A Hierarchical Strategy for Path Planning Among Moving Obstacles

KIKUO FUJIMURA, STUDENT MEMBER, IEEE, AND HANAN SAMET, SENIOR MEMBER, IEEE

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 investigate an appropriate model to represent the extended world for the path planning task, and give a time-optimal solution using this model.**

I. Introduction

Research on and development of intelligent mobile robots have attracted much interest recently [3], [13], [14], [16]. To realize such an autonomous mobile robot, it is necessary to synthesize many techniques including some elements from artificial intelligence. Principal requirements for a robot include:

1) obtaining information from the outside world using visual or auditory sensors;
2) matching the information with the internal database to understand the environment;
3) designing an appropriate plan to execute a given task;
4) handling unexpected events, arriving either from the outside world or from the robot itself;
5) learning from experience to improve its performance.

Planning a collision-free path is one of the fundamental requirements for a mobile robot to execute its tasks. Much of the prior work on this topic is concerned with methods for generating a path among stationary obstacles [3], [13], [14], [16]. Realistically, however, obstacles are not always stationary. It should be clear that a robot that can deal with moving obstacles will be capable of performing a much larger and more complex class of tasks. Motivated in this manner, we consider the situation where the obstacles are in motion. As an example of path planning involving moving obstacles, consider an airport, where an unmanned cart moves among taxiing airplanes, carrying their cargoes from one place to another. Here, we can assume that the motion of each airplane (i.e., obstacle) is known to the cart before it moves. The cart is usually required to navigate in an environment which contains several moving obstacles. Considering that the motion of most of the obstacles is known beforehand and that a cart is not required to be at work all the time, it will be acceptable that it takes a while to plan a navigation path. On the other hand, in relatively short range path planning, as is required in a vehicle navigating an unexplored area [18], the situation can be imminent. While a robot is exploring an area, its sensor may suddenly perceive a moving object crossing its path. In such a case, it should be able to generate and execute a movement to safely avoid the object, e.g., by decelerating to let the object pass by, or by accelerating to dodge the object. In this case, although the size of the problem in terms of distance, time, and the number of obstacles involved may be smaller than those of the previous example, prompt judgement is a key factor in successfully handling the situation.

The path planning problem is also crucial in a situation where several robots operate together in a workspace. Issuing a feasible navigation command for each robot in motion without losing much efficiency is a nontrivial problem. With the capability of handling moving objects, we will be able to implement concurrent motion for multiple robots. For example, when planning a task for the second robot, we can regard the first robot as one of the moving obstacles. This capability can be instrumental in raising the productivity in an automated factory.

As can be seen from these examples, the ability to deal with moving obstacles will significantly increase the potential capabilities and the range of applications of mobile robots.

From a research point of view, the presence of moving obstacles gives rise to new aspects of the path planning problem. For example, when moving obstacles are involved, the shortest distance path is not always the minimum time path. Therefore, more work is required to generate the optimal path with respect to both the elapsed time and the distance traveled. In addition, if our goal is to find an economical path, that is, a path that requires the consumption of a minimum amount of energy, then we must also pay attention to the speed and acceleration of the mobile robot.

Thus the problem of path planning among moving objects is in many ways different from, and usually more complex than, that of path planning with stationary obstacles. In this paper, we will discuss the fundamental issues relating to the avoidance of moving obstacles, and present a basic solution to

Manuscript received February 3, 1987; revised April 26, 1988. This work was supported by the National Science Foundation under Grant DCR-86-05557. Part of the material in this paper was presented at the IEEE International Conference on Robotics and Automation, Philadelphia, PA, April 1988.

K. Fujimura is with the Center for Automation Research, University of Maryland, College Park, MD 20742.

H. Samet is with the Center for Automation Research, the Computer Science Department and Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742.

IEEE Log Number 8823372.

the problem. Section II defines the problem, surveys related work, and describes our approach to solving the problem. Section III discusses the representation of the obstacles. The search strategy is described in Section IV. Section V contains experimental results obtained by using the method described here. Section VI contains some brief concluding remarks.

II. 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. In this paper, we will concentrate on the planning aspect of the problem and will not discuss the important issue of how a robot perceives moving objects, i.e., we assume that all the information regarding obstacles (such as shapes, movement directions, etc.) is somehow provided *a priori* to the robot. Although our approach is conceptually applicable to the collision avoidance problem in three-dimensional space, for the purpose of simplicity we will present our algorithm in the context of obstacles in two dimensions. One of our principal requirements is that the mobile object not collide with any of the obstacles. There are other factors that we have to take into consideration in the process of path generation. These factors include physical constraints on the mobile object. For example, the object cannot accelerate beyond some upper limit. In more realistic applications, knowledge of the area in which the mobile object navigates (e.g., slopes and ground conditions) may also affect the choice of the path.

In this paper, we are interested in paths that satisfy three fundamental factors in navigation: velocity, acceleration, and centrifugal force on a curve. In other words, 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. Here we only discuss the case where the mobile robot is considered to be a point. The general problem in which a robot is a polygon can be transformed into a simpler problem in which a robot is a point by expanding obstacles by the size of the robot while shrinking the robot to a point [16]. This simpler problem is equivalent to the original problem only when rotation of the robot is not allowed. In addition, we assume that each obstacle is a polygon which moves at a constant speed without rotation.

A. Related Work

Prior approaches at dealing with stationary objects are not always compatible with the problem of moving obstacles. For example, since visibility is constantly changing as obstacles move, the visibility graph method [16] does not apply to the situation involving moving obstacles. Various other methods divide space into some type of smaller spaces [3], [14]. These approaches are also based on the assumption that the geometry is invariant during the path search process. Therefore, applying these methods to our problem is generally not straightforward. Another characteristic of robot planning among moving obstacles is that the curvature of the path plays an important role in the choice of the path. As we discuss in the next section, because of this aspect, some approaches using free spaces [3] are not appropriate for our purpose.

The following papers address the collision detection problem among moving objects. Canny [5] formulates the motion of polyhedra in three dimensions using a quaternion representation. Samet and Tamminen [25] 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 Rosendale [11]. They divide the time dimension as well as the other dimensions recursively to quickly locate the collision point between two moving objects. Regarding collision avoidance, Reif and Sharir [22] show that motion planning for a three-dimensional environment containing moving obstacles is PSPACE-hard given bounds on the robot's velocity, and NP-hard without such bounds. Canny and Reif [6] show that motion planning for a point in the plane with a bounded velocity is NP-hard, even when the moving obstacles are convex polygons moving at constant linear velocity without rotation. These results indicate that the problem of motion planning in a time-varying environment is essentially much harder than the problem with stationary obstacles. Nevertheless, there are some approaches at solving the problem of collision avoidance among moving objects. These approaches are successful in a limited domain. Kant and Zucker [15] decompose the problem into two parts. In the first part, they ignore the moving obstacles in planning a path among the stationary obstacles. In the second part, a graph is used to define 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 velocity. Erdmann and Lozano-Perez [10] describe a planner for moving objects that constructs a configuration space each time some object in the scene changes its velocity. Their method is based on stacking two-dimensional planes, where each plane represents a configuration space at some time. Then, two adjacent planes are inspected to see if a path exists between these two planes. A path consists of a sequence of vertex-to-vertex transitions between two adjacent planes. O'Dunlaing [20] considers the problem of planning motion for a point-robot in one dimension. Given start velocity/location and goal velocity/location, the point-robot is to avoid collision with two obstacles moving at both ends without exceeding predetermined acceleration bound. Bolles and Baker [2] construct a three-dimensional solid from an image sequence for motion analysis.

B. Discretizing Space

As indicated in some studies [6], [22] on the complexity of motion planning problems in a time-varying environment, it is a nontrivial problem to find a path which satisfies all the requirements as to acceleration and centrifugal force while still avoiding the moving obstacles. We address this problem by discretizing the search space. This discretization has two aspects. First, we require that a path be represented as a sequence of specific points in space–time. Second, the acceleration of the robot takes on discretized value. In other words, a trajectory of the robot in space–time is represented as a parabolic curve of specific parameters. This is discussed further in Section IV-A.

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, z) is inside that volume in space–time, then the two-dimensional point (x, y) was occupied by object O at time z.

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 (space–time) path for the mobile object that connects the start position to the goal position. Minimizing the z value of the goal position will be a main concern in the problem of time-optimal path generation. In other cases, the z value is irrelevant as long as it leads the object to the final position without collision. Since our third dimension is time, the search should be carefully designed so as not to choose an unrealistic path—e.g., traveling in the negative time direction. Moreover, the path must satisfy the predefined conditions regarding velocity, acceleration, and curvature.

Note that paths and obstacles have to be specially represented to incorporate the space–time approach. In Section III, we discuss the type of representation that will fit naturally into our space–time formulation.

III. Space Representation

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 $x_1 < x < x_2, y_1 < y < y_2$, and $t_1 < t < t_2$. x and y are measured in terms of distance while t corresponds to time. Usually, it is convenient to let $x_1 = y_1 = t_1 = 0$ and $x_2 = y_2$. Note that time is also bounded. In this world, every motion of an object on the two-dimensional plane during the time period between t_1 and t_2 is represented as a three-dimensional object. Every point on the path must also be found inside this world.

The primary objectives of this section are to investigate methods of representing an object in three dimensions and then to determine the one that suits our purpose in path planning. The following are requirements that the obstacle representation must satisfy for our purposes.

1) It must be easy to detect interference between the obstacles and a path segment.

2) It must be computationally cheap to add and delete obstacles. This is an important aspect when a path is generated in a real-time environment. For example, an obstacle may change its course, which means that the mobile robot must update its obstacle representation.

Geometric models that are often used to represent three-dimensional shapes are Constructive Solid Geometry (CSG) and the boundary model [21]. Although the CSG representation is compact, it is not local, i.e., to identify what is in a given location requires some computation. The boundary model is useful for representing topologically connected information; however, it still requires some additional work to get geometrical neighborhoods.

We have adopted a quadtree-type hierarchical representation. In planning the motion of a robot, we may need to generate a path that avoids obstacles with some safety margin. In such a case, it is required to search the neighborhood to see if any obstacle exists within some distance. Here a hierarchical structure has an advantage over other representations, since using a tree structure as an index to the model world, efficient access to a location is possible. Thus requirement 1) is satisfied by the tree structure. Some approaches use a tree structure that is based on an irregular decomposition of the space [17]. Considering requirement 2), efficient updating of a tree is a desirable goal. Trees based on either a regular or an irregular decomposition require some work to modify the configuration when entities are added or deleted. Updates of a tree based on a regular decomposition (e.g., an octree) tend to be local; i.e., the part of the tree which does not involve the change remains undisturbed. In this paper, we use a hierarchical structure which is based on a regular decomposition.

Recall that we have assumed that the motion of the obstacles does not 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. When a polygon rotates, the decomposition rules described below are still valid; however, the trajectory it makes is no longer a simple polyhedron. For this reason, we restrict the motions of obstacles to be simple translations.

We represent the time dimension by the third dimension. Thus the time dimension is also subdivided. Methods to store polygonal and polyhedral shapes using quadtree-type decompositions have been studied in the context of computer cartography [26] and computer-aided design [1], [7]. Our representation 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 does not contain any part of the trajectory of a vertex, but contains part of the trajectory of an edge of an obstacle.

c) A cell does not 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. Note that this terminology is not compatible with that used in [1], [7]. Here, we use terms based on a two-dimensional rather than a three-dimensional viewpoint. For example, a cell created by criterion b) is called a surface cell in the papers cited above. However, we call it an edge cell to make it clear that a two-dimensional edge has moved in the cell.

Unlike conventional octrees [27] in which objects are approximated, the tree structure defined above stores polygonal shapes exactly and requires less memory space [1]. Fig. 1 illustrates the concept described above. Suppose that an object moves in the x direction (Fig. 1(a)). A solid line and a dashed line depict the initial and final position, respectively, of the

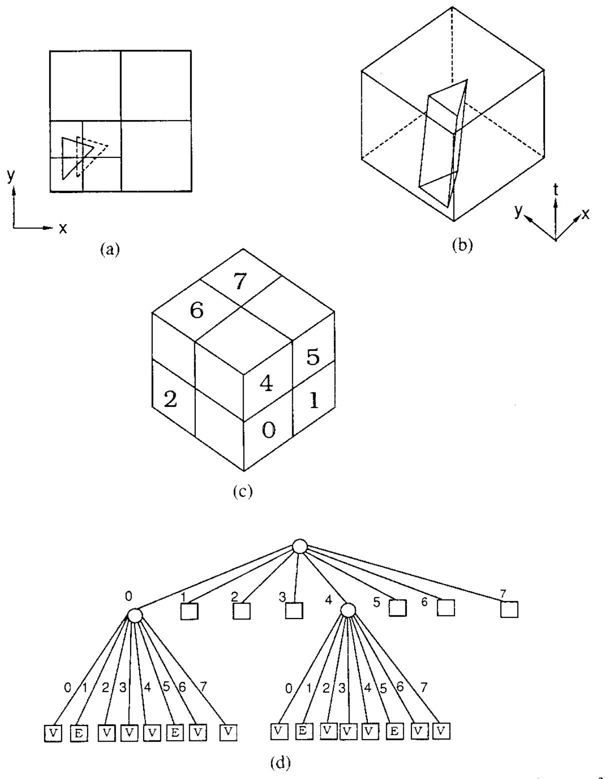


Fig. 1. (a) An object moving in the x direction. (b) The time-space image of the object in (a). (c) The cell numbering convention in the octree structure. (d) A tree representation of the space (b). V and E represent a vertex cell and an edge cell, respectively.

object. Fig. 1(b) shows the volume swept by the object in space-time. A tree representation of this three-dimensional image is shown in Fig. 1(d), where the cell numbering convention of Fig. 1(c) is used.

Next, an octree is built corresponding to the volume swept by the motion of objects along the given trajectories. This conversion is performed in the following way. Initially, the entire universe is treated as a single cell which is represented by an octree 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 more details, see [1], [12].

IV. Path Searching

This section describes how the search procedure generates a collision-free path using the representation introduced in the previous section.

A. Control Points

In much of the prior work, the path is represented by one of the following techniques. One way to specify a path is to use a sequence of points. The object follows these points as it proceeds. Approaches making use of a visibility graph [16] and medial axis transforms [23] fall in this category.

Another way which has frequently been used in recent work is to represent a path by a sequence of empty spaces. Here, the free space is divided into possibly overlapping pieces of empty space, and the path is constructed by connecting these empty spaces. The use of generalized cylinders [3] and quadtrees [14] are examples of this approach. Applying this idea in three-dimensional space-time gives rise to one problem in our situation—i.e., how the path is determined inside the free space. For example, when using cylinders we see from Fig. 2 that two possible paths (a) and (b) can be chosen in the same cylinder. However, considering the length and curvature of the path, the difference between paths (a) and (b) is not negligible. In other words, factors such as velocity, acceleration, and curvature are highly susceptible to a slight change in the trajectory.

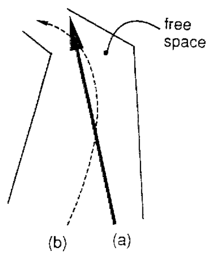


Fig. 2. Two different paths (a) and (b) are possible within one free space.

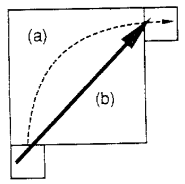


Fig. 3. The quadrant representation of free space can contain paths (a) and (b) with different curvatures.

The same comment also applies to the quadtree approach. Quadrants are not sufficiently specific to nail down a particular path. In Fig. 3, path (a) requires more time than path (b), which means that the next quadrant chosen by path (a) should be different from that for path (b). This difference in time must be explicitly expressed in the representation.

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 an L-point and a T-point. An L-point represents two-dimensional location (i.e., (x, y)) of the C-point, and a T-point represents the time at which the mobile object passes that L-point. The x and y values of an L-point take on discrete values. Let a square denote the projection of a three-dimensional space-time cell onto the x–y plane. We define the arrangement of these L-points such that the x and y coordinates of L-points lie only at the following nine locations in a square: 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); see Fig. 4. The value of a T-point 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. Next, it determines the appropriate velocity. This, in turn, determines the T-point. Since we have a choice as to the velocity (or acceleration) value, the T-point varies depending on velocity values that

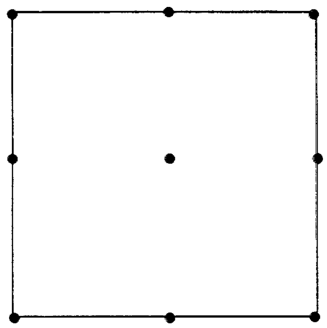


Fig. 4. A cell has nine L-points.

have been chosen. Thus two identical sequences of L-points with different sets of T-points represent two different motions. In other words, L-points (two-dimensional components of C-points) are fixed when the obstacles are encoded in an octree, while the T-point components of these C-points are not fixed. T-points are only determined as a consequence of the T-point component of the previous C-point on the path and of the acceleration chosen at the L-point component of the previous C-point.

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 L-points, while it retains the same acceleration between two L-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. As a result, the velocity of the mobile object, which is a function of acceleration, changes continuously throughout the entire path. (In our implementation, gradual incrementing of the acceleration is not required, which means that there is no restriction on the choice of the acceleration value at an L-point as long as it stays between the predetermined upper limit and lower limit.) In fact, a mobile object will not pass directly through the L-point. Instead, it follows a curved path that does not pass through the L-point in order to avoid a sudden change of movement direction. However, we will assume that the distance traveled can be approximated by the sum of the lengths of the two tangents to the curve that meet at the L-point. For example, in Fig. 5, we approximate a path length depicted by a dashed line by the two line segments AC and BC.

Obviously, the greater the number of different L-points in the plane, the more degrees of control we gain. At first glance, a simple grid-type regular configuration of L-points may seem sufficient. However, observing that having too large a branching factor can easily lead to a combinatorial explosion, a smaller number of L-points is desirable. On the other hand, with too small a number of L-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 the L-point configuration is an important problem

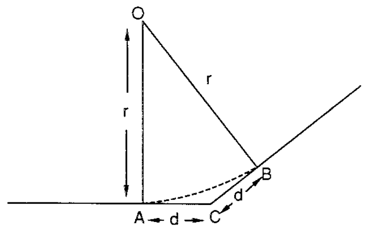


Fig. 5. A path (dashed line) is approximated by two line segments AC and BC.

in this approach. The L-point configuration is adaptable so that the area of importance (i.e., in the vicinity of obstacles) has a higher density of points, while the area far from the obstacles has a relatively sparse distribution of C-points. To embody this principle, a hierarchical representation is appropriate as it means that 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 subdivided further into smaller cells.

The main search procedure is as follows. We use a priority queue of C-points where the T-point component of a C-point serves as the point's priority.

1) Push the start point onto the queue.

2) While the queue is not empty, recursively 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. Generate all the neighboring L-points (described below) of the L-point component of the removed element, and select an acceleration value. Determine T-points corresponding to the generated L-points. Put the C-points that satisfy the path conditions (described in Section IV-B) into the queue.

3) Report that the procedure has not found the goal (described in Section IV-C).

In the second step of the procedure, we need to inspect all the neighboring candidate L-points. Neighboring L-points are all the L-points in the cells that share an edge with the cell in which the current L-point is found; see Fig. 6. A neighboring L-point can be quite distant, like the one in cell D of Fig. 6(c), so that we can move rapidly over an area where there are few obstacles.

B. The Path Conditions

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

1) The acceleration is not out of range.
2) The speed at the next L-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 L-point is collision-free.

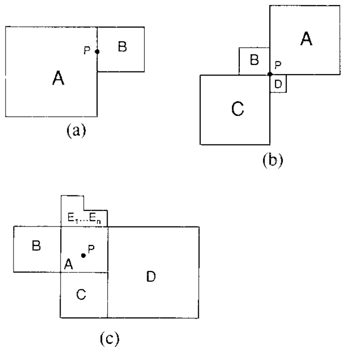


Fig. 6. Definition of neighborhood points. (a) L-point P is on an edge. Its neighboring points are the L-points in the cells A and B. (b) L-point P is at a corner. Its neighboring points are the L-points in the cells A, B, C, and D. (c) L-point P is in the center of a cell. Its neighboring points are the L-points in the cells A, B, C, D, E_1, E_2, $\cdots$, and E_n.

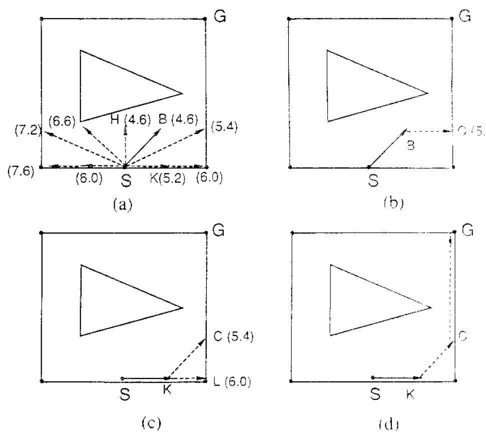


Fig. 7. An example of the process of computing a path.

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 L-point from the current values of velocity and acceleration, and the distance between the current and next L-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 L-point (C in Fig. 5). We also assume that the distance between an L-point and the points where the robot begins to deviate from a trajectory which would have taken it to the L-point is a small constant for all curves. We denote this distance by d (Fig. 5). Then the radius r is

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

where α is the angle made by two line segments that meet at the L-point. As mentioned earlier, we assume d to be sufficiently small in comparison with the distance between the two L-points. Then requirement 3) can be expressed as

$$\frac{mv^2}{r} < \text{constant}$$

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

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

for some constant C.

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

Regarding cost estimation in step 2) of the 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-point component of CP, i.e, the time elapsed so far, and h is equal to the distance between the L-point component of CP and the goal point, divided by the maximum speed 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 [19] having f as its estimate is admissible, i.e., the procedure is guaranteed to compute a time-optimal solution in this search space. 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 consumed energy.

Fig. 7 is an example of path computation using an A^* algorithm. Let S and G denote the start and goal points. Assume that the path conditions do not allow the robot to make a turn of degree less than 90. Dotted lines are drawn from S to the L-points that are reachable from the current L-point. The numbers in the parentheses correspond to the values of the function f at those L-points. The L-point with the minimum f value is selected as the next L-point to which the robot moves and this path is shown with a thick line in Fig. 7. The following is a more detailed explanation. Starting at S, there are nine candidate L-points from which to choose. They are all placed in the queue. f is at a minimum at L-points H and B (see Fig. 7(a)). We reject H since none of the L-points that are reachable from H satisfy the path conditions. Thus we are left with B. Continuing our search from B, we find that C is the only L-point reachable from B (see Fig. 7(b)). However, at this point we find that the minimum value of f is at K (i.e., $f(K)$ is smaller than $f(C)$) and hence K is the next node to be processed in the search process. L-points C and L are reachable from K while also satisfying the path conditions (see Fig. 7(c)). At this point, the goal point G is reachable from C (see Fig. 7(d)) and the path segment CG satisfies the path conditions. In this example, this is the optimal path. Of course, in the more general situation, the search process continues to check if a more minimal solution exists.

C. Search Failure

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

One way to remedy this drawback is to note that when the initial number of L-points is not enough to compute a solution, we can gradually increase the number of L-points in the search space, until a path is finally found or a predetermined resolution limit is reached. Instead of increasing the L-points uniformly over the universe, we can selectively expand the L-points. For example, we can have more L-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 L-points. Since deepening a tree does not require much work in an octree structure, this method is simple to implement. However, the granular nature of time as well as distance requires us to have a predefined resolution limit in our search space.

V. Experimental Results

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

Suppose that our testbed is a 512 (m) by 512 (m) world and that the time dimension varies between 0 (s) and 512 (s). Fig. 8 illustrates this example for three different speeds of a robot. Let A in the figures be an obstacle moving at 0.5 (m/s) 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/s^2). In addition, we require that the speed at the start and goal points be 0.

(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. Fig. 8(a) shows the trajectory and its speed transition graph. The maximum allowable speed in this case is 4 (m/s).

(Case 2) If the mobile robot is not fast enough, it will not be able to proceed to the left side of B, and will have to go to the right side of B. Fig. 8(b) shows the case where the maximum speed of the robot is 3 (m/s).

(Case 3) If the mobile robot is much slower, it will let A go by first. Fig. 8(c) shows the case where the maximum speed of the robot is 2 (m/s).

Fig. 9 shows a solution dealing with three moving obstacles at the same time. One obstacle, O_1, moves in easterly direction at 0.5 (m/s) as in the previous example. There are two more triangular obstacles, O_2 and O_3, whose speeds are 1.4 (m/s) and 1.0 (m/s), respectively. S and I represent the start and goal points. In this example, the maximum speed of the robot is 4.5 (m/s). 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

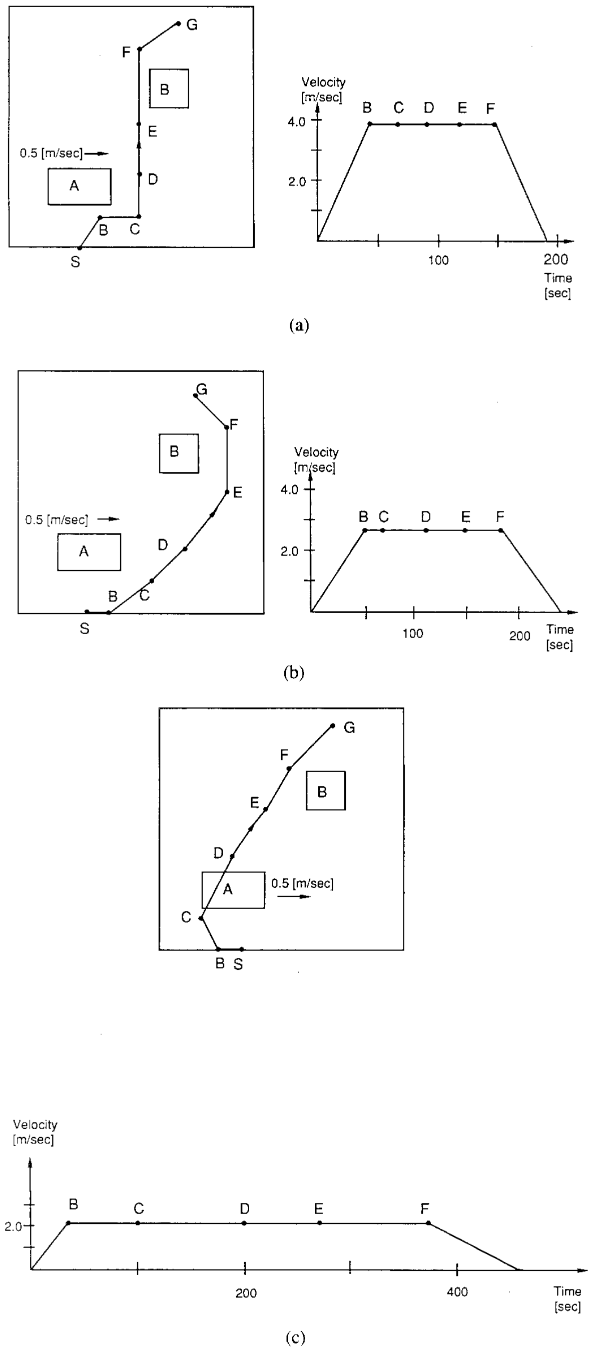
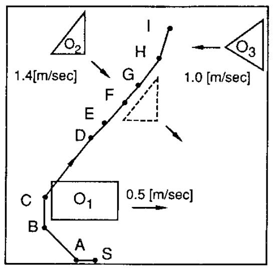
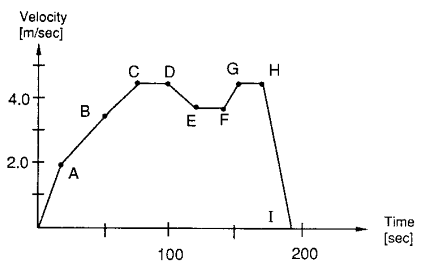


Fig. 8. One moving object heading in an easterly direction and one stationary object. (a) The maximum allowable speed of the robot is 4 (m/s). (b) The maximum allowable speed of the robot is 3 (m/s). The course the robot takes is different from that in Fig. 9. (c) The maximum allowable speed of the robot is 2 (m/s).



(a)



(b)

Fig. 9. A robot with three moving obstacles. The robot starts decelerating at D to avoid a collision. Broken lines show an obstacle at time F.

taking the speed and acceleration of the robot into consideration.

VI. 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 representation methods which would make good use of this formation and have adopted a quadtree-type hierarchical structure to represent the obstacles. The representation 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, speed, 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 introduce conditions which are imposed on a path in the search procedure. We showed that a time-optimal path is easily obtained using this formulation.

An important goal in path planning is to avoid being concerned with details that do not affect the choice of the path. In this aspect, hierarchical structures are promising in two-dimensional path planning [14]. 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 hierarchical 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 [10], then 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.

Nevertheless, our approach still suffers from the large search space size; the possible locations of L-points tend to grow rapidly in size, as the number of obstacles in space increases. For example, suppose that the world consists of $2^l \times 2^l \times 2^l$ octants of the same size. Then, the number of L-point locations can be as large as $O(2^{2l})$. At each L-point, the next location and the next acceleration are selected using the path conditions. As this choice is made at each L-point in the worst case, the worst case run time of our algorithm is exponential in the number of the total number of nodes in an input octree. Note that the closer two objects are in the world, the deeper the level of subdivision becomes, resulting in more nodes in the tree. For this reason, we observe that our algorithm works better in an uncluttered environment.

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 geographic information systems.

Acknowledgment

The authors would like to thank A. Rosenfeld for his comments.

References

[1] D. Ayala, P. Brunet, and I. Navazo, "Object representation by means of nonminimal division quadtrees and octrees," *ACM Trans. Graphics*, vol. 4, no. 1, pp. 41-59, Jan. 1985.
[2] R. C. Bolles and H. H. Baker, "Epipolar-plane image analysis: A technique for analyzing motion sequences," in *Proc. 3rd Workshop on Computer Vision: Representation and Control* (Bellaire, MI, Oct. 1985), pp. 168-178.
[3] R. A. Brooks, "Solving the find-path problem by good representation of free space," *IEEE Trans. Systems, Man, Cybern.*, vol. SMC-13, pp. 190-197, Mar./Apr. 1983.
[4] R. A. Brooks, "A robust layered control system for a mobile robot," *IEEE J. Robotics Automat.*, vol. RA-2, no. 1, pp. 14-23, Mar. 1986.
[5] J. Canny, "Collision detection for moving polyhedra," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-8, pp. 200-209, Mar. 1986.
[6] J. Canny and J. Reif, "New lower bound techniques for robot motion planning problems," in *Proc. 27th IEEE Symp. on Foundations of Computer Science* (Los Angeles, CA, Oct. 1987), pp. 49-60.
[7] I. Carlbom, I. Chakravarty, and D. Vanderschel, "A hierarchical data structure for representing the spatial decomposition of 3-D objects," *IEEE Comput. Graph. Appl.*, vol. 5, no. 4, pp. 24-31, Apr. 1985.
[8] J. L. Crowley, "Navigation for an intelligent mobile robot," *IEEE J. Robotics Automat.*, vol. RA-1, no. 1, pp. 31-41, Mar. 1985.
[9] L. S. Davis, F. Andersen, R. Eastman, and S. Kambhampati, "Visual algorithms for autonomous navigation," in *Proc. IEEE Int. Conf. on Robotics and Automation* (St. Louis, MO, Mar. 1985), pp. 856-861.

[10] M. Erdmann and T. Lozano-Perez, "On multiple moving objects," *Algorithmica,* vol. 2, no. 4, pp. 477–522, 1987.

[11] D. M. Esterling and J. Van Rosendale, "An intersection algorithm for moving parts," in *Proc. NASA Symp. on Computer Aided Geometry Modeling* (Hampton, VA, Apr. 1983), pp. 119–123.

[12] K. Fujimura and T. L. Kunii, "A hierarchical space indexing method," in *Computer Graphics: Visual Technology and Art,* T. L. Kunii, Ed. New York, NY: Springer-Verlag, pp. 21–31.

[13] M. Herman, "Fast, three-dimensional, collision-free motion planning," in *Proc. IEEE Int. Conf. on Robotics and Automation* (San Francisco, CA, Apr. 1986), pp. 1056–1063.

[14] S. Kambhampati and L. S. Davis, "Multiresolution path planning for mobile robots," *IEEE J. Robotics Automat.,* vol. RA-2, no. 3, pp. 135–145, Sept. 1986.

[15] K. Kant and S. W. Zucker, "Toward efficient planning: The path-velocity decomposition," *Int. J. Robotics Res.,* vol. no. 5, pp. 72–89, Fall 1986.

[16] T. Lozano-Perez and M. A. Wesley, "An algorithm for planning collision free paths among polyhedral obstacles," *Commun. ACM,* vol. 22, no. 10, pp. 560–570, Oct. 1979.

[17] T. Lozano-Perez, "Automatic planning of manipulator transfer movements," *IEEE Trans. Syst., Man. Cybern.,* vol. SMC-11, no. 10, pp. 681–698, Oct. 1981.

[18] H. P. Moravec, "The Stanford cart and the CMU rover," *Proc. IEEE,* vol. 71, no. 7, pp. 872–884, July 1983.

[19] N. J. Nilsson, *Principles of Artificial Intelligence.* Palo Alto, CA: Tioga Press, 1980, ch. 2.

[20] C. O'Dunlaing, "Motion planning with inertial constraints," *Algorithmica,* vol. 2, no. 4, pp. 431–476, 1987.

[21] A. A. G. Requicha and H. B. Voelcker, "Solid modeling: A historical summary and contemporary assessment," *IEEE Comput. Graph. Appl.,* vol. 2, no. 6, pp. 9–24, June 1982.

[22] J. Reif and M. Sharir, "Motion planning in the presence of moving obstacles," in *Proc. 25th IEEE Symp. on Foundations of Computer Science* (Portland, OR, Oct. 1985), pp. 144–154.

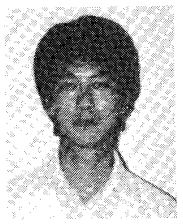
[23] R. Ruff and N. Ahuja, "Path planning in a three-dimensional environment," in *Proc. 7th Int. Conf. on Pattern Recognition* (Montreal, Que,. Canada, July 1984), pp. 188–191.

[24] H. Samet, "Neighbor finding techniques for images represented by quadtrees," *Comput. Graph. Image Processing,* vol. no. 18, no. 1, pp. 37–57, Jan. 1982.

[25] H. Samet and M. Tamminen, "Bintrees, CSG trees, and time," *Computer Graphics,* vol. 20, pp. 121–130, July 1985. Also in *Proc. Siggraph '85 Conf.* (San Francisco, CA, July 1985).

[26] H. Samet and R. E. Webber, "Storing a collection of polygons using quadtrees," *ACM Trans. Graphics,* vol. 2, no. 3, pp. 182–222, July 1985.

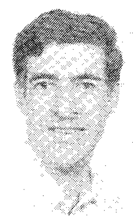
[27] M. Yau and S. N. Srihari, "A hierarchical data structure for multidimensional digital images," *Commun. ACM,* vol. 26, no. 7, pp. 504–515, July 1983.



Kikuo Fujimura (S'88) received the B.S. and M.S. degrees in information science from the University of Tokyo, Tokyo, Japan, in 1983 and 1985, respectively. He is now working toward the Ph.D. degree at the University of Maryland, College Park.

He has been a Research Assistant at the Center for Automation Research, University of Maryland, since August 1985. His current research interests include robotics, computational geometry, computer vision, and fault-tolerant systems.

Mr. Fujimura is a student member of the IEEE Computer Society and ACM.



Hanan Samet (S'70–M'75–SM'84) received the B.S. degree in engineering from the University of California, Los Angeles, and the M.S. degree in operations research, as well as the M.S. and Ph.D. degrees in computer science from Stanford University, Stanford, CA.

In 1975 he joined the Computer Science Department at the University of Maryland, College Park, where he is now Professor. He also serves as the Director of the Graduate Program in Computer Science, is a member of the Computer Vision Laboratory of the Center for Automation Research, and has an appointment in the University of Maryland Institute for Advanced Computer Studies. His research interests are data structures, computer graphics, geographic information systems, computer vision, robotics, programming languages, artificial intelligence, and database management systems.