CRD

CORPORATE RESEARCH AND DEVELOPMENT · SCHENECTADY, NEW YORK

NAVIGATION STRATEGIES FOR AN AUTONOMOUS VEHICLE WITH INCOMPLETE INFORMATION ON THE ENVIRONMENT

by
V. Lumelsky and A. Stepanov
Information Systems Laboratory

Report No. 84CRD070 April 1984

TECHNICAL INFORMATION SERIES

CLASS 1

GENERAL ELECTRIC

GENERAL ELECTRIC

General Electric Company
Corporate Research and Development
Schenectady, New York 12345

TECHNICAL INFORMATION SERIES

AUTHOR
Lumelsky, V
Stepanov, A

SUBJECT
robotics, artificial intelligence

NO.
84CRD070

DATE
April 1984

TITLE
Navigation Strategies for an Autonomous Vehicle with Incomplete Information on the Environment

GE CLASS
1

NO. PAGES
13

ORIGINATING COMPONENT
CORPORATE RESEARCH AND DEVELOPMENT SCHENECTADY, N.Y.

SUMMARY

A problem of path planning for an automaton moving in two-dimensional space filled with obstacles is considered. A new computational model — a continuous model — for the environment and for the automaton operation is presented. Information about the environment (the scene) is assumed to be incomplete except that at any moment the automaton knows the coordinates of its target as well as its own coordinates. The automaton is presented as a point; obstacles may be of any shape, with continuous borderline and finite size. Algorithms guaranteeing reaching the target (if the target is reachable) by the automaton are presented. Tests for ensuring that the target is (or is not) reachable are developed.

Minimum worst case upper bounds are given on the length of paths generated by the algorithms; these are expressed in terms of perimeters of obstacles in the scene. Also, a universal lower bound on the path length for the path planning problem is presented. It is shown that unless the automaton in initially positioned rather unusually relative to the obstacles, one of the presented algorithms guarantees a path whose length reaches the universal lower bound.

KEY WORDS
autonomous vehicle, path planning, robotics, navigation strategies, artificial intelligence, maze traversing

INFORMATION PREPARED FOR _____

_____

Additional Hard or Microfiche Copies Available from

**Technical Information Exchange
Bldg. 5 Room 321, Schenectady, N.Y. 12345**

NAVIGATION STRATEGIES FOR AN AUTONOMOUS VEHICLE WITH INCOMPLETE INFORMATION ON THE ENVIRONMENT

V. Lumelsky and A. Stepanov

1. INTRODUCTION

To plan a path for a traveling automaton means to find a continuous trajectory leading from the initial position of the automaton to its target position. The environment (the scene) in which the automaton travels is defined in a (two-dimensional) plane. The scene may be filled with obstacles. Obstacles may 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 may intersect with only a finite number of obstacles.

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

In *path planning with complete information*, one popular version is the "Piano Movers" problem. Given (in two- or three-dimensional space) a solid object of known dimensions, its initial and target position and orientation, and a set of obstacles whose shapes, positions and orientations in space are known, the task is to find a continuous (2-D or 3-D) path for the object from the initial position to the target position while avoiding collisions with obstacles along the way.

Schwartz and Sharir [6] solve a two-dimensional case of the "Piano Movers" problem with convex obstacles presented as polygons; the presented algorithm is shown to produce a solution in a polynomial time $0(n^5)$ where n is the number of distinct convex walls of obstacles.

In a number of works, final dimensions of the solid object are 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 non-planar walls (even if original obstacles are polyhedra). In order to keep the problem manageable, it is typical to impose various constraints. For example, Lozano-Perez [8] allows the object to change its orientation only once, if at all, during its motion.

Moravec [7] considers a path planning algorithm in two dimensions, with the object presented as a circle. Brooks, [9] in his treatment of a two-dimensional path planning problem with a convex polygon object and convex polygon obstacles, uses generalized cylinders presentation of space [10] to reduce the problem to a search on a graph; 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). Reif [11] solves the two- and three-dimensional problems of moving a solid polygon or polyhedron object in polynomial time, by direct computation of the "forbidden" volumes in spaces of higher dimensions d (d = 3 for a two-dimensional "Piano Mover" problem, d = 6 for a three-dimensional problem).

A version of the "Piano Movers" problem, where the moving object is allowed to consist of a number of free-hinged links, is a more difficult problem. This version was started by Pieper [15] and then investigated by Paul [13] because of its obvious relation to path generation and coordinate transformation problems of multiple-degrees-of-freedom robot arms. Schwartz and Sharir [12] gave a general (but not polynomial time) algorithm for generating a path for a free-hinged object. Reif [11] showed that generating a path for a free-hinged object is, in general, a P-space complete problem. Following this conclusion, some limited versions of the problem have been considered. Hopcroft et al. [14] gave a polynomial-time path planning algorithm for a two-dimensional free-hinged object shaped like a folding carpenter ruler. Even this limited two-dimensional problem was shown to become difficult if the links of the carpenter's ruler are not of equal length.

Among works on *path planning with incomplete information*, in [5] a two-dimensional navigation problem is considered. Obstacles are described by polygons, produced paths lie along edges of the connectivity graph formed by obstacle vertices, the start point, and the target point, with an obvious constraint on intersection of the path with obstacle polygons. Path planning is limited to the automaton's immediate surrounding for which information on the scene is complete; within this surrounding, the problem is actually treated as one with complete information.

In the work by Bullock et al. [1] on the autonomous vehicle control, the input information on the scene comes from an image acquistion and understanding module. [2,3] (Techniques for deriving two-dimensional structure of the scene from partial snapshots of the 3-D terrain taken by a moving observer, have been studied, e.g., by Lavin [4]).

Manuscript received March 16, 1984

The work uses knowledge-based techniques and relies heavily on canned strategies for typical recurring situations (e.g., going around an obstacle from the left). The system emphasizes usage of prestored two-dimensional map information on the scene. Incoming (visual) sensor information is used for fine tuning of the path. Obstacles are represented as polygons; path planning strategy is based on search on a connectivity graph.

In the work presented here, a continuous model of the environment (the scene) and of the automaton operation is considered. That is, the automaton may be measuring its coordinates and planning its actions continuously. No approximation of obstacles (e.g., by polygons) is done, and, consequently, no connectivity graphs arise. Since no reduction to a discrete space is done, 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. Due to continuous models, a new type of path planning algorithms appears. For these, the usually used criteria for measuring their performance (such as computational complexity as a function of the number of nodes of the connectivity graph, or the time or memory required) are no more applicable; hence, new performance criteria — in particular, those dealing with the length of generated paths, as a function of obstacle perimeters — are introduced.*

A simple traveling automaton (called a *minimal automaton*) will be considered that has no knowledge about the scene except its current coordinates and the coordinates of the target. Its only sensor lets it feel an obstacle when it hits. Expansion of the approach presented here on situations when the automaton has additional sources of information (such as vision or prestored maps) and additional capabilitities, is a subject of future work.

First, the model of the environment and of the automaton is formulated. Then, a universal lower bound (that is, one applicable to any theoretically feasible algorithm) for the path planning problem is produced. After that, two Basic Algorithms for path planning are described. As one will see, the algorithms have different characteristics, and, depending on the scene, one of them may produce a shorter path than the other. For the algorithms, the upper bounds on the length of generated paths are established; these are compared with the universal lower bound on the length of generated paths; also, tests for target reachability are formulated. This is followed by an improved version of the path planning algorithm that combines good sides of both Basic Algorithms while not sacrificing much of their clarity. In the last section, the performances of the algorithms are compared, and an additional insight into the algorithms' mechanisms is provided by showing their relevance to the maze search problem.

*Because of incompleteness of information, the path cannot be preplanned, and so its global optimality is ruled out. Instead, one can judge on performance of algorithms by how optimal algorithms are locally, or how "reasonable" they look from the standpoint of a human traveler; or how they compare with other existing or theoretically feasible algorithms.

II. MODEL

The model to be considered includes two parts: one related to the geometry of the scene, and another related to the characteristics and actions of the automaton.

Environment. The scene in which the automaton travels is defined in a plane. A scene may 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 (part of an obstacle). A scene may have a locally finite number of obstacles. Formally, it 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. The automaton 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 automaton will have to be taken into consideration; in this work the automaton's size and shape are ignored). The minimal automaton has "sensors" that provide it with the following information: 1) its current coordinates, 2) the fact that it hit an obstacle ("force sensor"). It is assumed that the automaton knows the coordinates of the target. The automaton, therefore, may always calculate its direction on, and its distance from, the target.

In terms of its movement, the automaton is capable of three actions: 1) move toward the target on a straight line, 2) move along an obstacle, 3) stop.

Definition 1. A *local direction* is always a decided direction of passing around an obstacle. For the two-dimensional problem, it can be either left or right.

Because of incompleteness of information, every time a minimal automaton 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 the automaton is always *left* (as in Figure 4). Unless told otherwise, the automaton will be assumed to follow the local direction while walking around obstacles.

Definition 2. The automaton *defines* a Hit point H when, while moving along a straight line toward the

Target, the automaton 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 the Target. (See, for example, Figure 4).

In the following sections, these notations will be 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 the automaton 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 the automaton on its way from Start to Target,

p_i is the perimeter of the i-th obstacle,

The performance of the algorithms analyzed in the following sections will be evaluated using a quantity Σp_i the sum of perimeters of obstacles met by the automaton 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.

III. LOWER BOUND FOR PATH PLANNING PROBLEM

The lower bound determines, within the framework of the environment and the automaton models described above, what ultimate performance may be expected from any path planning algorithm. The lower bound (formulated in Theorem 1 below) is a powerful means for measuring performance of path planning procedures.

Theorem 1. For any algorithm of path generation, there is a scene for which the length P of the generated path will obey the relationship

$$P \geq D + \Sigma p_i - \delta \qquad (1)$$

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

Proof. This present work attempts to prove that no matter what algorithm someone comes up with, a scene may be designed for which the length of the path generated by this yet unknown algorithm (called Algorithm X), will satisfy (1). Algorithm X may be deterministic or random; its intermediate steps may or may not depend on intermediate results — it may be any; it is assumed to be operated by an automaton (AT) and within an environment whose models have been described above.

At the beginning, points Start and Target are defined and given to AT. Since AT starts at a distance $D = d$(Start,Target) from Target, it obviously cannot avoid the term D in (1). It is necessary to concentrate, then, on the second term in (1). We suggest a scheme for designing scenes such that for any X the designed scene will force X to generate a path not shorter than P in (1).

The defeating scene is built in two stages. We start with a *plan* of the scene which has in it some *maximum* obstacle — that is, an obstacle parts or all of which (but not more than that) will (on the second stage) make *actual* obstacle(s) of the scene.

Then let AT walk from Start to Target. If, on its way, AT hits the maximum obstacle, it will somehow have to go around it using its algorithm X. When the path is complete, the second stage starts; we observe the path and designate as parts of actual obstacle(s) only those areas of the maximum obstacle that have been touched by AT from *inside the corridor.** If AT moved along the maximum obstacle borderline, this area becomes a part of the actual obstacle; if AT only touched the maximum obstacle and then bounced back, an area of the length δ of the obstacle around the point of touch becomes a segment of an obstacle. The rest of the maximum obstacle can be left out because (at least, under the accepted model) the behavior of AT does not depend on the areas of obstacle(s) which it never touches. Therefore, the resulting scene could have been the one in which AT would operate and use its Algorithm X. This completes the design of the scene.

Now it is necessary to prove that the AT's path in the resulting scene obeys (1). At this point, one may see that the main idea behind the described process of designing an appropriate scene is that no matter what Algorithm X is, under the described process AT will always be forced to generate, for each actualized segment of an obstacle, a segment of the path at least twice as long.

Consider a maximum obstacle shown in Figure 1a: this is a corridor of finite width $2W > \delta$ and of finite length L, with the Start point located in the middle of the "bottom" (the closed end) of the corridor, and the axis of the corridor going in the direction opposite to the interval (Start,Target).** The width of the corridor walls is negligible compared to δ. AT's path in the scene may be divided into two parts, *P1* and *P2*. *P1* corresponds to AT's travel inside the corridor, and *P2* outside the corridor. Both parts may be intermixed since, after getting out of the corridor, AT may temporarily return into it. It is clear, though, that since part *P2* starts at the exit point of the corridor, then $P2 \geq L + C$ where C is the difference between the hypotenuse and the side in the triangle ATS. As for the

*With this condition of counting only the inside touching, we ensure that for each actualized segment of an obstacle, AT will generate at least a twice as long segment of the path.

** It is enough to show that (1) holds for a scene with one maximum obstacle. Notice, though, that depending on AT's behavior, one or more actual obstacles may be created in the scene of Figure 1.

part *PI*, there may be various alternatives all of which fall in one of three groups.†

1. Part *PI* of the path never touches the walls of the corridor (Figure 1a). As a result, no obstacles will be created ($p_i=0$) but a non-zero second term in (1) will appear. Therefore, for this kind of X the theorem holds. Moreover, at the final evaluation where only actual obstacles count, this AT strategy will not look very reasonable: it creates an additional path segment (second term in (1)) at least equal to $(2 \cdot L + C)$ — in a scene with no obstacles!

2. Part *PI* of the path is such that one or more of disconnected δ-segments on one or both side walls of the corridor (see Figure 1b) are actualized. As one can see, this kind of strategy is not very wise either: per each δ-segment of an actual obstacle, AT creates a larger segment of the path; the length of this path segment depends either on W (if the next δ-segment of the obstacle is located on the wall opposite to that of the previous δ-segment) or/and at least on the distance between two sequentially created disconnected δ-segments (if these are on the same wall of the corridor). Thus, the length P of the path exceeds the right side in (1), and the theorem holds.

3. AT closely follows the internal walls of the corridor, and then it chooses the shortest path to the Target (see Figure 1c). Here AT is doing its best in trying to compensate each segment of its path with an equivalent actual obstacle segment. In such a case, the generated path P will be equal to

$$P = \sqrt{D^2 + W^2} + (\Sigma p_i - W)$$

where there is only one term in Σp_i. Since we are free to choose the parameters D and W of the scene, they are taken such that

$$\delta \geq W + D - \sqrt{D^2 + W^2}$$

and, therefore, (1) is satisfied. Q.E.D.

† The actual path inside the corridor may be as capricious as one wishes — for example, it may follow some strange curves, or go back and forth, or go in circles, etc. Every segment of the path that does not "create" an equivalent actualized obstacle segment, makes, from the standpoint of (1), the path worse. Three groups above capture all possibilities in terms of relationship between the path length and the right side in (1).

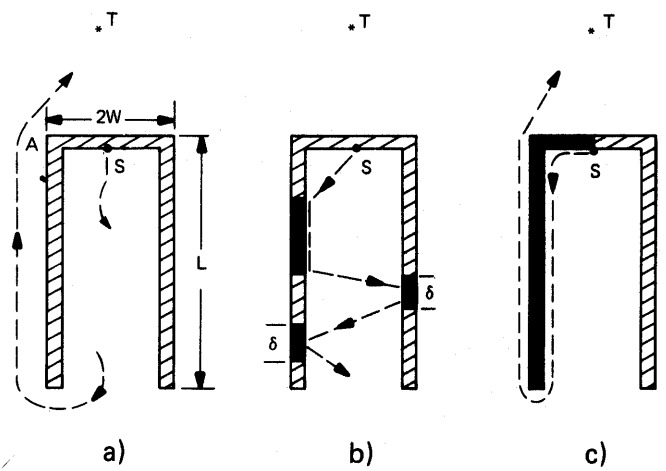

Figure 1. Illustration for Theorem 1. Actualized segments of the maximum obstacle are shown in solid. S — Start point, T — Target point.

IV. FIRST BASIC ALGORITHM: Bug1.

In this and the next sections, two specific path planning algorithms (called Basic Algorithms, or algorithms Bug1 and Bug2) are presented. Here, the First Basic Algorithm (procedure Bug1), the analysis of its characteristics, and a test for the target reachability when using Bug1, are discussed.

Procedure

The procedure Bug1 is executed at any point of a continuous path. Figure 2 demonstrates the behavior of the automaton. The goal is to generate a path from Start to Target. When hitting an i-th obstacle, the automaton defines a Hit point H_i, i=1,2,... . The automaton leaves the i-th obstacle (to continue its travel toward the Target)

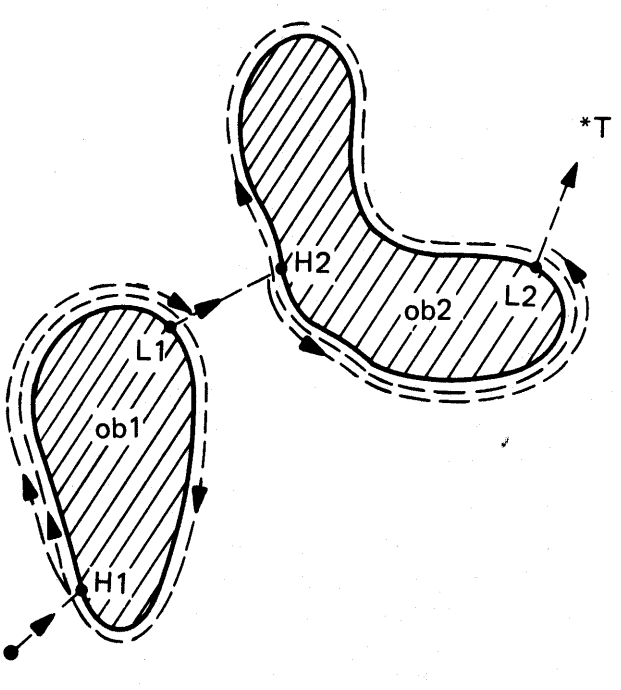

Figure 2. Automaton's path (dotted lines), algorithm Bug1. ob1, ob2 — obstacles; H1, H2 — Hit points; L1, L2 — Leave points.

from a Leave point L_i; L_o = Start. The procedure consists of the following steps.

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

Step 2. From H_i, the automaton starts moving along the obstacle borderline using the accepted local direction (as agreed above, this is always left), while looking for the point of minimal distance to the Target. By the time the automaton makes a full circle around the i-th obstacle, it knows the point of minimal distance L_i. In case the point L_i is not unique, only such a point is used as L_i which corresponds to a shorter path from H_i to 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, the automaton moves around the i-th obstacle, along the shorter path, to the point L_i. Go to Step 1.

Characteristics of Bug1.

The characteristics and performance of the algorithm Bug1 are analyzed in the following statements.

Lemma 1. Under Bug1, after leaving a Leave point of an obstacle on its way toward the Target, the automaton never returns to this obstacle again.

The Lemma guarantees that the strategy will never create cycles. Also, since, according to the accepted model of the environment, on its way to the Target the automaton may find only a finite number of obstacles (all of finite size), the Lemma guarantees convergence of the algorithm.

Proof. Assume that on its way from Start to Target, the automaton meets some obstacles. Number those obstacles in the order in which the automaton meets them. Then, a following sequence of distances appears:

$$D,\ d(H_1),\ d(L_1),\ d(H_2),\ d(L_2),\ d(H_3),\ d(L_3),\ldots$$

If point Start happened to be on the border of an obstacle that occludes the view of Target, then $D = d(H_1)$.

If the automaton's path touches the i-th obstacle tangentially then there is no need to invoke the procedure of walking around an obstacle — the automaton just continues its straight line walk toward Target. In all other cases of hitting an i-th obstacle, a relation will hold,

$$d(H_i) > d(L_i) \qquad (2)$$

This is because, on the one hand, according to the model, any straight line (except a line that touches the obstacle tangentially) crosses an obstacle at least in two distinct points (finite "thickness" of obstacles), and, on the other hand, according to the algorithm Bug1, point Leave is the closest point from the obstacle to Target. Starting from L_i, the automaton walks straight to Target until it meets the (i + 1)-th obstacle. Since, by the model, obstacles do not touch one another, a relation holds:

$$d(L_i) > d(H_{i+1}) \qquad (3)$$

Therefore, our sequence of distances obeys an inequality,

$$d(H_1) > d(L_1) > d(H_2) > d(L_2) > d(H_3) > d(L_3)\ldots \qquad (4)$$

where $d(H_1)$ is or is not equal to D. Since $d(L_i)$ is the shortest distance from the i-th obstacle to Target, and since (3) guarantees that the algorithm Bug1 monotonically decreases distances to Target, Lemma 1 follows. Q.E.D.

Corollary. Under Bug1, independent of the geometry of an obstacle, the automaton defines on it not more than one Hit and not more than one Leave point.

To produce an upper bound on the length of the paths generated by Bug1, an assurance is needed that on its way to Target, the automaton always encounters only a finite number of obstacles. This is not obvious since, following the algorithm Bug1, the automaton may "look" at Target from different attitudes (that is, besides moving toward Target, it may rotate around Target, see Figure 3) and from different distances.

Lemma 2. Under Bug1, on its way to Target the automaton may meet only a finite number of obstacles.

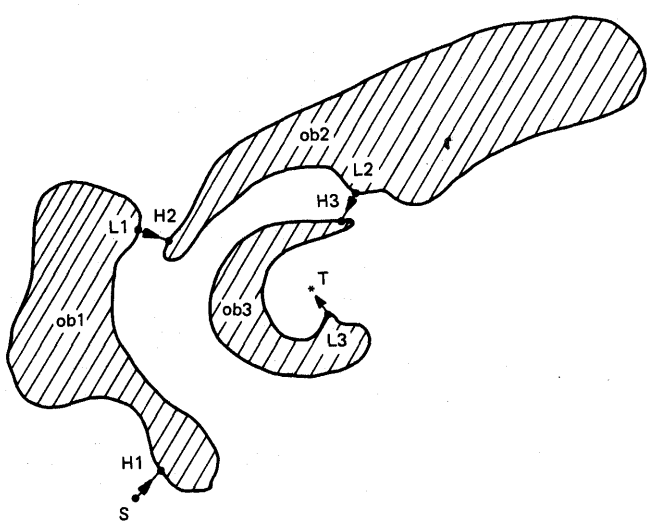

Figure 3. Algorithm Bug1. Dotted lines indicate straight-line segments of the automaton's path. Segments around obstacles are not shown; these are similar to the ones shown in Figure 2.

Proof. Although, while walking around an obstacle, the automaton may, at some moments, be at distances larger than D from Target (see Figure 3), its straight line segments are always within the same circle with the center in Target and with the radius D. This is guaranteed by the inequality (4). Since, by the model, any circle of finite radius may intersect with only a finite number of obstacles, the Lemma follows. Q.E.D.

The following theorem gives an upper bound on the length of the path produced by the procedure Bug1.

Theorem 2. The length of the path produced by the procedure Bug1 will never exceed the limit

$$P = D + 1.5 \cdot \Sigma p_i \qquad (5)$$

The Theorem shows (compare (5) and (1)) that even if some algorithm better than Bug1 does (or will) exist, it may not exceed the performance of Bug1 (as measured by the length of the path) by more than one third.

Proof. Any path may be looked at as consisting of two parts — straight line segments of the automaton's walk toward Target (between obstacles), and segments related to walking around obstacles. Because of the inequality (4), the sum of straight line segments will never exceed D. As to the path segments around obstacles, the algorithm Bug1 requires that in order to define a point Leave on the i-th obstacle, the automaton has to make a full circle around it; this produces a segment equal to one perimeter, p_i, of the obstacle. By the time the automaton is prepared to walk from Hit to Leave, in order to depart for Target, it knows the direction (go left or go right) of the shorter path to Leave. Thus, its path segment from Hit to Leave around the i-th obstacle will not exceed $0.5 \cdot p_i$. Summing up all the partial estimates for straight line segments of the path and for segments around all the obstacles that the automaton meets on its way to Target produces the upper bound (5). Q.E.D.

Test for Target Reachability by Algorithm Bug1.

Every time the automaton "studies" a new obstacle it defines an L point on it; then it starts moving from L to Target along a straight line. If, after having defined the point L, the automaton discovers that the straight line (L,Target) crosses an obstacle at the point L, it may mean only that Target is not reachable — either Start is *trapped* inside the current obstacle, or Target is trapped inside the current obstacle. This simple fact is used in the test.

Test for Target Reachability. If, while using the algorithm Bug1, after having defined a point L, the automaton discovers that the straight line (L,Target) crosses the obstacle at the point L, then Target is not reachable.

V. SECOND BASIC ALGORITHM: Bug2.

In this section, the procedure Bug2, analysis of its characteristics, and a test for target reachability when using Bug2, are presented.

Procedure

The procedure Bug2 is executed at any point of a continuous path. The goal is, again, to generate a path from Start to Target. Since, as will be clear, the algorithm does not always distinguish between different obstacles, in addition to the subscript i to indicate the i-th obstacle, we will use the superscript j to indicate the j-th occurrence of the Hit or Leave points — on the same or a different obstacle; L^0 = Start. The behavior of the automaton under Bug2 is demonstrated on the example shown in Figure 4. The procedure consists of the following steps.

Step 1. The automaton 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 4); it may also reach Target in which case it stops.

Step 2. Then, the automaton begins moving along the obstacle (always using the accepted local direction) until it reaches a Leave point, L^j, (point L1, Figure 4) which satisfies two requirements: 1) L^j is located on the straight line (Start, Target), and 2) 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.

Notice that unlike the algorithm Bug1, more than one point Hit and more than one point Leave may be

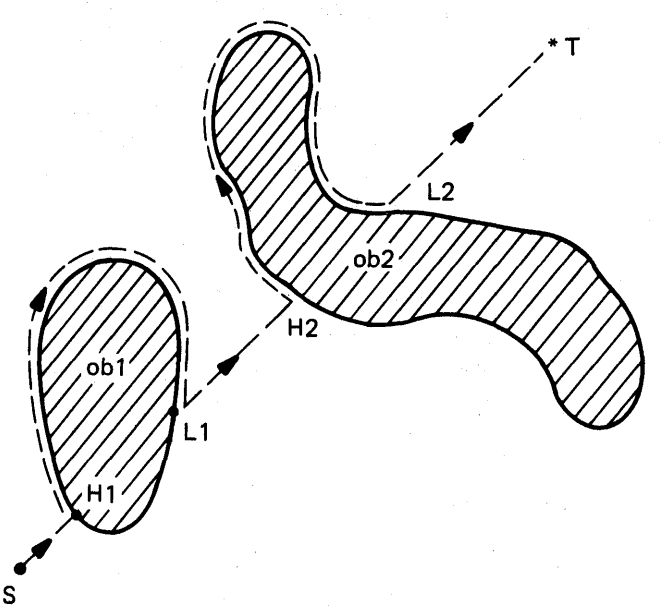

Figure 4. Automaton's path (dotted line) under the algorithm Bug2.

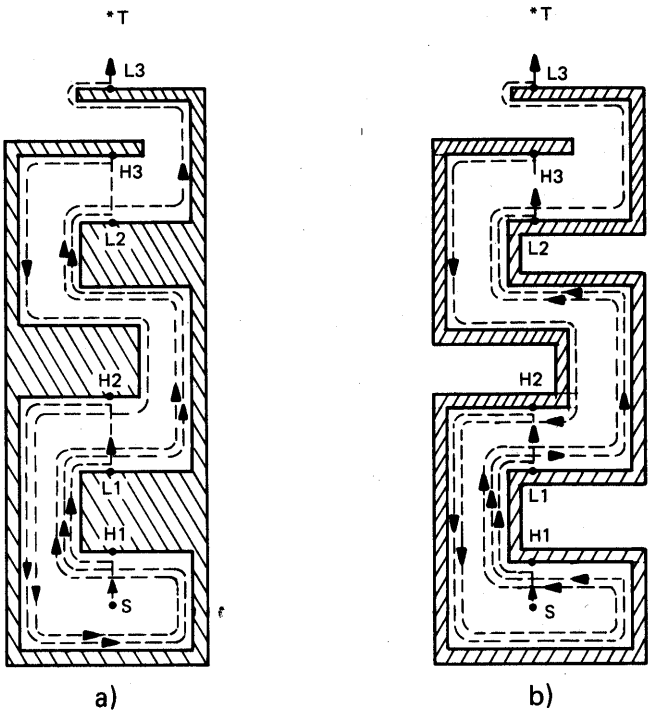

Figure 5. Automaton's path around a maze-like obstacle (in-obstacle position) under the algorithm Bug2. In terms of path complexity, both obstacles (a b) are the same, whereas for (a) the straight line (S,T) crosses the obstacle 10 times, $n_i = 10$, and for (b), $n_i = 16$. At most, the path passes one segment (here (H1,L1)) three times; that is, there are at most two local cycles.

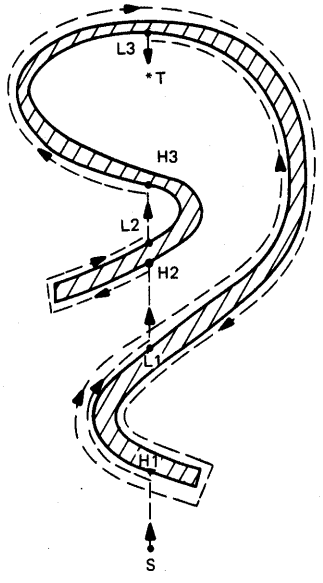

Figure 6. Automaton's path in case of an in-obstacle position; here S is outside the obstacle, and T is inside.

generated during "processing" of a single obstacle (see, for example, Figure 5). Also, notice that dependence between the perimeters of obstacles and the length of the path generated by Bug2 is not as clear as in the case of Bug1. For some scenes, Bug2 may create shorter paths compared to Bug1; often the path around an obstacle will be shorter than the obstacle perimeter (compare Figures 2 and 4). In some more unfortunate cases, when the straight line segment of the path meets the obstacle almost tangentially and the automaton goes around the obstacle in a "bad" direction, the path may be actually equal to the full perimeter of the obstacle (see Figure 7). Finally, as Figures 5 and 6 demonstrate, the situation may get even worse, and the automaton may have to pass along some segments of a maze-like obstacle even more than once.

Characteristics of Bug2.

For the analysis of performance of Bug2, we introduce additional definitions.

Definition 3. For a given local direction, a *local cycle* is created when the automaton has to pass some segment of the same obstacle *more than once*. In the example in Figure 7, no cycles are created; in Figures 5 and 6 there are some local cycles.

Definition 4. A case of an *in-obstacle* refers to such a mutual position of the pair of points (Start, Target) and a given obstacle where 1) the interval (Start,Target) of the corresponding straight line crosses the obstacle borderline at least once, and 2) either Start or Target lie inside the minimal convex hull of the obstacle. A case of *out-obstacle* refers to such a mutual position of the pair (Start, Target) and the obstacle in which both points Start and Target lie

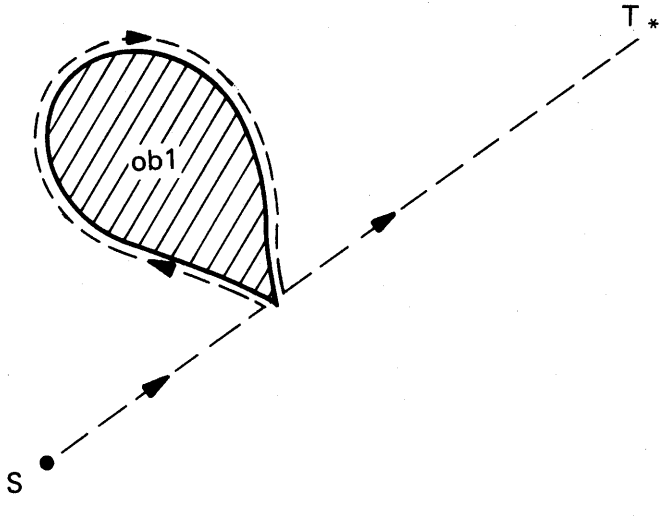

Figure 7. A case when, under the algorithm Bug2, the automaton will have to make almost a full circle around a convex obstacle.

outside the minimal convex hull of the obstacle. For example, in Figure 3 the pair (Start, Target) are located outside the obstacles ob1, ob2, and inside the obstacle ob3.

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 may still be as simple as for convex obstacles, if $n_i = 2$ (Figure 4, obstacle ob2). The situation may become more complicated if $n_i > 2$. In Figure 5a,b, one can see that, under Bug2, the segment of the borderline from H1 to L1, (H1,L1), will be passed three times; segments (L1,L2) and (H2,H1) — two times each, and segments (L2,L3) and (H3,H2) — one time each.

Lemma 3. Under Bug2, the automaton will pass any segment of the i-th obstacle borderline at most $n_i/2$ times.

The Lemma, therefore, guarantees that the procedure terminates, and gives a limit on the number of generated local cycles.

Proof. Assume that only one obstacle is present so that we can drop the index i. For each Hit point, H^j, the procedure will make the automaton walk around the obstacle until it reaches the corresponding Leave point, L^j; therefore, all H and L points appear in pairs, (H^j,L^j). Under the accepted model (finite"thickness" of obstacles), for each pair (H^j,L^j) an inequality holds: $d(H^j) > d(L^j)$. After leaving L^j, the automaton walks along a straight line to the next Hit point, H^{j+1}. Since, by the model, the distance between two segments of obstacle borderline is finite, then $d(L^j) > d(H^{j+1})$. This produces an inequality for all the H and L points,

$$d(H^1)>d(L^1)>d(H^2)> d(L^2)>d(H^3)>d(L^3)>\dots \quad (6)$$

Therefore, although each H (or L) point may be passed more than once, it will be defined as an H (or L) point only once; thus, it may generate only one new passing of the same segment of the obstacle perimeter. In other words, each pair (H^j,L^j) may give rise to only one passing of an obstacle borderline segment. Q.E.D.

Using this Lemma, we can now produce an upper bound for the length of the path generated by Bug2.

Theorem 3. The length of the path produced by the procedure Bug 2 will never exceed the limit

$$P = D + \Sigma\frac{n_i p_i}{2} \quad (7)$$

Proof. Any path may be looked at as consisting of two parts — straight line segments of the automaton's walk toward Target (between obstacles or between parts of the same obstacle) along the line (Start, Target), and segments related to walking around the obstacle borderline. Because of the inequality (6), the sum of straight line segments will never exceed D. As to the path segments around obstacles, Lemma 3 guarantees for each obstacle met by the automaton the upper bound—not more than $n_i/2$ walks along the same segment of the obstacle borderline. Summing up both the straight line walk estimate and estimates for walking around all obstacles, produces (7). Q.E.D.

The following theorem shows that the upper bound given by (7) is constructive, in the sense that there exist scenes for which generated paths will be as close to the upper bound (7) as one wishes. The theorem is formulated for one obstacle only; by placing obstacles one inside another, one can easily extend its results to any number of obstacles.

Theorem 4. For any $\epsilon>0$, there exist a scene with just one obstacle in it for which the length of the path produced by Bug2 will be equal to

$$P = D + (1 - \epsilon) \cdot \frac{np}{2} \quad (8)$$

Proof. To prove the theorem, it is enough to give an example of a scene for which the theorem holds. Notice that to reach the upper bound (7), the automaton must make a full circle around each obstacle maximum number of times as suggested by Lemma 3 — namely, $n_i/2$ times. In actuality, in its walk around an obstacle, the automaton will pass some segments of its perimeter less than $n_i/2$ (these are "good" parts of the path) and some other segments of the perimeter — exactly $n_i/2$ times ("bad" parts of the path). To satisfy (8), the "bad" parts of the path have to account for at least $(1-\epsilon)$ part of the total path length.

Consider an example shown in Figure 8. Under Bug2, the automaton will have to walk around the comb-like segments of the obstacle, (H1,L1), two times; according to Lemma 3, since $n=4$, then two is the maximum number of passing around a perimeter segment. One can see that, without changing any other details of the scene, the length of the segment (H1,L1) may be made arbitrarily long, and, in particular, such that for any specific $\epsilon>0$ the upper bound (8) is reached. Q.E.D.

So far as the performance of the algorithm Bug2 is concerned, Theorems 3 and 4 sound rather depressive; namely, they suggest that, under Bug2, the automaton sometimes may have to go around an obstacle any (large albeit finite) number of times. Because of this, an important question is: How typical are "bad" scenes, and, in particular, What characteristics of a scene influence the length of the path? The Theorem 5 and the following Corollary below address this question. They suggest that, indeed, only in rather special cases will generated paths be as long as estimated by (8). Theorem 5 states, in particular, that the mutual position of the Start point, Target point, and the obstacles is an important characteristics of the

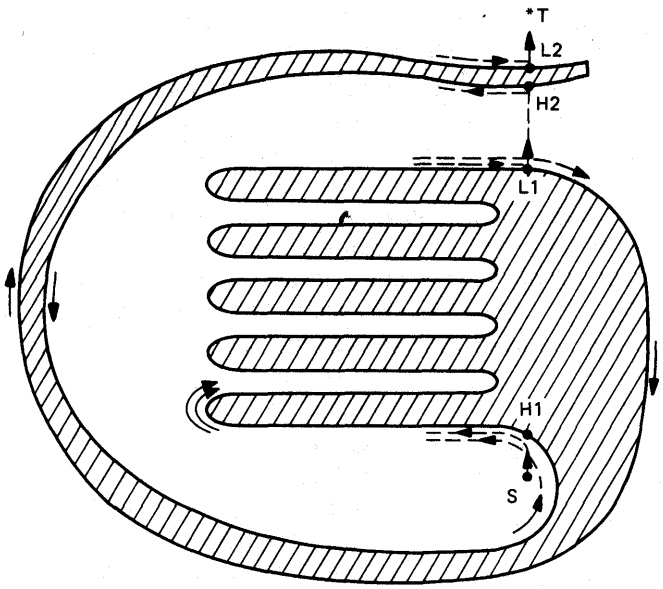

Figure 8. Illustration for Theorem 4. The part of the path corresponding to the comb-like segment (H1,L1), accounts for (1-ϵ) part of the total path length. This part will be passed the maximum number of times (here two times, according to Lemma 3).

scene influencing the path length. The Corollary states that for convex obstacles the upper bound on the length of the paths generated by Bug2 may be improved significantly.

Theorem 5. Under Bug2, in case of an out-obstacle, the automaton will pass the obstacle's perimeter *at most once*.

In other words, no matter how "bad" the obstacle is in itself, for an outside position (see the definition above) the estimate on the length of the path reaches its lower bound (1)!

Proof. Figure 9 illustrates the proof. Shaded areas in Figure 9 correspond to one or many obstacles. Dotted lines indicate that obstacle borderlines may be any. Consider the first point H the automaton defines on its way from Start to Target. The automaton reaches this point first along the straight line (Start,Target), then turns left and starts walking around the obstacle. To form a local cycle, the automaton has to return to the point H again. Since a point may be defined as an H point only once (see the proof for Lemma 3), the next time the automaton returns to the point H it will be approaching point H from the right (see Figure 9), along the obstacle borderline. Therefore, after having defined and left point H, in order to approach it again (this time from the right), the automaton has to cross somehow the straight line (Start,Target). In general, there may be two ways of crossing this straight line:

1) The crossing occurs outside the interval (Start,Target). This case corresponds to an in-obstacle position (see Definition 4 above)—either due to Start point or due to Target point. Theorem 5, therefore, does not apply.

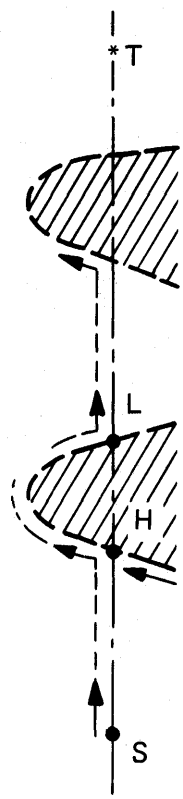

Figure 9. Illustration for Theorem 5.

2) Imagine that the crossing occurs inside the interval (Start,Target). Now we prove that such crossings of the path with the interval (Start,Target) may not result in local cycles. Notice that a crossing may not occur anywhere within the interval (Start,H) because otherwise H would not be the first H point defined. If a crossing occurs within the interval (H,Target), then at the point of crossing, the automaton, following the algorithm steps, defines a corresponding L point and starts moving along the line (Start,Target) toward Target until it defines the next H point; therefore, here it cannot reach into the right semiplane (see Figure 9). If, after this maneuver the automaton hits another part of an obstacle, the situation repeats itself. In other words, for any point within the interval (H,Target) the automaton will operate only in the left semiplane of the scene plane — this fact prevents it from having a local cycle. Q.E.D.

So far, no constraints on the shape of the obstacles have been imposed. If one assumes now that all of the obstacles are convex, then the upper bound for the length of the path may be improved as illustrated by the following statement.

Corollary. If all obstacles met by the automaton are convex, then the "average" length of the path produced by the procedure Bug2 is

$$P = D + 0.5 \cdot \Sigma p_i \qquad (9)$$

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

$$P = D + 1.0 \cdot \Sigma p_i \qquad (10)$$

Consider a statistically representative number of scenes with a random distribution of convex obstacles over each scene, a random distribution of points Start and Target over the set of scenes, and a fixed local direction as defined above. Then the straight line (Start,Target) will cross all the obstacles it meets in such a way that for some obstacles the automaton will have to walk around them while covering the bigger part of their perimeters (as with the obstacle ob1, Figure 4). For some other obstacles, the automaton will cover only a smaller part of their perimeters (as with the obstacle ob2, Figure 4). On the average, one would expect a path that satisfies (9). As for (10), Figure 7 presents an example of such a bad scene.

This Corollary assures, therefore, that for a wide range of scenes the length of paths generated by the algorithm Bug2, will approach the universal lower bound (1).

Test for Target Reachability

At this point, a simple test may be defined to check for existence of a path between points Start and Target. We will use the fact that the automaton may determine, store, and later recognize its own coordinates.

As Lemma 3 suggests, under Bug2 the automaton may pass the same point H^j of the same obstacle more than once; in other words, it may make a finite number p of local cycles, $p = 0,1,2,\ldots$. It follows from the inequality (6) that after having defined a point H^j, the automaton will never define this point again as an H or L point. Therefore, on each of the subsequent local cycles (if any), point H^j will be passed not along the straight line (Start,Target), but along the obstacle borderline. Every time after leaving point H^j, the automaton may expect one of the following possibilities:

a) it will not return again to H^j; this happens, for example, if the automaton leaves this obstacle altogether, or/and reaches the Target,

b) it will define at least the first two of the points $L^j, H^{j+1}, \ldots$ and then return to the point H^j to start a new local cycle,

c) it will come back to the point H^j without having defined on the previous cycle a point L^j; this may happen only if either the automaton or the Target are being trapped inside the current obstacle (see Figure 10). Now, the test for Target reachability will be formulated.

Test for Target Reachability If, on the p-th local cycle, $p = 0,1,\ldots$, after having defined a point H^j, the automaton returns to this point before it defines at least the first two out of the possible set of points $L^j, H^{j+1}, \ldots, H^k$, it means that the automaton has been *trapped* and, hence, that Target is not reachable.

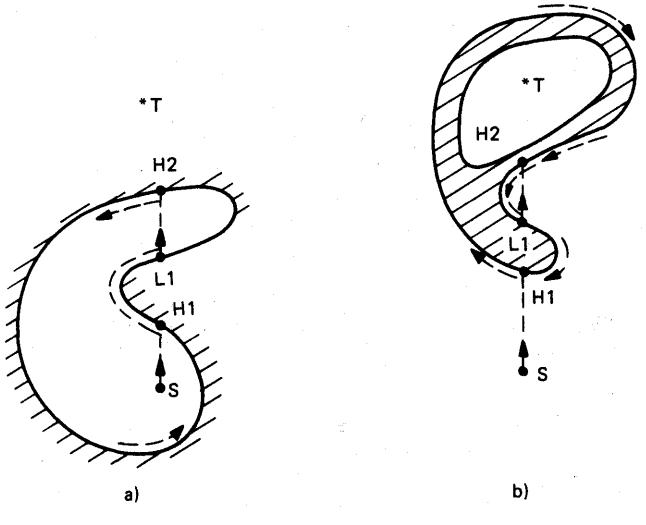

Figure 10. Examples of traps.
The path (dotted line) is executed under the algorithm Bug2. After having defined the point H2, the automaton returns to it before it defines any new L point. Therefore, the target is not reachable.

VI. IMPROVING PERFORMANCE OF BASIC ALGORITHMS

Basic Algorithms Bug1 and Bug2 have each a clear simple idea behind them. These ideas helped us prove theorems that define the upper and lower bounds on the length of generated paths. In the actual implementations of the algorithms, improvements can and should be introduced, which in many situations will make the paths shorter. Although the flow of action in such modified versions may be not as "clean" as in the Basic Algorithms, the bounds on the path length defined in the theorems above for the Basic Algorithms may be still applicable. Such a version, which actually combines the mechanisms of both Basic Algorithms, follows; it is called BugM1 (for "modified").

Although, in general, the algorithm Bug2 is quite efficient, in cases of in-obstacles, it may create local cycles (see Theorem 5). Bug2 will be modified in such a way that the number of local cycles will never be larger than two. In other words, the automaton will never pass the same segment of the obstacle borderline *more than three times.*

The procedure BugM1 is executed at any point of a continuous path. It effectively uses a straight line (L^j,Target) where the point Target is permanent, and the point L^j changes; j = 1,2,... . The procedure consists of the following steps.

Step 1. The automaton 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); alternatively, it may reach Target in which case it stops. L^o = Start; that is, the first Leave point coincides with Start.

Step 2. From H^j, the automaton begins moving along the obstacle (always using the accepted local direction) until it defines a point L^j. There may be two possibilities:

a) Moving along the obstacle borderline, the automaton crosses the straight line (L^{j-1}, Target) inside the interval (L^{j-1},Target); in this case the automaton defines a point L^j in such a way that it satisfies two requirements: 1) L^j is located on the straight line (L^{j-1},Target), and 2) 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) Moving along the obstacle borderline the automaton crosses the straight line (L^j, Target) outside the interval (L^j,Target); in this case the automaton defines a point J^j according to the *Steps 2 and 3 of the algorithm Bug1*. Go to Step 1.

Notice that if the scene is such that every time only Step 2a is 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 may be created in such situations.

As Theorem 5 suggests, local cycles appear in cases of in-obstacles when the condition accounted for in Step 2b of BugM1 is created. Having recognized a danger of multiple local cycles by the Step 2b condition being satisfied, the automaton, instead of risking an uncertain number of local cycles it may expect under Bug2 (see Lemma 3), "decides" to go to a more conservative but guaranteed upper bound (5) provided by the algorithm Bug1; 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 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 3).

With such a modification, the automaton will, in general, have the efficiency of Bug2 (in the sense that it does not have to necessarily walk around the full obstacle perimeter) while it will never pass the same segment of the obstacle borderline more than three times.

VII. SOME OBSERVATIONS ON PERFORMANCE OF BASIC ALGORITHMS

From the analysis above, one conclusion is that Basic Algorithms guarantee termination. Another conclusion is that in no way are the algorithms equivalent. Depending on the scene, one of them may produce a path significantly shorter than another. The question on when what algorithm should be used goes beyond a formal analysis. One could say, for example, that the algorithm Bug1 will, probably, appeal to a rather conservative (pessimistic) automaton, whereas the algorithm Bug2 may appeal to a more optimistic automaton.

If the automaton wants to minimize the effort (path length) for the worst scenes (a pessimistic automaton), Bug1 provides a guarantee that the path will never exceed the limit (5). Unfortunately, Bug1 will never produce a path as short as the one shown in Figure 4, but, on the other hand, it will never create local cycles.

However, if the automaton wants to minimize the effort on simple scenes, or if it has reasons to believe that the scene in question will not present any unpleasant surprises (an optimistic automaton) then it will use Bug2, which for convex and simpler non-convex obstacles, promises paths as short as given by (9).

Another reason for the optimistic automaton 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 (10) (which is better than (5) for Bug1) if the mutual position of Start, Target, and obstacles corresponds to a case of an out-obstacle.

And, of course, the algorithm BugM1 provides some reasonable compromise between the good and bad sides of Basic Algorithms.

One gets an additional insight into the operation and the "area of expertise" of the Basic Algorithms by trying to use them in maze search problems. There are a number of ways by which the problem of search in an unknown maze may be set. In one version (see, e.g., [16]) the automaton, started at any cell of the maze, must eventually visit every single cell without passing through any barriers (it 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. Thus, neither of the Basic Algorithms can be used.

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

In still another version of the maze search problem (see, e.g., [17]), the automaton 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 close to the problem considered in this paper.

Consider, for example, the behavior of the algorithm Bug2 in a maze search problem. As Theorem 3 states (and Figure 5 demonstrates), given an unfortunate scene, Bug2 may be rather inefficient and may create quite a few local cycles. On the other hand, as Theorem 5 ascertains, in many scenes Bug2 should behave rather efficiently. For a maze, the test on the in-obstacle condition relates not to the whole maze (as one may first think), but to the individual maze barriers which may, or may not, create in-obstacles.

This problem is demonstrated in Figure 11 on a randomly designed maze with Start and Target points thrown randomly in more or less opposite directions of the maze. (Since typically maze search algorithms — see, e.g., [16] — use discrete models, Figure 11 presents a discrete version

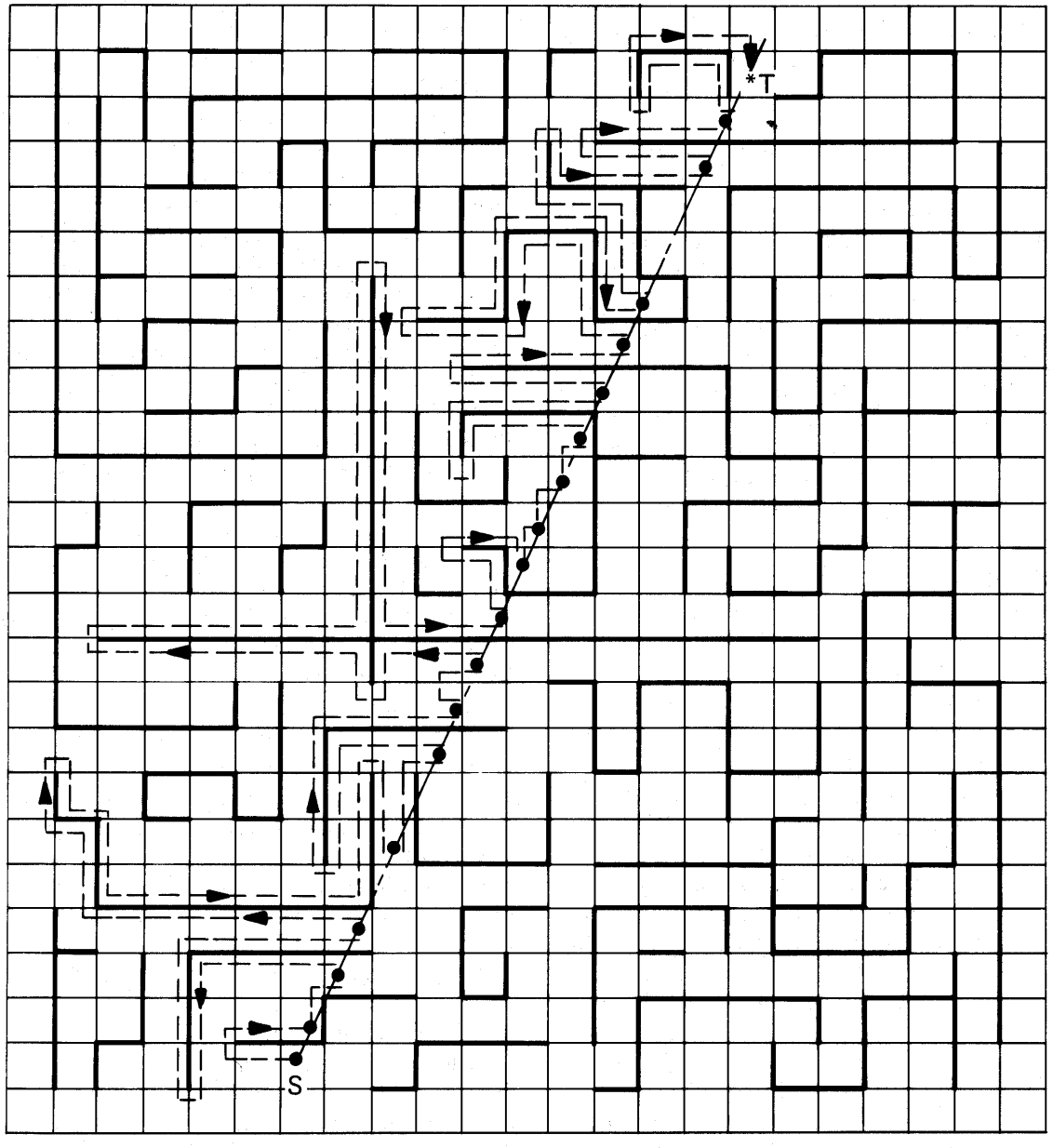

Figure 11. Example of a walk in a maze using Algorithm Bug2. (S = Start, T = Target.) Points in which the automaton's path (dotted line) crosses the imaginary straight line (S,T) are indicated by dots. Maze barriers are shown in thick lines.

of the continuous path planning problem; the automaton walks through cells presented by little squares; any cell touched by the straight line (S,T) is considered to be lying on this line). A quick look at the barriers between S and T suggests that the automaton is dealing here with the case of an out-obstacle; therefore, Theorem 5 should apply; no local cycles should be generated; and the generated path may be expected to be relatively short. Indeed, as Figure 11 demonstrates, this is the case. Given the fact that the automaton knows nothing about the design of the maze, it behaves rather reasonably.

REFERENCES

1. 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.
2. R.A. Brooks, R. Greiner, T.O. Binford, The ACRONYM Model-Based Vision System, Proceedings: IJCAI-6, Tokyo, Japan, August 1979.
3. B. Bullock, G.R. Edwards, D.M. Keirsey, D.Y. Tseng, F.M. Vilnrotter, D. Close, J.F. Bogdanowicz, E.P. Preyss, H.A. Parks, D.R. Partridge, Image Understanding Application Project: Implementation Progress Report. Proceedings of IEEE Computer Society Conference *"Trends and Applications, 1983: Automating Intelligent Behavior,"* Gaithersburg, Maryland, May 1983.
4. M.A. Lavin, Analysis of Scenes From a Moving Viewpoint. In *"Artificial Intelligence: an MIT Perspective,"* The MIT Press, Cambridge, 1980.
5. A.M. Thompson, The Navigation System of the JPL Robot. Proceedings of 5th Joint International Conf. on Artificial Intelligence, Cambridge, Massachusetts, August 1977.
6. J.T. Schwartz, M. Sharir, On the "Piano Movers" Problem. I. The Case of a Two-Dimensional Rigid Polygonal Body Moving Amidst Polygonal Barriers. Dept. of Computer Science, New York University, *Tech. Report No. 39,* October 1981.
7. H.P. Moravec, Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover, *Stanford AIM-340,* Sept. 1980.
8. T. Lozano-Perez, M. Wesley, An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles. *CACM 22* (1979).
9. 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.
10. T.O. Binford, Visual Perception by Computer. *IEEE Systems Science and Cybernetics Conference,* Miami, December 1971.
11. J. Reif, Complexity of the Mover's Problem and Generalizations. Proc. 20th Symposium of the Foundations of Computer Science, 1979.
12. J.T. Schwartz, M. Sharir, On the "Piano Movers" Problem.II. General Techniques for Computing Topological Properties of Real Algebraic Manifolds. Dept. of Computer Science, New York University, *Tech. Report No. 41,* February 1982.
13. R. Paul, Modelling Trajectory Calculation and Servoing of a Computer Controlled Arm. Ph.D Thesis, Stanford University, November 1972.
14. 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.
15. D.L. Pieper, The Kinematics of Manipulators Under Computer Control. PhD Thesis, Stanford University, October 1968.
16. 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.
17. W. Lipski, F.P. Preparata, Segments, Rectangles, Contours. *Journal of Algorithms,* 2, 1981.

NAVIGATION STRATEGIES FOR AN AUTONOMOUS VEHICLE WITH INCOMPLETE INFORMATION ON THE ENVIRONMENT

V. Lumelsky
A. Stepanov

Report No. 84CRD070
April 1984

GENERAL ELECTRIC COMPANY
CORPORATE RESEARCH AND DEVELOPMENT
P.O. BOX 8, SCHENECTADY, N.Y. 12301

GENERAL ELECTRIC