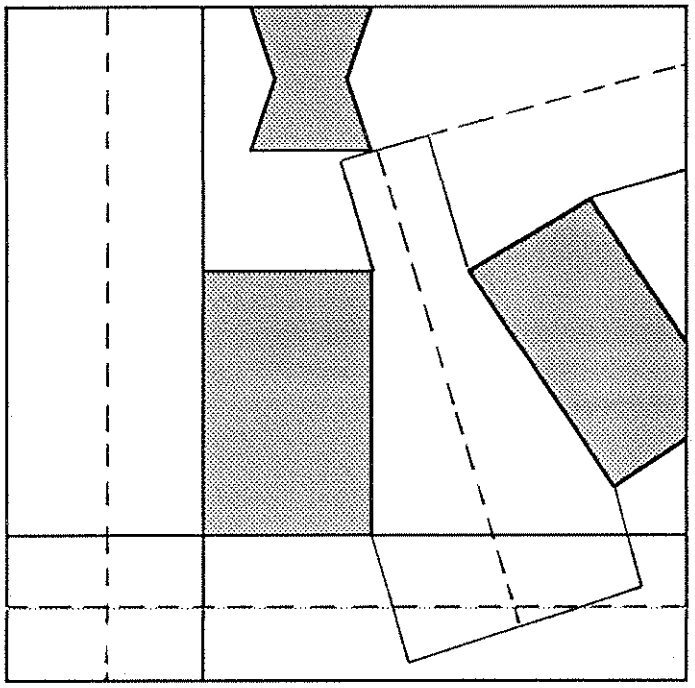


Fig.1 Generalized Cones Generated by Two Obstacls and Workspace Boundary.

Lozano-Perez[LP 81][LP 83] represented the space in terms of the configuration space or Cspace, as it is often called, which virtually means transforming the object into a point, and enlarging the obstacles accordingly. In practical terms, a collision-free path is one that does not intersect with any of the expanded obstacles. This method performs reasonable well so far as only translation is concerned. When rotation is considered, however, configuration space has to be generated by means of approximations, and the computation task is significantly larger.

3. Lozano-Perez Method

The algorithm discussed in this paper is based on the Lozano-Perez Method[LW 79][LW81][LW83]. Cspace is obtained by shrinking moving object and expanding obstacles

accordingly. A "visibility graph" is created to represent the free space by a graph of vertices between which the object can move in a straight line. The nodes of the graph correspond to the corners of the obstacles, and the arcs represent the link between adjacent nodes which can see each other. The algorithm used in this paper consists of three steps:

1. Constructing the Cspace Obstacles;
2. Representing the Free Space; and
3. Searching for a Collision-Free Path.

3.1 Constructing the Cspace Obstacles

The basic problems in path planning are Findspace and Findpath problems. Findspace is to find out where a moving object A can be placed within a certain range R so that the object A does not intersect with any obstacles Bi already in the range R. Findpath is to find out how to move a moving object A from one location (S) to another location (G) without causing collision with the obstacles Bi. See Fig. 2. In the Findspace and Findpath Algorithm described here we use geometric objects, called configuration space objects, that represent all the positions of object A that cause collision with the Bi. Given these objects, the problems are equal to finding a single point (a space of A) or a path (a sequence of position of A), outside of the configuration space obstacles. If the point does not intersect with any of the expanded obstacles, it is certain that the moving object will not collide with any obstacles, as is shown in Fig.3.[LW 81]

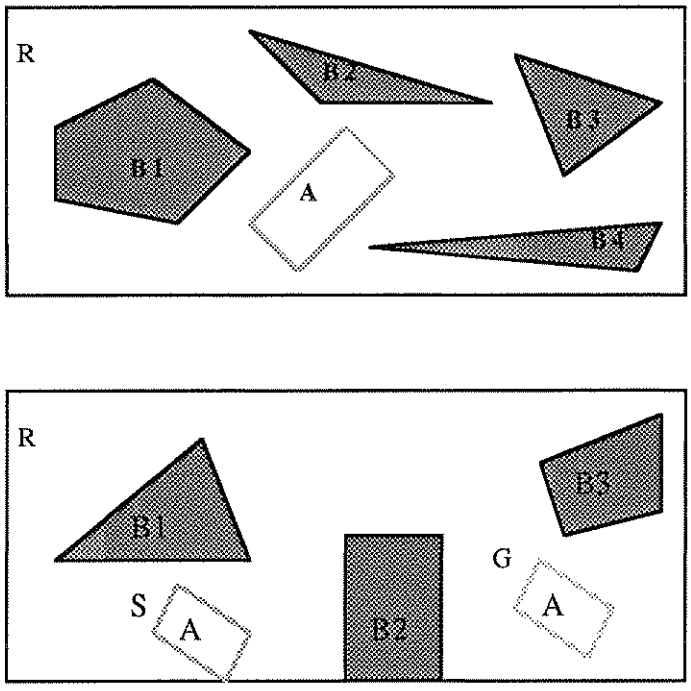


Fig.2 (a) Findspace for A within Range R Clutted with Obstacles Bi.

(b) Findpath for A from S to G within Range R Clutted with Obstacles Bi.

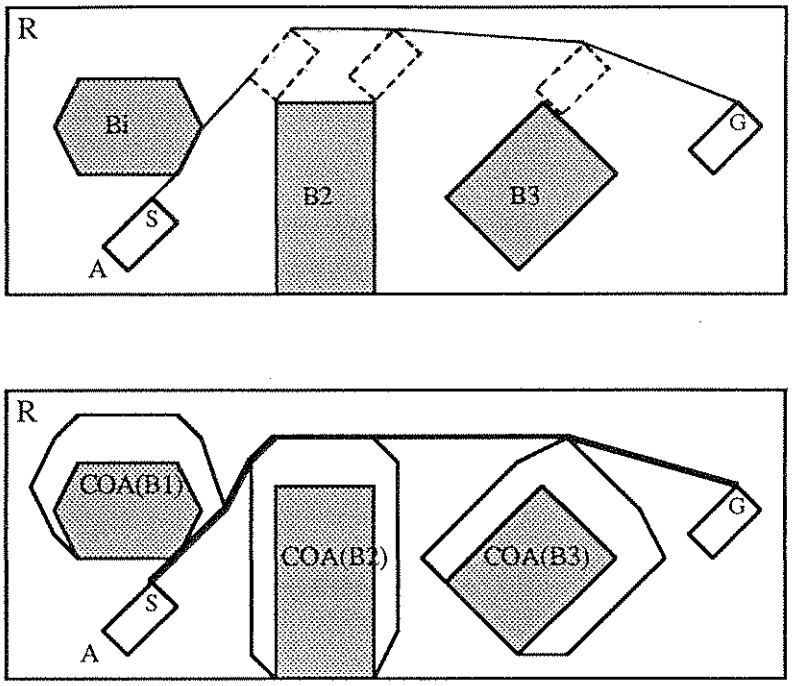


Fig.3 (a) Initial: Moving Object A and Obstacles Bi.

(b) Shrunk Moving Object A and Expanded Obstacles COA(Bi)

Representing the position of objects requires specifying all their degrees of freedom, both translation and orientation. We use configuration to denote the degree of freedom. The configuration of a polyhedron is a set of independent parameters that characterize the position of every point of the polyhedron. The space of configuration for a polyhedron A is called its configuration space and is denoted CspaceA. In CspaceA, the set of configuration of A where A overlaps obstacle B is denoted $CO_A(B)$. $CO_A(B) = \{x \in CspaceA \mid (A)x \cap B \neq 0\}$. We regard the CspaceA obstacle due to B. $CO_A(B)$ as an enlarged version of obstacle B, whereas the moving object A can be represented by a point. Similarly those configurations of A where A is completly inside B is denoted $CI_A(B)$. $CI_A(B) = \{x \in CspaceA \mid (A)x \subseteq B\}$. As such finding a path in a range R amounts to identifying a configuration of A or a connected set of A, which is a path outside $CO_A(B)$, but inside $CI_A(R)$.

3.1.1 Two-Dimensional Cspace

In two-dimensional workspaceR , we use the Lozano-Perez's algorithm[LP 83]: $CO_A(B) = B - Ao$ to compute $CO_A(B)$. Here Ao is A in its initial configuration.

3.1.2 Three-Dimensional Cspace

$CO_A(B)$ in three-dimensional R is more complicated than the Lozana-Perez algorithm above can tackle. Not only the edges of the obstacles but also the faces have to be increased significantly. An "approximate algorithm" has been proposed as an alternative to facilitate the computation of three-dimensional R. Although the Cspace obstacle worked out with this method is an approximation and some free space may be ruled out, it performs reasonably well in computation, for it is dimension independent and the number of faces does not need to be increased. The algorithm used is based on the modelling of 3-D obstacles Bi as pyramids, and the moving object A as any shape convex polyhedron.

Comparing any two vertices of the object A, the distance between them can be determined. Let the maximum distance be denoted *d*, and the middle point of these two vertices as the reference point. $CO_A(B_i)$ is obtained by expanding every vertex of B_i *d/2*. Then, we have an expanded version of B_i ---- $CO_A(B_i)$, which is equivalent to B_i in shape. In this way, the Cspace obstacles can be constructed. See Fig. 4.

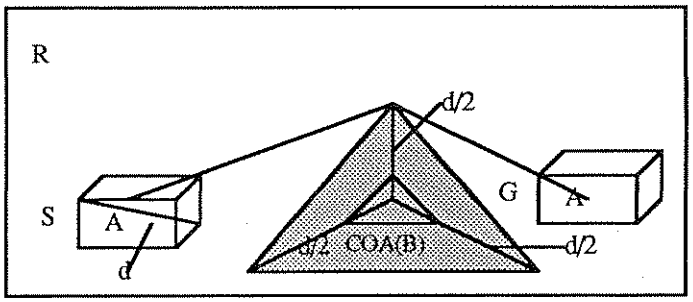


Fig.4 Moving Object A and COA(B) by Using Approximate Method.

3.2 Representing Free Space

Now that the Cspace obstacles $CO_A(B)$ are constructed, the next step is to represent the free space and obstacles. For this purpose it is useful to visualize the free space as a graph containing nodes which correspond to certain states of space, and arcs that represent the relationships between states. A problem state, or a state, is a particular problem situation or configuration [FU 87]. In the Visibility Graph Algorithm, for instance, a node corresponds to a vertex of $CO_A(B_i)$, start node S or goal node G, and an arc links two nodes which can see each other. In the Cell Decomposition Algorithm, however, a node stands for a free cell, while two adjacent cells are linked by an arc when travel between them is possible. Here, we only discuss the Visibility Graph.

3.2.1 Visibility Graph (V-Graph)

The Visibility Graph is defined as: A node set N is V {S,G}, where V is the set of all vertices of obstacles $CO_A(B_i)$. A link set is a set of links (ni,nj) such that a straight line connecting ni and nj does not intersect any obstacles $CO_A(B_i)$. Fig.5.

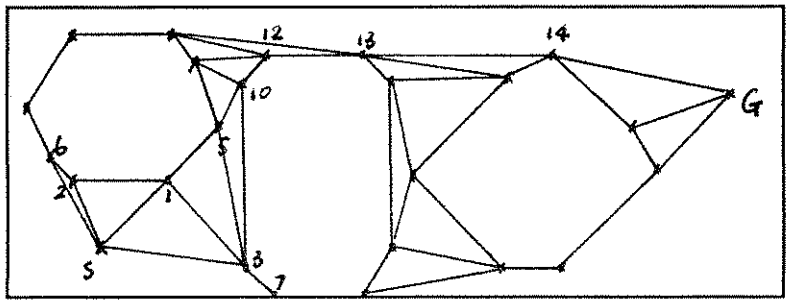


Fig.5 The Visibility Graph Based on the Obstacles for A Correspands to Fig.3(b).

The data structure of a node in the terms of Pascal is as follows:

```
point = record
            x,y,z:real;
            cost:real;
            visit:boolean;
        end;
```

where (x,y,z) is the position of the node in Cardision Coordinate. An arc links node n_i and node n_j provided that a true visibility function exists between them. Fig.6 is the search tree corrsoponding to Fig.5.

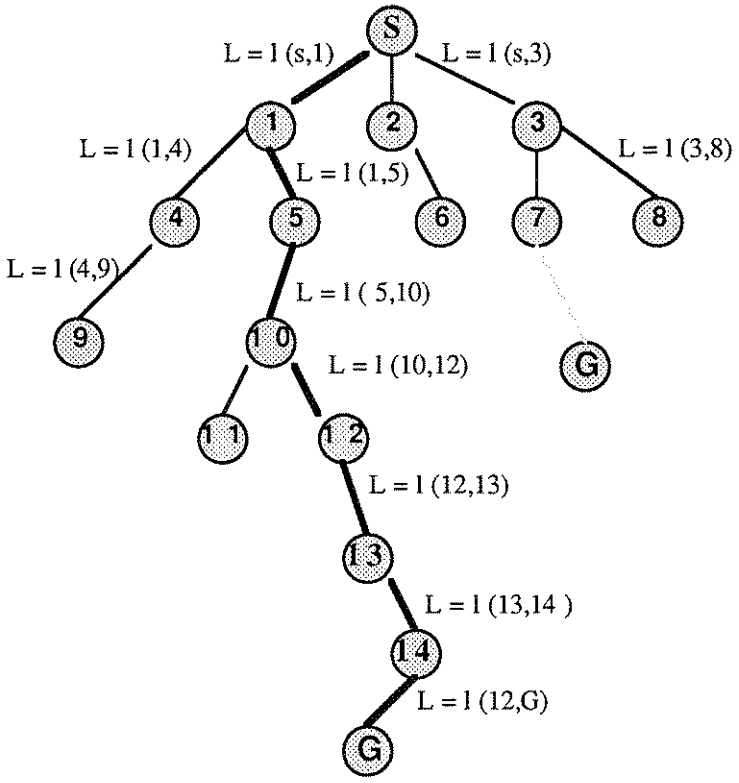


Fig.6 Search Tree Corrosponding to Fig.5.

Nodes 1, 2, 3, ...represent the vertices of $CO_A(B)$. S, G represent start node and goal node. Arc between node i and node j represents visibility function l (i,j) is true, that is, node i and node j can see each other.

3.3 Searching for a Collision-Free Path

Free space having been represented by way of a graph, the next question is how to traverse the graph generated above from a node containing start point to a node containing the goal point, and finding the optimal path in terms of graph nodes and consequently a geometric representation of this path. In the terms of V-Graph algorithm, the definition of the path finding problem is: Given a node set N, find a sequence of node from node S to node G by way of an ordered set of intermediate nodes $n_1,n_2,...n_k$, so that the visibility function for each node pair :

$$L_1 = l\,(S, n_1);$$
$$L_2 = l\,(n_1, n_2);$$
$$\vdots$$
$$L_{k+1} = l\,(n_k, G)$$

is true. The cost function $C = C_i(n_i, n_j)$ is minimized during the process.

Nilssion proposed an Artificial Intelligence Method, i.e., "applying operators to state descriptions until an expression describing the goal state is obtained." [NI 71]. The operator, when applied to a state, transforms the state into another state. In the case under discussion, state description equals node n_i; start state and goal state are node S and node G; and the operator corresponds to visibility function. The operators are used to calculate the successors of node n_i, and then check every successors of n_i to see if they are node G. The flow chart is:

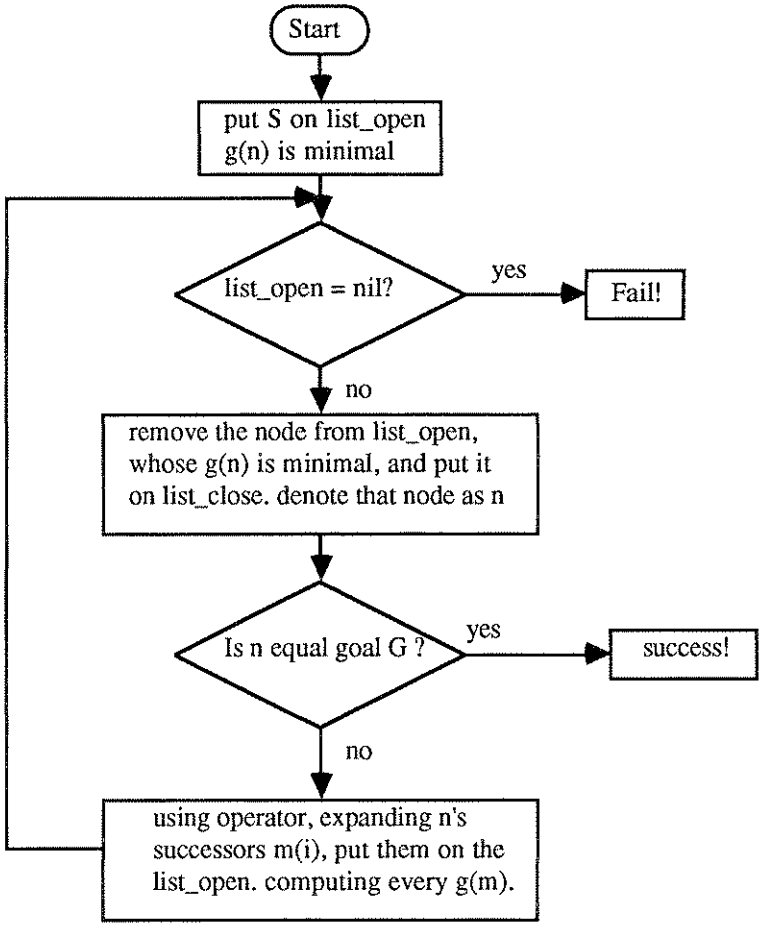


Fig.7 Path Searching Flow Chart.

A path is obtained by tracing back list_close. If no collision-free path exists, exit with fail. Fig.6 is a tree producted by path searching method above. The darker line in Fig.6 is obtained by using the flow-chart above. All the nodes on darker line are put on list_close.

The following is the data structure of list_open and list_close as represented in Pascal:

```
ptr_open = ^ open;
open = record
          spot:point;
          next:ptr_open;
       end;

ptr_close = ^ close;
close = record
          id:ptr_open;
          next:ptr_close;
       end;
```

At every step of the algorithm, list_open is revised. For node n, for instance, list_open includes node n and all its successors, m_i. They are compared in order to find a node m_k, which has the minimum cost estimate. Node m_k then is placed on list_close and the visit is now true. If m_k is not the goal node, all successors of m_k will be expanded and moved onto list_open. Then a node with minimum cost can be found among those nodes by repeating the above process.

3.3.1 Expanding Successors

The operators with which to expand the successors are visibility function. The question now is whether m is n's successor. A line function through m and n can be obtained by using m and n:

$$x(t) = m.x + (n.x - m.x)*t \quad .....(1)$$
$$y(t) = m.y + (n.y - m.y)*t \quad .....(2)$$
$$z(t) = m.z + (n.z - m.z)*t \quad .....(3)$$

A plane function can be obtained by using three of the four vertices of a pyramid:

$$Ax + By + Cz = D \quad .....(4)$$

The variable of t emerges from(1), (2), (3), and (4). If $(t > 1)$ or $(t < 0)$, the point $(x(t), y(t), z(t))$ is not within segment mn. If $t <= 1$ and $t >= 0$, then point $(x(t), y(t), z(t))$ is within the segment mn. The point $(x(t), y(t), z(t))$ is the intersection of segment mn and the plane. If $(x(t),y(t),z(t))$ is within face 1,2,3,4 then m is not a successor of n, i.e., m and n can not see each other.The flow chart of the expanding successors is:

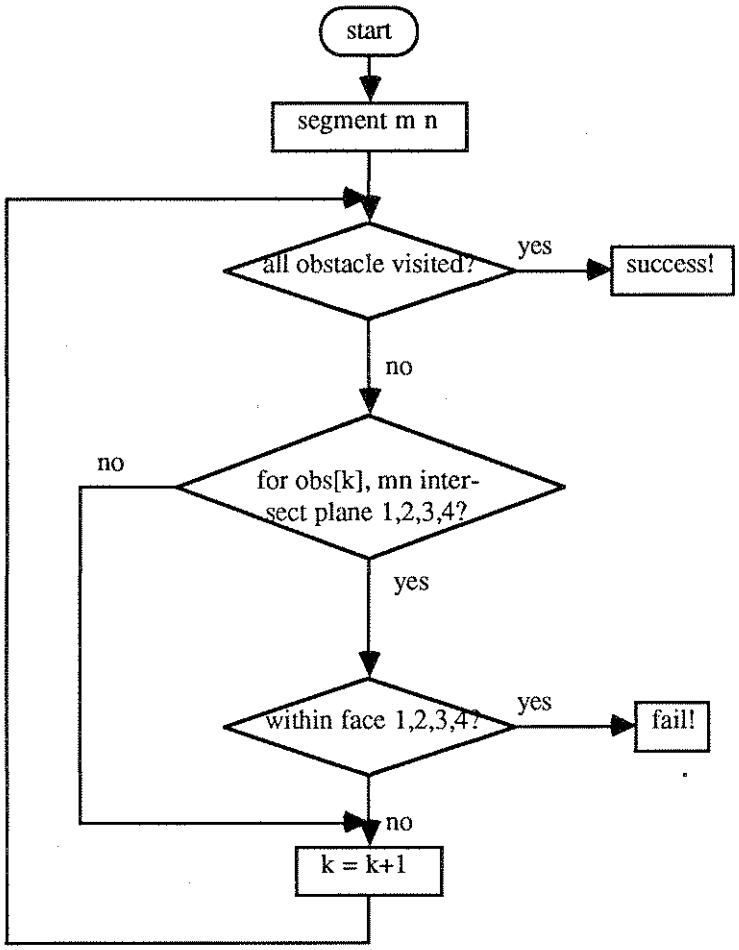
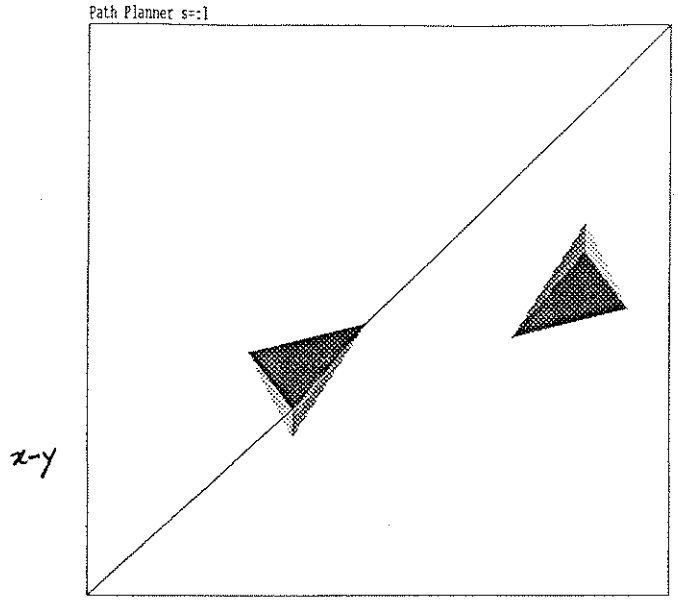


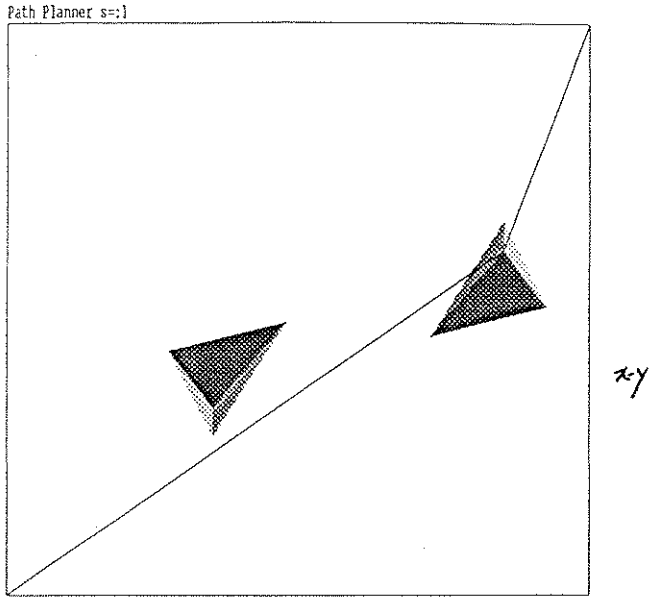
Fig.8 Flow Chart for Expanding Successors

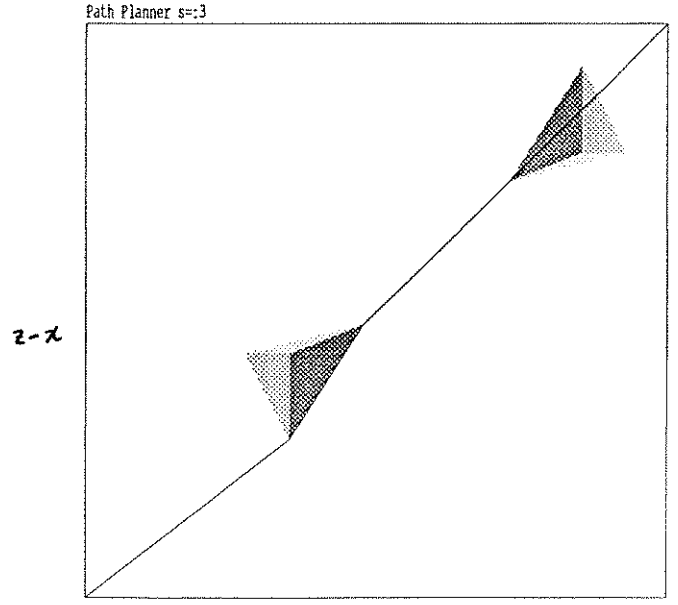
3.3.2 Cost Function

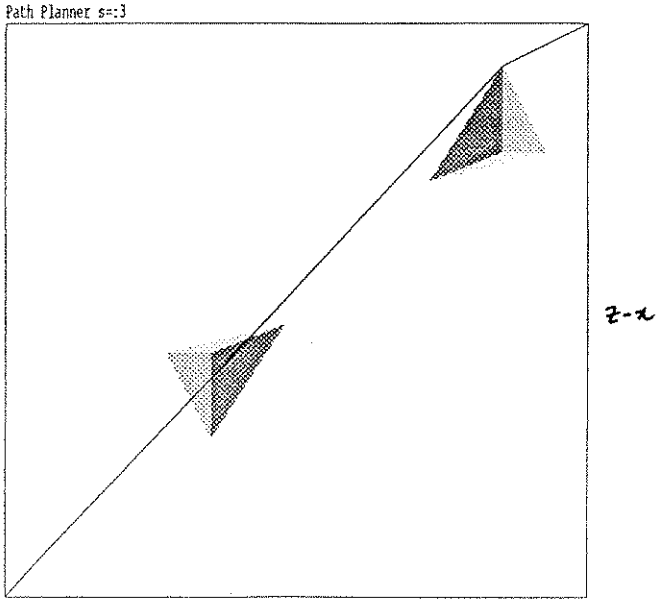
In the case of all obstacles, if m and n can see each other, then m is one of the successors of n. Among all the successors, one with minimum cost function is chosen and put on list_close.We use f(n) for the cost function of node n. As node n is related to node S and node G in terms of cost, f (n) = g(n) + h(n), where g(n) indicates the cost from start node S to node n, and h(n), the cost from node n to goal node G. The cost function can be tailored to the requirements of a certain problem environment, such as the functions of distances in parameter space or functions of energy or both of them.

Fig.9 and Fig.10 are the results of V-Graph. The cost functions are based on distance functions.


Path Planner s=:1
x-y


Path Planner s=:1
x-y


Path Planner s=:3
z-x


Path Planner s=:3
z-x

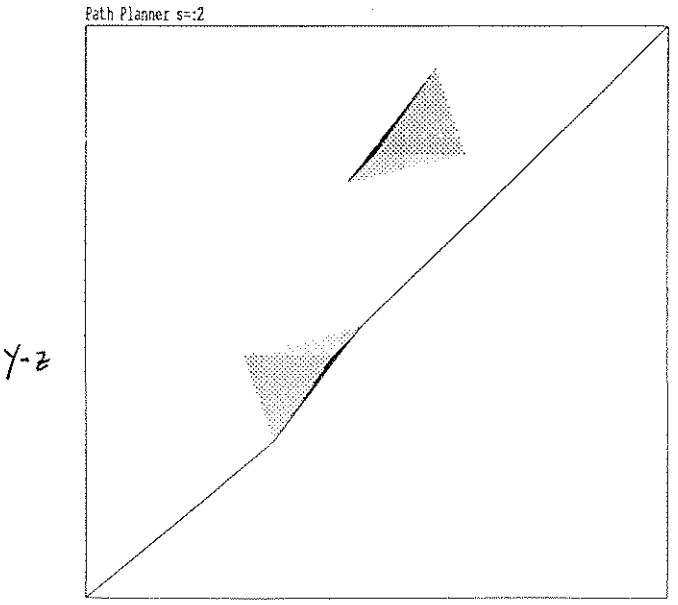

Path Planner s=:2
Y-Z

Fig. 9

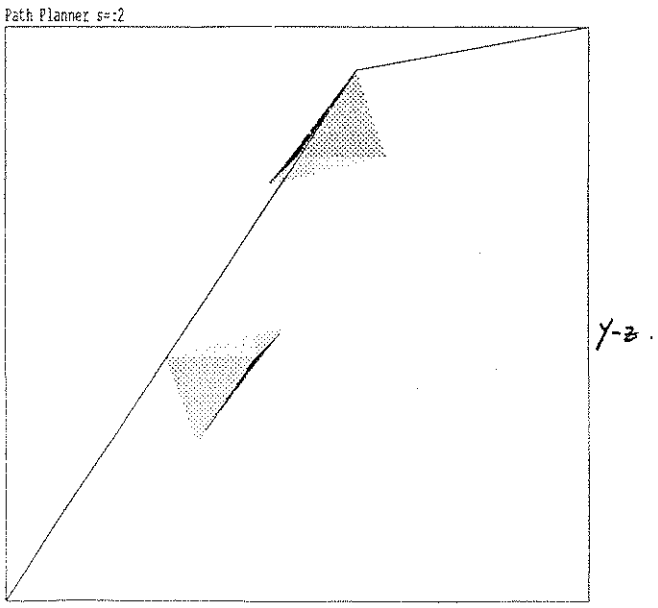

Path Planner s=:2
Y-Z

Fig. 10

4. Conclusions

We have discussed the process of path finding with V-Graph Algorithm. It is an explicit approach that guarantees an optimal collision-free path to be found in two-dimensional workspace. In the computation of three dimensional workspace, however, an optimal path around the vertices of the obstacles cannot be guaranteed, especially for convex polyhedra of varied shapes, because an optimal path may exist among faces of the polyhedra. An alternative is the Cell Decomposition approach, which is free from the constraints of the shape of the obstacles. We will discuss this approach in a later report.

Acknowledgements

We wish to thank all memebrs of the Laboratory for Concurrent Computing Systems at the Swinburne Institute of Technology for their support. Special thanks to Pau Chang, who constantly provided valuable suggestions and to Brendan Rogers, for reading the paper.

References

[LW 79] Tomas Lozano-Perez and Michael A. Wesley, 'An Algorithm for Planning Collision-Free Paths Among Polyhedra Obstacles', Communications of the ACM, Vol. 22, No.10, pp 560-570, Oct. 1979.

[LP 83] Tomas Lozano-Perez, 'Spatial Planning: A Configuration Space Approach', IEEE Transactions on Computing, Vol. C-32, No. 2, pp 108-119, Feb. 1983.

[NI 71] Nils J. Nilsson, *Problem-Solving Methods in Artificial Intelligence*, pp 43-79, New York, McGraw-Hill Book Company, 1971.

[BR 83] Rodney A. Brooks, 'Solving the Find-Path Problem by Good Representation of Free Space', IEEE Transactions on Systems, Man and Cybernetics, Vol. SMC-13, No.3, pp 190-197, March / April 1983.

[BE 91] C.B.Besant, 'The Application of Artificial Intelligence to Robotics', pp 175-191, Parallel Processing and Artificial Intelligence, 1990.

[LA] Jean-Claude Latombe, 'Toward Automatic Robot Programming', Proceedings .....

[FU 87] K.S.Fu, R.C.Gonzalez and C.S.G.Lee, *Robotics: control, sensing, vision, and intelligence,* pp 400-450, New York, McGraw-Hill Book Company, 1987.

[LP 81] Tomas Lozano-Perez, 'Automatic Planning of Manipulator Transfer Movements', IEEE Transactions on Systems, Man and Cybernetics, Vol.SMC-11, No. 10, pp 681-698, Oct. 1981.