PROCEEDINGS OF THE

23rd IEEE CONFERENCE ON DECISION & CONTROL

1884 1984
A CENTURY OF ELECTRICAL PROGRESS
IEEE

CSS

IEEE
Control
Systems
Society

DECEMBER 12-14, 1984
LAS VEGAS HILTON
LAS VEGAS, NEVADA

84CH2093-3

Proceedings of 23rd Conference
on Decision and Control
Las Vegas, NV, December 1984

EFFECT OF UNCERTAINTY ON CONTINUOUS PATH PLANNING FOR AN AUTONOMOUS VEHICLE

Vladimir J. Lumelsky and Alexander A. Stepanov*

General Electric Company
Corporate Research and Development
Schenectady, New York 12301

This paper describes one approach to the problem of path planning for an autonomous vehicle (an automaton) moving in two-dimensional space filled with obstacles. The approach is based on continuous processing of incoming local information about the environment. A continuous computational model for the environment and for the vehicle operation is presented. Information about the environment (the scene) is assumed to be incomplete except that at any moment the vehicle knows the coordinates of its target as well as its own coordinates. The vehicle is presented as a point; obstacles can be of any shape, with continuous borderline and finite size. Algorithms guaranteeing reaching the target (if the target is reachable), and tests for target reachability are presented. The efficiency of the algorithms is evaluated in terms of perimeters of obstacles met by the vehicle. It is shown that with the exception of some rather unusual initial positions of the vehicle relative to the obstacles, one of the presented algorithms guarantees an optimal path.

Introduction

To "plan a path" for an autonomous vehicle (called below a traveling automaton, TA) means to find a continuous trajectory leading from the initial position of TA to its target position. In this work, the environment (the scene) in which TA travels is defined in a two-dimensional plane. This does not mean that the described approach applies only to planar or near-planar cases. The constraint on the dimensionality implies only that TA travels along some surface (which may have all kinds of hills and valleys) of the three-dimensional space, and cannot leave this surface. The significance of introducing a surface is that when TA encounters an obstacle it can turn only left or right to pass it (whereas meeting an obstacle in "real" three-dimensional space — say, a spacecraft meeting a planet — would result in an infinite number of possibilities for passing the obstacle). The scene may be filled with obstacles. Obstacles can be of any shape and size, with the following (rather practical) constraints: (1) Each obstacle is a simple closed curve; this simply means that the obstacle borderline is a continuous curve with no self-intersections. (2) Obstacles do not touch each other. (3) Any circle of a given radius can intersect with only a finite number of obstacles. This guarantees a finite number of obstacles in an area of finite size, and alleviates some mathematical problems.

The existing body of work on path planning can be classified into two categories — works dealing with situations with complete information on the scene, and works which assume that the information on the scene is incomplete.

Although the problem considered here relates to situations with uncertainty, a brief review of the approaches for *path planning with complete information* is in order. A popular version of path planning with complete information is the "Piano Movers" problem. Given (in two- or three-dimensional space, 2D or 3D) a solid object of known size and shape, its initial and target position and orientation, and a set of obstacles whose shapes, positions, and orientations in space are fully described, the task is to find a continuous (2D or 3D, respectively) path for the object from the initial position to the target position while avoiding collisions with obstacles along the way.

A number of approaches have been suggested [1,2] for the Piano Movers problem, with convex obstacles presented as polygons. Conceptually, final dimensions of the solid object can be viewed as shrinking to a point, while the obstacles are viewed as expanding inversely to the shape of the object. This requires increasing the dimensionality of the initial space — one extra dimension per each degree of rotational freedom. Resulting obstacles have nonplanar walls (even if original obstacles are polyhedra). Typically, various constraints are imposed in order to keep the problem manageable. For example, in [3], the problem of handling an object's orientation is alleviated by considering a circular object moving in two-dimensional space. In [4], the problem of two-dimensional path planning with a convex polygon object and convex polygon obstacles is solved using a generalized cylinders presentation [5] which reduces the problem to a graph search; a generalized cylinder is formed by a volume swept by a cross section (in general, of varying shape and size) moving along an axis (in general, a spine curve). The two- and three-dimensional problems of moving a solid polygon or polyhedron object in polynomial time were solved in [6] by direct computation of the "forbidden" volumes in spaces of higher dimensions d ($d=3$ for the 2D case, and $d=6$ for the 3D case).

A version of the Piano Movers problem where the moving object is allowed to consist of a number of free-hinged links is more difficult. This version was started by Pieper [7] and Paul [8] because of its obvious relation to path generation and coordinate transformation problems of multiple-degrees-of-freedom robot arms. Recently, the computational complexity of this version of the problem was investigated and new approaches were suggested in [6], [9], and [10].

Works on *path planning with incomplete information* come mainly from studies on autonomous vehicle navigation. In [11], [12], and [13] a two-dimensional navigation problem is considered. Obstacles are approximated by polygons; produced paths lie along edges of a connectivity graph formed by polygon vertices, the start point, and the target point, with an obvious constraint on intersection of the path with obstacles. Path planning is limited to the automaton's immediate surroundings for which information on the scene is available (for example, from a vision module). Within these surroundings, the problem is actually treated as one with complete information.

Since often (especially in natural scenes) obstacles can be approximated by polygons in different ways, the paths generated by these algorithms tend to strongly depend on specific approximations. With finer approximation of the obstacles, more nodes are introduced, and resulting paths can change drastically. On the other hand, the approximation itself depends on considerations that are actually secondary to the path planning problem (such as accuracy of presentation or — a conflicting criterion — computational costs of processing connectivity graphs). In this category of

* Currently with Polytechnic Institute of New York, Department of Electrical Engineering and Computer Science, Brooklyn, New York 11201

works, not much attention has been paid to the specificity introduced by the assumption of uncertainty of the information on the environment, or to the termination properties and path analysis of the proposed strategies.

In this paper, a continuous model of the environment (the scene) and of the automaton operation is considered. Under this model, the automaton is assumed to be continuously analyzing the incoming local information on its surroundings and continuously planning its path. This is very similar to the philosophy of treatment of geometrical phenomena based on local information developed in [16]. No approximation of obstacles (e.g., by polygons) is done, and, consequently, no connectivity graphs arise. Since no reduction to a discrete space takes place, all the points of space (and not only those points that lie along certain subspaces — e.g., along edges of a connectivity graph) are available for path planning purposes.

Because of the continuous model, a new type of path planning algorithm appears. Two algorithms (called Basic Algorithms), both guaranteeing convergence and, because of their very different characteristics, applicable to a wide range of scenes, are described. Also, a third algorithm, which combines strong sides of the Basic Algorithms, is introduced. The usually applied criteria for evaluating performance (such as, for example, computational complexity as a function of the number of nodes of the connectivity graph) are not applicable to these algorithms. Hence, a performance criterion based on the length of generated paths as a function of obstacle perimeters is introduced.

Because of space limitations, proofs are not given for the statements formulated below; these will be presented in a forthcoming publication [17].

Model

The model includes two parts — one related to the geometry of the scene, and another related to the characteristics and actions of the traveling automaton.

Environment

The scene in which TA travels, and TA's Start (S), Target (T), and path points, are defined in a plane. A scene can contain obstacles each of which is a simple closed curve. Obstacles or their parts do not touch each other; that is, a point on an obstacle (or on a part of an obstacle) belongs to one and only one obstacle (or part of an obstacle). A scene can have a locally finite number of obstacles. Formally, this means that any circle of a limited radius or any straight line segment in the plane will intersect with a finite set of obstacles. Any obstacle is homeomorphic to a circle; that is, for any obstacle there is some continuous topological mapping that transforms the obstacle into a circle.

Automaton

TA is a point; this means that an opening of any size between two distinct obstacles is considered to be passable. (In practice finite dimensions of the TA must be taken into consideration; in this work, TA's size and shape are ignored). The only information TA is provided with by its "sensors" is its current position (coordinates), and a fact of hitting an obstacle ("force sensor"). At the beginning, TA is given the position of Target. Therefore, TA can always calculate its direction on and its distance from Target. For simplicity of presentation, assume that the position of Target is fixed, although this fact is never used in the sequel.

In terms of its movement, TA is capable of three actions: move toward Target on a straight line, move along an obstacle, and stop.

A *local direction* is defined as a once-and-for-all-decided direction of passing around an obstacle. For the two-dimensional problem, it can be either left or right. Because of incompleteness of information, when TA hits an obstacle, there is no information or criteria that could help it decide whether it should go around the obstacle from the left or from the right. For the sake of clarity and without losing generality, assume that the local direction of TA is always *left* (as in Figure 2).

TA is said to *define* a Hit point H when, while moving along a straight line toward Target, TA hits the point H of an obstacle; it defines a Leave point L when it starts moving along a straight line from the point L toward Target. (See, for example, Figure 2).

Throughout, the following notation is used:

D is the distance from Start to Target

$d(A,B)$ is the distance between any points A and B of the scene; thus, d (Start,Target)$=D$

$d(A_i,B)$ signifies the fact that the point A is located on the borderline of the i^{th} obstacle met by TA on its way to Target

$d(A_i)$ is used as a shorthand notation for d (A_i,Target)

P is the total length of the path generated by TA on its way from Start to Target

p_i is the perimeter of the i^{th} obstacle

The performance of the presented algorithms will be evaluated using a quantity Σp_i, the sum of perimeters of obstacles met by the TA on its way to Target. This quantity will allow us to compare various path planning procedures in terms of the length of paths they produce.

Lower Bound for Path Planning Problem

The lower bound determines, within the framework of the environment and TA models, what ultimate performance can be expected from any path planning algorithm. The lower bound (formulated in the Theorem 1, [17]) is a powerful means for measuring performance of path planning procedures. It states that for any algorithm of path generation with uncertainty, there is a scene for which the length P of the generated path will obey the relationship

$$P \geqslant D + \Sigma p_i - \delta \tag{1}$$

where P, D, and p_i have been defined above, and δ is any constant.

This statement suggests that no matter what algorithm someone will come up with, a scene can be designed such that the length of the path generated by this yet-unknown algorithm will satisfy (1). In the following sections, the performance of each of the introduced path planning algorithms (in terms of the generated paths) will be compared with this lower bound.

First Basic Algorithm: Bug1

The procedure Bug1 is executed at any point of a continuous path. Figure 1 demonstrates the behavior of TA. When hitting an i^{th} obstacle, TA defines a Hit point H_i, $i=1,2,\ldots$. When leaving the i^{th} obstacle (to continue its travel toward the Target), TA defines a Leave point L_i; $L_0=$Start. The procedure consists of the following steps:

Step 1. Starting at L_{i-1}, TA moves toward the Target along a straight line until it hits an i^{th} obstacle, thus defining a point H_i. Or it may reach Target, in which case it stops.

Step 2. From H_i, TA starts moving along the i^{th} obstacle borderline using the accepted local direction, while looking for the point of minimum distance to Target. By the time TA makes a full circle around the i^{th} obstacle, it knows the point of minimum distance; this point is defined as L_i. In case the point L_i is not unique, only such a point is used which corresponds to a shorter path from H_i to L_i. If there is more than one point with this shorter path from H_i, any one of them is defined as L_i. A shorter path from H_i to L_i may correspond now to any direction around the obstacle (left or right), and not necessarily to the local direction.

Step 3. Now TA moves around the i^{th} obstacle, along the shorter path, to the point L_i. Go to Step 1.

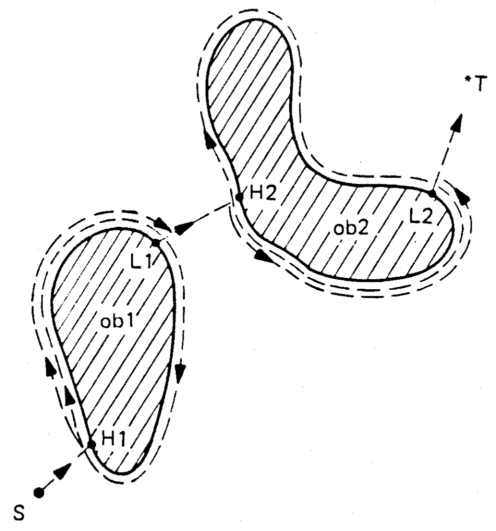

Figure 1. Automaton's path (dotted line), algorithm Bug1. Obstacles: ob1, ob2; Hit points: H1, H2; Leave points: L1, L2.

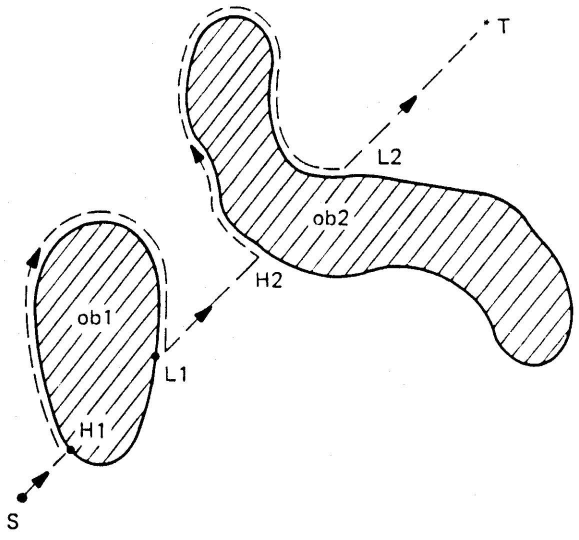

Figure 2. Automaton's path (dotted line), algorithm Bug2.

It can be shown that the procedure never creates cycles and that it always converges. Also, the length of the path produced by this procedure will never exceed the limit

$$P = D + 1.5 \cdot \Sigma\, p_i \qquad (2)$$

This suggests (compare (2) and (1)) that even if some algorithm better than Bug1 does (or will) exist, it cannot exceed the performance of Bug1 (as measured by the length of the path) by more than one-third.

According to the model, every time TA leaves an obstacle for Target there should be some distance between it and the next obstacle, if any. If either Target or Start happens to be trapped and, therefore, Target is not reachable, then at some point TA, when ready to leave for Target, will be facing an obstacle. This simple fact is used in the *target reachability test*, which is formulated as follows: If, while using the algorithm Bug1, after having defined a point L, TA discovers that the straight line (L,Target) crosses some obstacle at the point L, then Target is not reachable.

Second Basic Algorithm: Bug2

The procedure Bug2 is executed at any point of a continuous path. As will be clear later, the algorithm does not always distinguish between different obstacles. Therefore, in addition to the subscript i to indicate the i^{th} obstacle, the superscript j will be used to indicate the j^{th} occurrence of the Hit or Leave points — on the same or a different obstacle; the subscript i will be used only when necessary to refer to more than one obstacle; L^0=Start (see an example in Figure 2). The procedure consists of the following steps:

Step 1. TA moves from L^{j-1} along a straight line (Start,Target) until it hits an obstacle at some point H^j, j=1,2,... (point H1, Figure 2); it may also reach Target, in which case it stops.

Step 2. Then, TA begins moving along the obstacle borderline, always using the accepted local direction, until it reaches a Leave point, L^j, (point L1, Figure 2) which satisfies two requirements: L^j is located on the straight line (Start,Target), and the distance from L^j to Target is smaller than the distance from H^j to Target, $d(L^j) < d(H^j)$. Go to Step 1.

Note that unlike the previous algorithm, more than one point Hit and more than one point Leave may be generated during "processing" of a single obstacle (see Figure 3). Also, the relationship between perimeters of obstacles and the length of paths generated by Bug2 is not as clear as in the case of Bug1. Namely, for some scenes, Bug2 may create shorter paths than Bug1; often the path around an obstacle will be shorter than the obstacle perimeter (compare Figures 1 and 2). In some unfortunate cases when a straight line segment of the path meets an obstacle almost tangentially and TA goes around the obstacle in a "wrong" direction, the path actually may be equal to the full perimeter of the obstacle (see Figure 4). Finally, as Figure 3 demonstrates, the situation may get even worse, and TA may have to pass along some segments of a maze-like obstacle more than once.

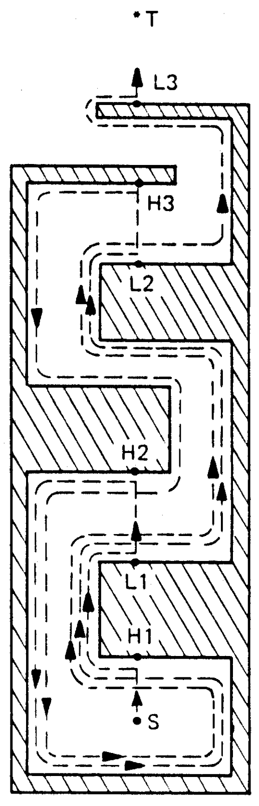

Figure 3. Automaton's path in a maze-like obstacle, algorithm Bug2. The obstacle complexity is measured by the number of times, n_i, the straight line (S,T) crosses it. Here n_i= 10. At most, the path passes one segment (H1,L1) three times; that is, there are at most two local cycles in this path.

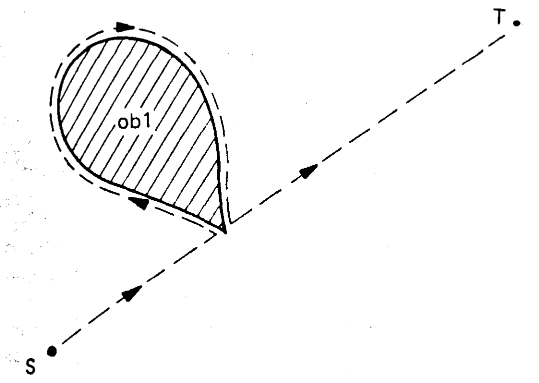

Figure 4. A case when, under the algorithm Bug2, the automaton will have to make almost a full circle around a convex obstacle.

Some new definitions: a *local cycle* is said to be created when TA passes some segment of the same obstacle *more than once.* In the example in Figure 2, no cycles are created; in Figure 3 there are some local cycles. A case of an *in-obstacle* (Figure 3) refers to a mutual position of the pair of points (Start and Target) and a given obstacle where (1) the interval of the corresponding straight line (Start,Target) crosses the obstacle borderline at least once, and (2) either Start or Target lie inside the minimum convex hull of the obstacle. A case of *out-obstacle* (Figure 2) refers to such a mutual position of the pair (Start and Target) and the obstacle in which both points Start and Target lie outside the minimum convex hull of the obstacle. Below, n_i is the number of intersections between the straight line (Start, Target) and the i^{th} obstacle; thus, n_i is a characteristic of the set (scene,Start,Target) and not of a specific algorithm. Obviously, for any convex obstacle $n_i = 2$.

If an obstacle is not convex, the situation still may be as simple as for convex obstacles; no local cycles appear if $n_i = 2$ (Figure 2, obstacle ob2). However, in Figure 3 the segment of the borderline from H1 to L1, (H1,L1), will be passed three times; segments (L1,L2) and (H2,H1), twice each; and segments (L2,L3) and (H3,H2), once each.

This procedure can be proven to converge, with the path whose length never exceeds the limit

$$P = D + \Sigma\, n_i p_i / 2 \qquad (3)$$

This limit is constructive, in the sense that simple scenes can be designed for which generated paths will be as close to the upper bound (3) as one wishes.

As for the performance of the algorithm Bug2, the upper bound (3) looks rather depressive; namely, it suggests that, under Bug2, sometimes TA must go around an obstacle any (large, but finite) number of times. Because of this, an important question is how typical "bad" scenes are, and in particular, what characteristics of a scene influence the length of the path. Fortunately, for out-obstacle situations that can be expected to prevail in applications, it can be shown that TA will pass the obstacle's perimeter *at most once.* Moreover, if all obstacles met by TA on its way to Target can be assumed to be convex (or, even less, if for them $n_i = 2$) then, on the average, the length of the path produced by the procedure Bug2 is

$$P = D + 0.5 \cdot \Sigma\, p_i \qquad (5)$$

and the length of the path produced for the worst scene is

$$P = D + 1.0 \cdot \Sigma\, p_i \qquad (6)$$

Therefore, for a wide range of scenes the length of paths generated by the algorithm Bug2 will not exceed the universal lower bound (1).

Based on the mechanism of defining Hit and Leave points in the procedure Bug2, a simple *test for target reachability* can be formulated: If, on the p^{th} local cycle, $p = 0,1,...$, after having defined a point H^j, TA returns to this point before it defines at least the first two out of the possible set of points $L^j, H^{j+1}, ..., H^k$, it means that TA has been *trapped* and, hence, that Target is not reachable (Figure 5).

Improving Performance of Basic Algorithms

In the actual implementations, improvements based on combining features of the Basic Algorithms can be introduced. Although the flow of action in such modified versions may be not as "clean" as in the Basic Algorithms, the termination properties and the estimates on the path length presented above still apply. Such a version (called BugM1, for "modified") consists of the following steps:

Step 1. TA moves from L^{j-1} (Leave point), $j = 1,2,...$, along a straight line (L^{j-1},Target) until it hits an obstacle at some point H^j (Hit point); or, it may reach Target, in which case it stops. L^0 = Start (i.e., the first Leave point coincides with Start).

Step 2. From H^j, TA begins moving along the obstacle using the accepted local direction until it defines a point L^j. Here, one of two possible cases occurs:

(a) While moving from H^j along the obstacle borderline, TA crosses the straight line (L^{j-1},Target) inside the interval (L^{j-1},Target); in this case TA defines a point L^j in such a way that it satisfies two requirements: L^j is located on the straight line (L^{j-1},Target), and the distance from L^j to Target is smaller than the distance from H^j to Target, $d(L^j) < d(H^j)$. Go to Step 1.

(b) While moving from H^j along the obstacle borderline, TA crosses the straight line (L^j,Target) outside the interval (L^j,Target); in this case TA defines a point L^j according to Steps 2 and 3 of the algorithm Bug1. Go to Step 1.

Notice that if the scene is such that only Step 2a is ever executed then the actual flow of the algorithm is that of Bug2, and the straight lines (L^j,Target) always coincide with the straight line (Start,Target). No local cycles can be created in such situations. If (in cases of in-obstacles) local cycles do appear, this creates the condition accounted for in Step 2b of BugM1. From the condition of Step 2b being satisfied, TA recognizes a danger of multiple local cycles, and "decides" to go to the conservative action of Bug1, which guarantees an upper bound (2), instead of risking the uncertain number of local cycles it can now expect under Bug2. It does this by executing Steps 2 and 3 of Bug1. After at least one execution of Step 2b, the straight line (L^j,Target), in general, no

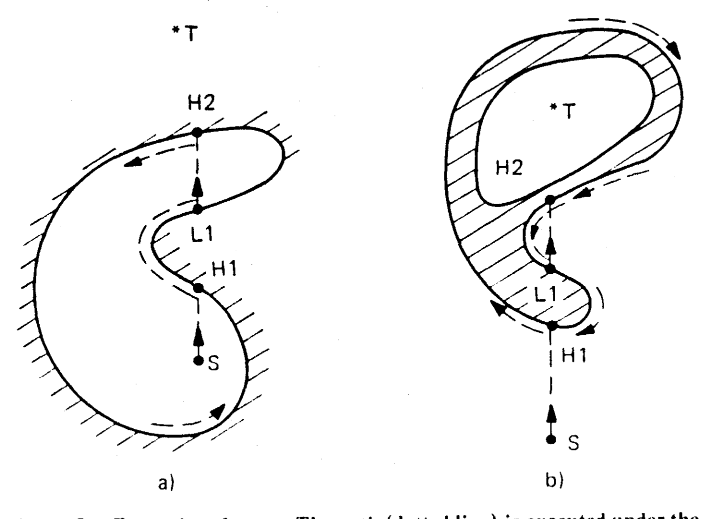

Figure 5. Examples of traps. The path (dotted line) is executed under the algorithm Bug2. After having defined the point H2, the automaton returns to it before defining any new point L. Therefore, the target is not reachable.

longer coincides with the straight line (Start,Target); instead, the straight line segments of the path look similar to those created by the algorithm Bug1 (see Figure 1).

In general, with such a modification, TA will have the efficiency of Bug2 (in the sense that it does not necessarily have to cover full perimeters of obstacles as in Bug1) while it is guaranteed to never pass the same segment of the obstacle borderline more than three times.

Some Remarks on Performance of Basic Algorithms

Depending on the scene, one Basic Algorithm may produce a path significantly shorter than the other. The question of when which algorithm should be used goes beyond formal analysis. One could say, for example, that the algorithm Bug1 probably will appeal to a conservative (pessimistic) TA, whereas the algorithm Bug2 might appeal to a more optimistic TA.

If TA wants to minimize the effort (path length) for the worst scenes (a pessimistic TA), Bug1 provides a guarantee that the path will never exceed the limit (2). Unfortunately, Bug1 will never produce a path as short as the one shown in Figure 2, but, on the other hand, it will never create local cycles.

However, if TA wants to minimize the effort on simple scenes, or if it has some reason to believe that the scene in question will not present any unpleasant surprises (an optimistic TA) then it will use Bug2, which for any convex or simpler nonconvex obstacles promises paths as short as given by (5). Another reason for the optimistic TA to be optimistic, and thus to use Bug2 instead of Bug1, is provided by Theorem 5 which guarantees that even for the most complicated scenes, the path will never exceed (6) (which is better than (2) for Bug1) if the mutual positions of Start, Target, and obstacles correspond to a case of an out-obstacle.

An additional insight into the operation and the "area of expertise" of the Basic Algorithms is gained by trying to use them in maze search problems. The problem of search in an unknown maze may be set in a number of ways. In one version (see, e.g., [14]) TA, starting at an arbitrary cell of the maze, must eventually visit every single cell without passing through any barriers. (This means, of course, that any pair of cells in the maze is connected via other cells). Notice that in this version there is no notion of a Target cell whose coordinates are known; no sense of direction is present. Because of that, neither of the Basic Algorithms can be used.

In another version of the maze search problem, given a starting cell, TA is to find an exit from the maze; the coordinates of the exit are not known. Although no target is presented explicitly, TA may choose any point (direction) somewhere in infinity, and then use the Basic Algorithms as usual. With such an operation, an exit is guaranteed to be found.

In still another version of the maze search problem [15], TA is given coordinates of two points (cells), S (Start) and T (Target), in a maze and is asked to find a route from S to T. Clearly, this version is the closest to the problem considered in this paper. For this version of the maze search problem, the behavior of the algorithm Bug2 is demonstrated in Figure 6 on a randomly designed maze with Start and Target points thrown randomly in more or less opposite directions of the maze. (Since maze search algorithms — see, e.g., [14] — typically use discrete models, Figure 6 presents a discrete version of the continuous path planning

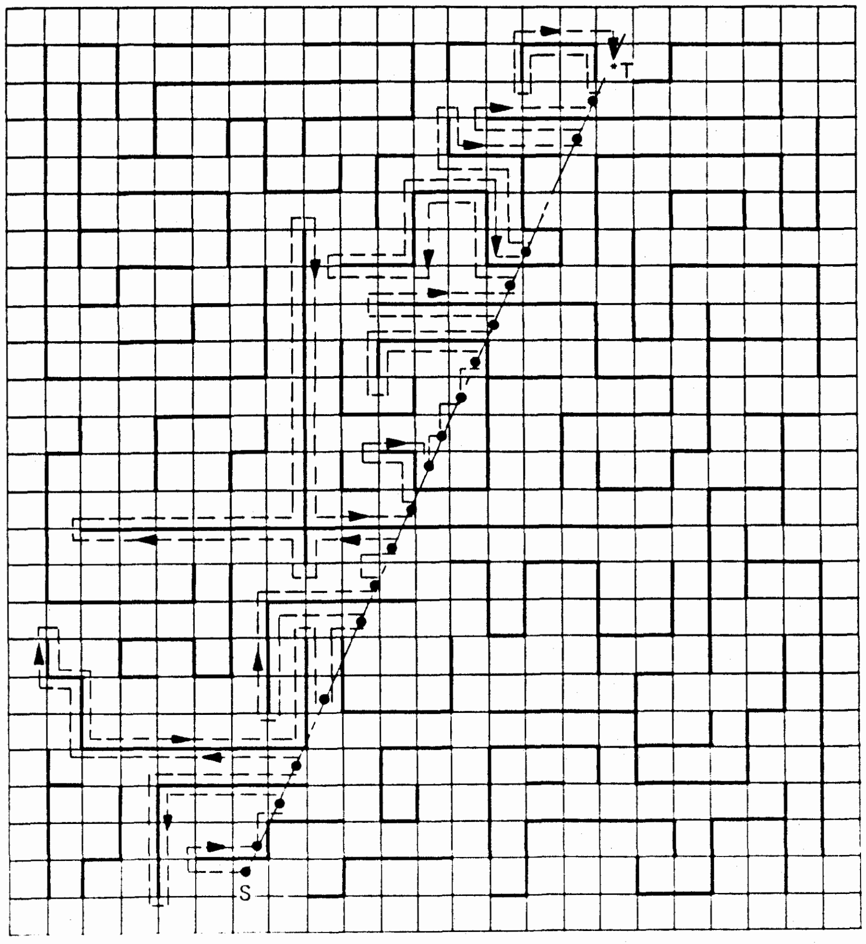

Figure 6. Example of walk in a maze, algorithm Bug2. S = Start, T = Target. Points at which the automaton's path (dotted line) crosses the imaginary straight line (Start,Target) are indicated by dots. Maze barriers are shown in thick lines.

roblem; TA walks through cells represented by little squares; any d touched by the straight line (S,T) is considered to be lying on e line). A quick look at the barriers between S and T suggests at here TA is operating in an out-obstacle scene and, therefore, e estimates (5),(6) should apply. Indeed, as Figure 6 demonrates, no local cycles are created, and the generated path, given e fact that TA knows nothing about the design of the maze, is ather good.

References

[1] J.T. Schwartz, M. Sharir, "On the 'Piano Movers' Problem. I. The Case of a Two-Dimensional Rigid Polygonal Body Moving Amidst Polygonal Barriers," New York University Dept. of Computer Science, Tech. Report No. 39, October 1981.

[2] T. Lozano-Perez, M. Wesley, "An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles," *CACM 22* (1979).

[3] H.P. Moravec, "Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover," Stanford AIM-340, Sept. 1980.

[4] R.A. Brooks, "Solving the Find-Path Problem by Representing Free Space as Generalized Cones," MIT Artificial Intelligence Laboratory, AI Memo No. 674, May 1982.

[5] T.O. Binford, "Visual Perception by Computer," IEEE Systems Science and Cybernetics Conference, Miami, December 1971.

[6] J. Reif, "Complexity of the Mover's Problem and Generalizations," Proc. 20th Symposium of the Foundations of Computer Science, 1979.

[7] D.L. Pieper, *The Kinematics of Manipulators Under Computer Control,* Ph.D. Thesis, Stanford University, October 1968.

[8] R. Paul, *Modelling Trajectory Calculation and Servoing of a Computer Controlled Arm,* Ph.D. Thesis, Stanford University, November 1972.

[9] J.T. Schwartz, M. Sharir, "On the 'Piano Movers' Problem. II. General Techniques for Computing Topological Properties of Real Algebraic Manifolds," New York University Dept. of Computer Science, Tech. Report No. 41, February 1982.

[10] J. Hopcroft, D. Joseph, S. Whitesides, "On the Movement of Robot Arms in 2-Dimensional Bounded Regions," Proc. of the IEEE Foundations of Computer Science Conference, Chicago, November 1982.

[11] B. Bullock, D. Keirsey, J. Mitchell, T. Nussmeier, D. Tseng, "Autonomous Vehicle Control: An Overview of the Hughes Project," Proceedings of IEEE Computer Society Conference *Trends and Applications, 1983: Automating Intelligent Behavior,* Gaithersburg, Maryland, May 1983.

[12] A.M. Thompson, "The Navigation System of the JPL Robot," Proceedings of 5th Joint International Conf. on Artificial Intelligence, Cambridge, Massachusetts, August 1977.

[13] D.M. Kersey, E. Koch, J. McKisson, A.M. Meystel, J.S.B. Mitchell, "Algorithm of Navigation for a Mobile Robot," Proc. of International Conference on Robotics, IEEE Computer Society, Atlanta, Georgia, March 1984.

[14] M. Blum, D. Kozen, "On the Power of the Compass (or, Why Mazes are Easier to Search than Graphs)," Proc. of the 19th Annual Symposium on Foundation of Computer Science, Ann Arbor, Michigan, October 1978.

[15] W. Lipski, F.P. Preparata, "Segments, Rectangles, Contours," *Journal of Algorithms 2,* 1981.

[16] H. Abelson, A. diSessa, *Turtle Geometry,* MIT Press, 1980.

[17] V. Lumelsky, A. Stepanov, "Path Planning Strategies for a Traveling Automaton in an Environment With Uncertainty," submitted for publication.