PATH PLANNING AMONG MOVING OBSTACLES USING SPATIAL INDEXING

Kikuo Fujimura
Hanan Samet

Computer Science Department and
Center for Automation Research
University of Maryland
College park, MD 20742, USA

ABSTRACT

A method is presented for planning a path in the presence of moving obstacles. Given a set of polygonal moving obstacles, we focus on generating a path for a mobile robot that navigates in the two-dimensional plane. Our methodology is to include time as one of the dimensions of the model world. This allows us to regard the moving obstacles as being stationary in the extended world. For a solution to be feasible, the robot must not collide with any other moving obstacles, and, also, it must navigate without exceeding the predetermined range of velocity, acceleration, and centrifugal force. We use a spatial index to facilitate geometric search for the path planning task. Computer simulation results are also presented to illustrate the feasibility of this approach.

1. INTRODUCTION

Planning a collision-free path is one of the fundamental requirements for a mobile robot to execute its tasks [Broo86, Davi85, Crow85, Mora83]. Much of the prior work on this topic is concerned with methods for generating a path among stationary obstacles [Loza79, Broo83, Kamb86]. It should be clear that a robot that can cope with moving obstacles will be capable of performing a much larger and more complex class of tasks. For example, the capability of handling moving objects is essential in implementing concurrent motion for multiple robots.

The robot motion planning among stationary obstacles may be divided into two stages, i.e., the determination of a collision-free path and then the determination of control (velocity, acceleration, etc) along the fixed path. When obstacles are moving, we cannot simply separate these two stages. For example, a collision-free path for a robot that moves at some velocity may not be collision-free if it moves at a different velocity. Here we present a method to plan motion in the presence of moving obstacles, i.e., determine a path and velocity along the path such that the path is collision-free.

1.1. Statement of the problem

Our objective is to navigate a mobile object from the start position to the goal position in the presence of a given set of moving polygonal obstacles on a two-dimensional plane. We require that the mobile object move on a flat two-dimensional plane, avoid obstacles, observe a predetermined range of velocity and acceleration, and not negotiate any curve beyond a velocity that exceeds a predetermined upper limit on the allowable centrifugal force. We also assume that all the information regarding obstacles (such as shapes, movement directions, etc.) is known *a priori* to the robot. We consider the mobile robot to be a point. In addition, we assume that each obstacle is a polygon which moves at a constant speed without rotation. These assumptions are not essential to our method, but they make our presentation simpler. We also consider planning a path in the presence of some uncerainty in the movements of obstacles.

The support of the National Science Foundation under Grant DCR-86-05557 is gratefully acknowledged.

1.2. Space-time

Our approach to the problem is to use a three-dimensional space in which time is the third dimension. This space is usually called space-time. An object, say O, moving in a two-dimensional plane can be regarded as a three-dimensional stationary object whose volume is the trajectory that is swept as it moves. If a point (x,y,t) is inside that volume in space-time, then the two-dimensional point (x,y) is occupied by object O at time t. Therefore, an interference between two objects in three-dimensional space means that a collision has occurred in the two-dimensional plane. Note that two different objects which occupy the same location at different times don't collide and will occupy different locations in space-time.

Now, our task becomes one of finding a collision-free path in space-time. Given a start position, time, and goal position, the search process generates a path for the mobile object that connects the start position to the goal position. Minimizing the t value of the goal position will be a main concern in the problem of time optimal path generation. Since our third dimension is time, we have to be careful so as not to choose an unrealistic path—e.g., traveling in the negative time direction. Moreover, the path must satisfy the predefined conditions with respect to velocity, acceleration, and curvature.

1.3. Related work

There are some approaches that examine the moving objects. Samet and Tamminen [Same85a] describe a way to add the time dimension to a CSG. 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 Rosendale [Este83]. They divide the time dimension as well as the other dimensions recursively to quickly locate the collision point between two moving objects. Kant and Zucker [Kant86] attempt to

determine an appropriate velocity along a fixed trajectory. They employ static path planning techniques to solve a fixed-path trajectory planning problem in time-varying environment. Reif and Sharir [Reif85] have made the first theoretical step toward the dynamic path planning problem. They prove that a path among a set of moving polygons can be found in polynomial time with respect to the total number of vertices of the obstacle polygons. Erdmann and Lozano-Perez [Erdm86] describe a planner for moving objects that constructs a configuration space each time the environment changes. Their method is based on stacking two-dimensional planes, where each plane represents a configuration space at some time. In this approach, two adjacent planes are inspected to see if a path exists between these two planes. Bolles and Baker [Boll85] construct a three-dimensional solid from an image sequence for motion analysis. Our method also incorporates time explicitly in the model world, which makes it easier to analyze factors like velocity and acceleration. To substantially facilitate geometric search, we use a spatial index. This is described in Section 2. Section 3 describes the search strategy, and Section 4 contains experimental results. Section 5 contains some concluding remarks.

2. SPATIAL INDEXING

We assume that the motion of the obstacles doesn't involve rotation. As long as a polygon moves at a constant speed without rotation, the trajectory (i.e., the volume swept by the polygon) becomes a polyhedron in three dimensions. A polyhedron can be modeled in terms of its vertices, edges, and surfaces. The representation method based on the relationship between these three geometric entities is usually called the boundary representation [Requ82]. Boundary representations are desirable for retrieving topologically connected information; however, they still require some supplementary work to perform geometric operations such as neighborhood finding. Since neighborhood search accounts for the major part of path search process, we use a spatial index to facilitate its computation. A tree structure serving as an index to the model world yields efficient access to a location.

Both the mobile object and the obstacles are defined in a world with bounded x, y, and t values. A point in the space is represented by (x,y,t), where $x1<x<x2$, $y1<y<y2$, and $t1<t<t2$. x and y are measured in terms of distance while t corresponds to time. Usually, it is convenient to let $x1=y1=t1=0$ and $x2=y2$. Note that time is also bounded. In this world, every motion of an object on the two-dimensional plane during the time period between $t1$ and $t2$ is represented as a three-dimensional object. Our index tree is built by repeatedly subdividing three dimensional space-time into eight subspaces of equal size called *cells*, until each cell satisfies either of the following conditions:

(A) A cell contains part of the trajectory of a vertex of an obstacle.
(B) A cell doesn't contain any part of the trajectory of a vertex, but contains part of the trajectory of an edge of an obstacle.
(C) A cell doesn't contain any part of the trajectory.
(D) A cell is entirely contained in the trajectory.

The cells defined by these criteria are respectively called *vertex* cells, *edge* cells, *empty* cells, and *full* cells. Similar methods based on regular decomposition of the space have been studied in the octree domain [Meag82, Same85b, Ayal85, Carl85].

Figure 1 illustrates the concept of the index tree described above. Suppose that an object moves in the x direction (Figure 1a). A solid line and a dashed line depict the initial and final position of the object. Figure 1b shows the volume swept by the object in space-time. The index tree for this three-dimensional image is shown in Figure 1d, where the cell numbering convention of Figure 1c is used. An index tree must be built corresponding to the volume swept by the motion of objects along the given trajectories. Building an index tree can be performed in the following way. Initially, the entire universe is treated as a sigle cell which is represented as a tree containing one node. If any of conditions (A)-(D) are violated by this cell, then the cell is subdivided and resulting cells are checked for violation of conditions (A)-(D). This process is applied recursively. For detailed discussion of this conversion, see [Ayal85, Fuji85]. Note that we don't decompose the universe to the voxel level as in [Meag82].

3. PATH SEARCHING

This section describes how the search procedure generates a collision-free path using the index tree introduced in Section 2.

3.1. Control Points

We define a point called a *C-point* (control point) in the space. The sequence of these C-points forms a skeleton of the final path. We consider a C-point as an ordered pair consisting of a 2D location (i.e., (x,y)) and a t-value. A t-value represents the time at which the mobile object passes that C-

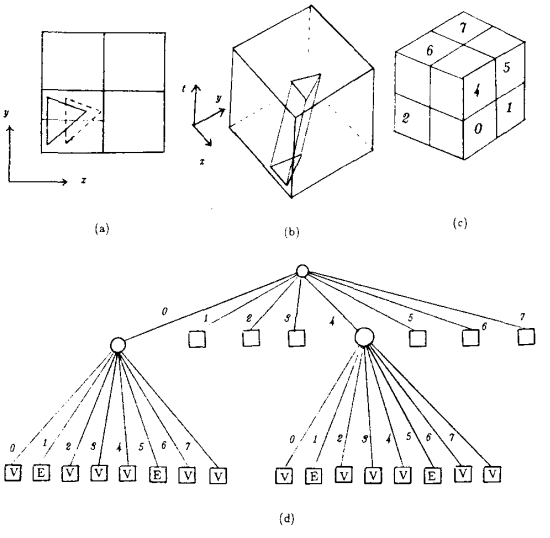

Figure 1

(a) An object moving in the x direction.
(b) The time-space image of the object in (a).
(c) The cell numbering convention in an octree structure.
(d) A tree representation of the space (b). V and E represent a vertex cell and an edge cell, respectively.

point. The x and y values of a C-point take on a discrete value. Let a square denote the projection of a cell onto the x–y plane. We define the arrangement of these C-points such that the x and y coordinates of C-points lie only at either of the following nine locations in a square, i.e., one at the center, one at each of its four corners (for a total of four) and one at the middle of each edge (for a total of four). The t-value is assigned in the search stage. In other words, the search procedure first chooses the next location to go to from the nine types described above. Then it determines appropriate velocity. This determines the t-value at that C-point. Since we have a choice as to the velocity (or acceleration) value, the t-value at each C-points can vary depending on velocity values that have been chosen. Two identical sequences of C-points with different sets of t-values, hence, represent two different motions.

In order to make our search feasible, we pose two restrictions with respect to the choice of acceleration. The mobile object can change its acceleration and direction only at the C-points, while it retains the same acceleration between two C-points. This is one constraint we impose on our path. Another constraint is that we assume that acceleration takes on discrete values. These restrictions are necessary, since otherwise there can be infinite possibilities as to when and where to change acceleration. Since we can let the acceleration be 0, navigating at a constant speed is also allowed.

'Nine C-point locations in a cell' seems a severe restriction. Obviously, the more C-points in the plane, the more degrees of control we gain. However, since our solution is based on search, having too large a branching factor can easily lead to a combinatorial explosion, and thus a smaller number of C-points is desirable. On the other hand, with too small a number of C-points, frequent changes in direction and speed in a short range are not realizable; hence, the search process may fail to find a feasible path. Thus, choosing the C-point configuration is an important decision in this approach. Using the index tree, the space is organized using various sizes of cells. Larger blocks of cells are used to represent areas having a lower density of obstacles while an area with many obstacles is divided further into smaller cells. Therefore, C-points are naturally distributed such that the area of importance, i.e., in the vicinity of obstacles, has a higher density of C-points, while the area far from the obstacles has a relatively sparse distribution of C-points.

The main search procedure is as follows.

1. Push the start point onto the queue.
2. While the queue is not empty, perform the following:
   Remove the lowest cost element from the queue. If it is the goal point, then report the path and exit the procedure. Examine all the neighboring C-points (described below) and put the ones that satisfy the path conditions (described in Section 3.2) into the queue.
3. Report that the procedure has not found the goal (described in Section 3.3).

In the second step of the procedure, we need to inspect all the neighboring candidate C-points. Neighboring C-points are all the C-points in the cells that share an edge with the cell in which the current C-point is found. Our definition of the neighborhood does not use geometric distance. Instead, it uses a cell-wise neighborhood which somehow reflects the "density" of the obstacles. In other words, we define the neighborhood so that in an area where there are few obstacles, it can be large, while in an area of high obstacle density, we have many neighborhood points near the current point. Considering the nature of path planning, this is preferable to using a neighborhood based on geometric distance.

3.2. The Path Conditions

The path conditions to be satisfied at step 2 are defined as follows. Suppose we are at some C-point, say P. The current velocity of the mobile robot and the location of the previous C-point are known. We now choose the acceleration and then the C-point to which we next proceed. Once an acceleration value has been chosen, we have to maintain it until the next C-point. The choice of C-point must satisfy the following path conditions.

(1) The acceleration is not out of range.
(2) The velocity at the next C-point will not be out of range.
(3) The angle made by P satisfies the conditions regarding the centrifugal force and the velocity at P.
(4) The path between P and the next C-point is collision-free.

Checking that conditions (1) and (2) are satisfied is straightforward. We can choose an acceleration within the range, and then compute the velocity of the next C-point from the current values of velocity and acceleration, and the distance between the current and next C-points.

As for condition (3), the following formulation is used for estimating the centrifugal force. We assume that the robot negotiates a curve having a curvature which depends on the angle formed by the two lines meeting at the C-point (e.g., point C in Figure 2). We consider a small distance (constant for all curves) between a C-point and the points where the robot begins to deviate from a trajectory which would have taken it to the C-point. We denote this distance by d (Figure 2). Then the radius r is

$$r = d \times \tan(\frac{\alpha}{2}),$$

where α is the angle made by two line segments that meet at the C-point. We assume d to be sufficiently small in comparison with the distance between the two C-points. Then

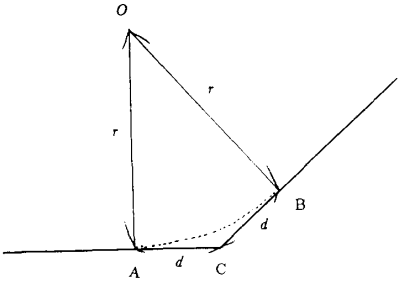

Figure 2.

A path (dashed line) is approximated by two line segments AC and BC.

requirement (3) can be expressed as

$$\frac{mv^2}{r} < \text{Constant},$$

where v is the current velocity and m is the mass of the robot. This means that on each curve we are required to satisfy the inequality

$$\frac{v^2}{\tan(\frac{\alpha}{2})} < C$$

for some constant C.

As to condition (4), cells containing the path segment connecting the current C-point and the next C-point are inspected for intersection points. If there is an intersection, then the next C-point is not qualified as a candidate C-point.

Regarding cost estimation in step 2 of the above search procedure, we can use different criteria depending on which aspects we wish to optimize. Here, we describe an estimation function used in our implementation that optimizes time. We define a function f at the current control point, say CP, by

$$f(CP)=g(CP)+h(CP),$$

where g is the t-value associated with CP, i.e., the time elapsed so far, and h is equal to the distance between CP and the goal point, divided by the maximum velocity of the robot. The function g is the cost of the path so far from the start point, and h represents the heuristic estimate of the cost of the remaining path from CP to the goal. Since h never overestimates the actual time cost from CP to the goal point, this A* heuristic search process having f as its estimate is admissible, i.e., the procedure is guaranteed to compute a time-optimal solution in this search space [Nils80]. In the next section, we will present some results obtained by using this heuristic. As an alternative, it is possible to use estimation functions based on distance traveled or energy consumed.

3.3. Search Failure

One drawback of this method is that it is possible that it does not find a solution in the search space. This can be interpreted as indicating either that a feasible solution doesn't exist at all, or that it doesn't exist in this search space. In the latter case, some possible reasons are that the arrangement of C-points is too coarse, the discretized acceleration values are not appropriate, etc.

One way to remedy this drawback is that when the initial number of C-points is not enough to compute a solution, we can gradually increase the number of C-points in the search space, until a path is finally found or a predetermined resolution limit is reached. Instead of increasing the C-points uniformly over the universe, we can selectively expand the C-points. For example, we can have more C-points in areas which have more obstacles than a given threshold. This scheme can be realized by deepening the tree by one level at each search failure. This has an effect of dividing a cell into eight smaller cells, resulting in more C-points. Since deepening an index tree does not require much work, this method is simple to implement. However, the granular nature of time as well as distance requires us to have a pre-defined resolution limit in our search space.

3.4. Uncertainty

We have earlier assumed that all the movements of the obstacles are precisely known. However, the movements of the obstacles may be availble with some uncertainty in the velocity or the direction. For example, the velocity of an obstacle is known to be between v_1 and v_2. In such a case, we are still able to construct a 3D volume that represents the union of all the possible trajectories. If the difference between v_1 and v_2 are not prohibitively large, we can use the same search procedure to plan a path as before.

4. EXPERIMENTAL RESULTS

In this section we present some experimental results obtained using the technique described in Sections 2 and 3.

Suppose that our testbed is a 512 [m] by 512 [m] world and that the time dimension varies between 0 [sec] and 512 [sec]. Figures 3 illustrates this example for three different velocities of a robot. Let A in the figures be an obstacle moving at 0.5 [m/sec] in an easterly direction. B is a stationary object. The start and goal points are denoted by S and G respectively. In the following cases, acceleration is chosen among the values 1.0, 0.5, 0.0, -0.5 and -1.0 [m/sec^2]. In addition, we require that the velocity at the start and goal points be 0. Figure 3 shows three trajectories and their velocity transition graph.

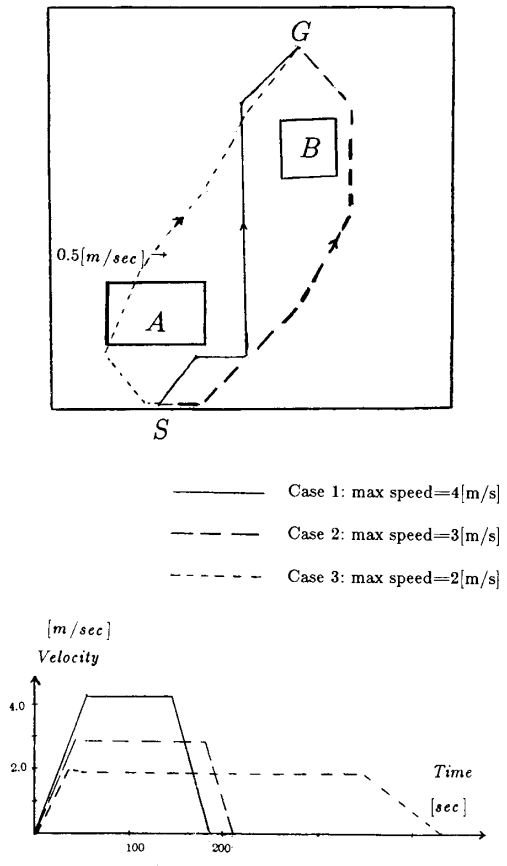

Figure 3
A is a moving object heading in an easterly direction and B is a stationary object.

(Case 1) If the mobile robot is fast enough, it will proceed to the right of A, to the left of B and get to the goal. The maximum speed is 4 [m/s].

(Case 2) If the mobile robot is not fast enough, it will not able to proceed to the left side of B, and it will have to go to the right side of B. The maximum speed is 3 [m/s].

(Case 3) If the mobile robot is much slower, it will let A go by first. The maximum speed is 2 [m/s].

Figure 4 shows a solution dealing with three moving obstacles at the same time. One obstacle O_1 moves in easterly direction at 0.5 [m/sec] as in the previous example. There are two more triangular obstacles, O_2 and O_3, whose velocities are 1.4 [m/sec] and 1.0 [m/sec], respectively. S and I represent the start and goal points. In this example, the maximum speed of the robot is 4.5 [m/sec]. Note that the robot starts decelerating at D to avoid a collision that would have occurred if it had proceeded at the same speed. This has an effect of letting obstacle O_2 go by first. The dashed lines show the position of obstacle O_2 at the time F. This technique of avoiding obstacles characterizes path planning among moving obstacles and can only be realized by taking the velocity and acceleration of the robot into consideration.

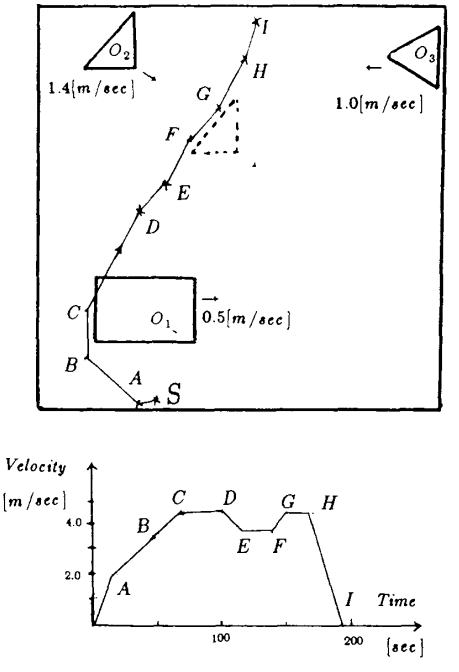

Figure 4

A robot with three moving obstacles.
The robot starts decelerating at D to avoid a collision.
Dashed lines show an obstacle at time F

5. CONCLUDING REMARKS

An approach has been proposed to solve the path planning problem for a mobile robot within an environment that contains moving obstacles. By adding time as an additional dimension to the world, a simple formulation was obtained. We have discussed the use of a spatial index which makes good use of this formulation. The spatial index is based on a cell decomposition scheme in which each cell has a simple geometry, i.e., it contains at most one vertex or one edge of an obstacle.

In this paper, we restricted our attention to the three most fundamental factors in navigation, that is, velocity, acceleration, and centrifugal force. These factors are essential in any path planning application for a vehicle that moves on land, on sea, or in air. Also, these factors form the basis for further considerations, such as optimization of the path with respect to energy consumption, etc. To model these factors, we introduced conditions which are imposed on a path in the search procedure. Our experimental results showed that a reasonable path is obtained using this formulation.

An important goal in path planning is to avoid being concerned with details that don't affect the choice of the path. In this aspect, hierarchical structures are promising in two-dimensional path planning [Kamb86]. Since the search space in a time-varying environment tends to become greater than that for stationary path planning, this comment is even more applicable to our problem. For this reason, we used a spatial decomposition with respect to the time dimension as well. In such a case, a large block means that it is located in an area where there is not much motion within some time period or within some distance. Hence the planner is not affected by the motions of distant obstacles, thus facilitating the planning procedure. If we simply stack two-dimensional planes as in [Erdm86], the path planner will miss this computational aspect since it has to consider every motion of the obstacles in the world, even though some of them are relatively remote from the robot and would not have affected the path planning operation.

However, large empty blocks mean large separations between C-points which could result in a zig-zagging path. In such a case, some smoothing mechanism may be necessary.

Our approach indicates that we can incorporate other time-varying factors into the path planning process. As a matter of fact, navigation is also affected by various road or field conditions. Although the method presented in this paper will itself be useful in some applications such as an unmanned carrier in a factory, more advanced and intelligent robot planning will become possible if it is combined with the ability to understand motion of moving objects in the outside world, and to utilize knowledge as obtained through spatial information systems.

ACKNOWLEDGMENTS

We would like to thank Azriel Rosenfeld for his comments.

REFERENCES

[Ayal85] - D. Ayala, P. Brunet, and I. Navazo, Object Representation by Means of Nonminimal Division Quadtrees and Octrees, *ACM Transactions on Graphics 4,* 1(January 1985), 41-59.

[Boll85] - R. C. Bolles and H. H. Baker, Epipolar-Plane Image Analysis: A Technique for Analyzing Motion Sequences, *Proceedings of the Third Workshop on Computer Vision: Representation and Control,* Bellaire, Michigan (October 1985), 168-178.

[Broo83] - R. A. Brooks, Solving the Find-Path Problem by Good Representation of Free Space, *IEEE Transactions on Systems, Man, Cybernetics 13,* 3(March/April 1983), 190-197.

[Broo86] - R. A. Brooks, A Robust Layered Control System for a Mobile Robot, *IEEE Journal of Robotics and Automation 2,* 1(March 1986), 14-23.

[Carl85] - I. Carlbom, I. Chakravarty, and D. Vanderschel, A Hierarchical Data Structure for Representing the Spatial Decomposition of 3-D Objects, *IEEE Computer Graphics and Applications 5,* 4(April 1985), 24-31.

[Crow85] - J. L. Crowley, Navigation for an Intelligent Mobile Robot, *IEEE Journal of Robotics and Automation 1,* 1(March 1985), 31-41.

[Davi85] - L. S. Davis, F. Andresen, R. Eastman, and S. Kambhampati, Visual Algorithms for Autonomous Navigation, *Proceedings of IEEE International Conference on Robotics and Automation,* St. Louis, Missouri (March 1985), 856-861.

[Erdm86] - M. Erdmann and T. Lozano-Perez, On Multiple Moving Objects, *Proceedings of IEEE International Conference on Robotics and Automation,* San Francisco, California (April 1986), 1419-1424.

[Este83] - D. M. Esterling and J. Van Rosendale, An Intersection Algorithm for Moving Parts, *Proceedings of NASA Symposium on Computer Aided Geometry Modeling,* Hampton, Virginia (April 1983), 119-123.

[Fuji85] - K. Fujimura and T. L. Kunii, A Hierarchical Space Indexing Method, *Proceedings of Computer Graphics '85,* Tokyo, (April 1985), T1-4, 1-14.

[Kamb86] - S. Kambhampati and L. S. Davis, Multiresolution Path Planning for Mobile Robots, *IEEE Journal of Robotics and Automation 2,* 3(September 1986), 135-145.

[Kant86] - K. Kant and S. W. Zucker, Toward Efficient Trajectory Planning: The Path-Velocity Decomposition, *The International Journal of Robotics Research 5,* 3(Fall 1986), 72-89.

[Loza79] - T. Lozano-Perez and M. A. Wesley, An Algorithm for Planning Collision Free Paths among Polyhedral Obstacles, *Communications of the ACM 22,* 10(October 1979), 560-570.

[Meag82] - D. Meagher, Geometric Modeling using Octree Encoding, *Computer Graphics and Image Processing 19,* 2(June 1982), 129-147.

[Mora83] - H. P. Moravec, The Stanford Cart and the CMU Rover, *Proceedings of IEEE 71,* 7(July 1983), 872-884.

[Nils80] - N. J. Nilsson, *Principles of Artificial Intelligence,* Chapter 2, Tioga, Palo Alto, California, 1980.

[Reif85] - J. Reif and M. Sharir, Motion Planning in the Presence of Moving Obstacles, *Proceedings of Symposium on Foundation of Computer Science,* Portland, Oregon (October 1985), 144-154.

[Requ82] - A. A. G. Requicha and H. B. Voelcker, Solid Modeling: a Historical Summary and Contemporary Assessment, *IEEE Computer Graphics and Applications 2,* 6(June 1982), 9-24.

[Same85a] - H. Samet and M. Tamminen, Bintrees, CSG Trees, and Time, *Computer Graphics 20* (July 1985), 121-130 (also *Proceedings of the Siggraph '85 Conference,* San Francisco, California, July 1985).

[Same85b] - H. Samet and R. E. Webber, Storing a Collection of Polygons using Quadtrees, *ACM Transactions on Graphics 2,* 3(July 1985), 182-222.