Algorithmica (1987) 2: 403-430

Algorithmica
© 1987 Springer-Verlag New York Inc.

Path-Planning Strategies for a Point Mobile Automaton Moving Amidst Unknown Obstacles of Arbitrary Shape[1]

Vladimir J. Lumelsky[2] and Alexander A. Stepanov[3]

Abstract. The problem of path planning for an automaton moving in a two-dimensional scene filled with unknown obstacles is considered. The automaton is presented as a point; obstacles can be of an arbitrary shape, with continuous boundaries and of finite size; no restriction on the size of the scene is imposed. The information available to the automaton is limited to its own current coordinates and those of the target position. Also, when the automaton hits an obstacle, this fact is detected by the automaton's "tactile sensor." This information is shown to be sufficient for reaching the target or concluding in finite time that the target cannot be reached. A worst-case lower bound on the length of paths generated by any algorithm operating within the framework of the accepted model is developed; the bound is expressed in terms of the perimeters of the obstacles met by the automaton in the scene. Algorithms that guarantee reaching the target (if the target is reachable), and tests for target reachability are presented. The efficiency of the algorithms is studied, and worst-case upper bounds on the length of generated paths are produced.

Key Words. Robotics, Robot motion planning, Collision avoidance algorithms, Planning with uncertainty, Provable algorithms.

1. Introduction.

To plan a path for a mobile automaton (a mobile robot) means finding a continuous trajectory leading from the initial position of the automaton to its target position. In this paper, we consider a case where the automaton is a point, and the environment (the scene) in which the automaton travels is defined in a two-dimensional plane. The scene can be filled with unknown obstacles of an arbitrary shape and size. The information about the obstacles comes from a simple sensor whose capability is limited to detecting an obstacle only when the automaton hits it. The main question being asked is whether, under such a model, a provable path-planning algorithm can be designed.

The current research on robot path planning can be classified into two large categories depending on which of the two following basic models is being used. In the first model, called *path planning with complete information* (another popular term is the *piano movers problem*), perfect information about the obstacles is assumed. In the second model, called *path planning with incomplete information*, an element of uncertainty about the environment is present. Another important distinction can be made between the provable (other terms—exact, algorithmic) and heuristic approaches. In these terms, this paper addresses the problem of designing provable path-planning algorithms in the framework of the model with incomplete information.

[1] Supported in part by the National Science Foundation Grant DMC-8519542.
[2] Yale University, New Haven, CT 06520, USA.
[3] Polytechnic University, Brooklyn, NY 11201, USA.

Received November, 1985; revised December 1, 1986. Communicated by Chee-keng Yap.

A brief review of the state of affairs in both categories of works is in order here; the review is by no means inclusive. The first model—the piano movers problem—is formulated as follows.[4] Given a solid object, in two- or three-dimensional space (2D or 3D), 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 path for the object from the initial position to the target position while avoiding collisions with obstacles along the way. It is assumed that the surfaces of the moving object and of the obstacles are algebraic; in most works, a stricter requirement of planar surfaces is imposed.

Because full information is assumed, the whole operation of path planning is a one-time, off-line operation. The main difficulty is not in proving that an algorithm that would guarantee a solution exists, but in obtaining a computationally efficient scheme. By a solution, we mean either reaching the target or concluding in finite time that the target cannot be reached. Conceptually, cases of arbitrary complexity can be considered, given the fact that a solution is always feasible. Another apparent advantage of dealing with complete information is that any optimization criteria (finding the shortest path, or the minimum-time path, or the safest path, etc.) can be easily introduced.

The computational complexity of the problem was first realized when Reif [6] showed that the general piano movers problem is PSPACE-hard; he also sketched a possible solution for moving a solid object in polynomial time, by direct computation of the "forbidden" volumes in spaces of higher dimensions.[5] Schwartz and Sharir [1] presented a polynomial-time algorithm for a two-dimensional piano movers problem with convex polygon obstacles. In a number of works (e.g., Lozano-Perez [2]), the solid object is viewed as shrinking to a point while the obstacles are viewed as expanding accordingly, to compensate for the shrinking object. The resulting *configuration space* has higher dimensionality compared with the original *workspace*—one extra dimension per each degree of rotational freedom. In general, the obstacles in the configuration space have nonplanar walls—even if the original obstacles are polyhedral. In order to keep the problem manageable, various constraints are typically imposed.

Moravec [3] considers a path-planning algorithm in two dimensions with the object presented as a circle. Brooks [4], in his treatment of a two-dimensional path-planning problem with a convex polygon object and convex polygon obstacles, uses a generalized cylinders presentation [5] to reduce 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 the cylinder axis (in general, a spine curve).

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. On a heuristic level, this version was started by Pieper [7] and then investigated by Paul [8] because

[4] A good survey of the work on provable algorithms for the problem can be found in [17].

[5] Higher dimensions d appear when one takes into account the orientation of the moving object along its way; $d = 3$ for the two-dimensional case, and $d = 6$ for the three-dimensional case.

of its obvious relation to path generation and coordinate transformation problems of multiple-degrees-of-freedom robot arms. Recently, new approaches for this version were considered in [9] and [10]. The most general algorithm (although very expensive computationally) for moving a free-hinged body was given by Schwartz and Sharir [9]; the technique is based on the general method of cell decomposition; the moving object and the obstacles are assumed to be limited by algebraic surfaces.

From the application standpoint, unless there is a reason to believe that the obstacles in the scene are polyhedral or, if the algorithm in question allows it, at least algebraic, the algorithms above cannot be applied directly. Then, an appropriate approximation has to be performed, which can introduce problems of its own. For example, given a prescribed accuracy of the approximation, the operation of approximating nonlinear surfaces with linear constraints itself requires exponential time [21]. Also, the space of possible approximations with constraints of a given order is not continuous in the approximation accuracy: in other words, a slight change in the specified accuracy of the approximation can cause a dramatic change in the approximated surfaces and eventually in the generated paths. On the other hand, the approximation itself depends on considerations that are secondary to the path-planning problem: these are, for example, the accuracy of the presentation of actual obstacles by polygons, or—a conflicting criterion—computational costs of processing the resulting connectivity graph.

The attractiveness of the model of *path planning with incomplete information* for robotics lies in the possibility of naturally introducing a powerful notion of feedback control, and thus transforming the operation of path planning into a continuous on-line process. In turn, using sensory feedback in the control scheme allows one to ease the requirements on the shape and location of the obstacles in the scene, and even lift the requirement that the obstacles be stationary. An important component of the model is the fact that the information about the environment has a local character—this is because at any given moment the robot sensors can provide information only about its immediate surroundings. This component changes the problem rather significantly. To start with, it is not clear whether an algorithm exists which would guarantee reaching a global goal (here, the robot target position) based on local means (the sensor information).

The question of reaching a global goal with local means presents a fundamental problem, various formulations of which have been studied in a number of areas: game theory (differential games and macroeconomics, e.g., [18]; collective behavior, e.g., [19]), computer science (maze search [14]), and studies in geometry [16]. The difficult question of relationship between uncertainty and the algorithm complexity has been studied in [20].

In the context of robot path planning, works related to the model with incomplete information have come primarily from studies on autonomous vehicle navigation; so far, they have been limited to various heuristics. In [11]-[13] a two-dimensional navigation problem is considered. Typically, obstacles are approximated by polygons; produced paths lie along the edges of the *connectivity graph* formed by the straight line segments connecting the obstacle vertices, the start point, and the target point, with a constraint on nonintersection of the graph edges with the 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 this limited area, the problem is actually treated as one with complete information.

One problem of working with incomplete information is that, because of the dynamic character of the incoming (sensor) information, the path cannot be preplanned, and so its global optimality is ruled out. Instead, one can judge the algorithm performance based on how it compares with other existing or theoretically feasible algorithms, or how optimal they are locally, or how "reasonable" they look from the human traveler standpoint.

Another inherent difficulty in designing algorithms for the model with incomplete information is that the problem dimensionality cannot be made arbitrarily high. Because the information about the obstacles is unknown, a natural way to limit the number of options available to the automaton at each step is to impose a constraint on the problem dimensionality. Otherwise, when an object meets an obstacle in the three-dimensional space, it has an infinite number of possibilities for passing around the obstacle. Hence the significance of the requirement introduced in this paper that the environment be limited to the two-dimensional plane (actually, our algorithms can be used on any surface homeomorphic to a plane). With this constraint, every time the automaton encounters an obstacle it can turn only left or right along the obstacle boundary. Essentially, the algorithms described below are exploiting the Jordan Curve Theorem, which states that any closed curve homeomorphic to a circle drawn around and in the vicinity of a given point on an orientable surface divides the surface into two separate domains, for which the curve is their common boundary [22]. Similar ideas can be used for designing algorithms for robot arm manipulators [23], [24].

In terms of the available information, the model considered in this paper can be viewed as being diametrically opposite to the piano movers model: instead of the full information about the obstacles assumed in the latter model, no information about the obstacles is given to the automaton in our model. The only available input information includes the automaton's own coordinates and those of the target. The automaton's capability for learning about an obstacle is limited to the "ultra-local" information provided by the automaton's "tactile sensor." In other words, the automaton learns about the presence of an obstacle only when it hits it.

Under the proposed algorithms, the automaton is continuously analyzing the incoming information about its current surroundings and is continuously planning its path. This is somewhat similar to the approach utilized in [16] for treating geometric phenomena based on local information. No approximation of the obstacles is done, and, consequently, no connectivity graphs or other intermediate computational structures are used. Since no reduction to a discrete space takes place, all points of the scene are available for the purpose of path planning. Because the model is continuous, the criteria typically used for evaluating algorithm performance—such as computational complexity as a function of the number of vertices of the obstacles, or the time or memory required—are not applicable. Instead, a new performance criterion based on the length of the generated paths as a function of the obstacle perimeters is introduced.

One point should be mentioned. The main goal of this work is to investigate how productive the model with incomplete information is for designing provable robot path-planning algorithms. Specifically, given the local nature of the incoming sensor information about the environment, is a global solution feasible? Furthermore, what are the minimum resources—the minimum knowledge and the minimum memory—that the automaton's model must assume in order to guarantee a solution? As a result of such narrow problem formulation, a number of important issues are left out. No learning of any kind is considered. Although the results we obtain are applicable to various types of sensor feedback, the presentation is limited to tactile sensing. The automaton's size and shape, which have to be taken into consideration in applications, are simply ignored in this work; a point automaton is considered.

In Section 2 the model of the environment and of the automaton is formulated. Assuming this model, a worst-case lower bound is produced in Section 3 for the general path-planning problem. In Sections 4 and 5 two *basic algorithms* for path planning are described and their convergence properties are analyzed. Each algorithm has quite different characteristics, and, depending on the scene, one can produce a shorter path than the other. For these algorithms, the worst-case upper bounds on the length of the generated paths are established, and tests for target reachability are formulated. This is followed, in Section 6, by an improved version of the path-planning algorithm which, while guaranteeing termination, combines good features of both basic algorithms without sacrificing much of their clarity. Finally, in Section 7, all three algorithms are compared, and an additional insight into the algorithms' mechanisms is provided by showing their relevance to the maze search problem.

2. Model.

The model includes two parts—one related to the geometry of the scene, and the other related to the characteristics and capabilities of the point mobile automaton (MA).

Environment. The scene is a plane with a set of obstacles and the points Start (S) and Target (T) in it. Each obstacle is a simple closed curve of finite length such that a straight line will cross it only in finitely many points; a case when the straight line coincides with a finite segment of the obstacle boundary is not a "crossing." (An equivalent term used in the text for a simple closed curve is the *obstacle boundary.*) Obstacles do not touch each other; that is, a point on an obstacle belongs to one, and only one, obstacle. A scene can contain only a locally finite number of obstacles; this means that any disc of finite radius intersects a finite set of obstacles. Note that the model does not require that the set of obstacles is finite.

Automaton. MA is a point; this means that an opening of any size between two distinct obstacles is considered to be passable. The only information MA is provided with by its sensors is (1) its current coordinates, and (2) the fact of contacting an obstacle. MA is also given the coordinates of the Target. Thus, it can always calculate its direction toward and its distance from the Target. The memory available for storing data or intermediate results is limited to a few computer words. The motion capabilities of MA include three possible actions: move toward the Target on a straight line; move along the obstacle boundary; stop.

DEFINITION 1. A *local direction* is a once and for all decided direction for passing around an obstacle. For the two-dimensional problem, it can be either left or right.

Because of the uncertainty involved, every time MA meets an obstacle, there is no information or criteria which 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 MA is always *left* (as in Figure 4). Unless stated otherwise, MA will be assumed to follow the local direction while walking around obstacles.

DEFINITION 2. MA is said to *define a hit point H* on the obstacle, when, while moving along a straight line toward the Target, MA contacts the obstacle at the point H. It *defines a leave point L* on the obstacle, when it leaves the obstacle at the point L in order to continue its straight line walk toward the Target. (See, for example, Figure 4.)

If MA moves along a straight line toward the Target and the line touches some obstacle tangentially then there is no need to invoke the procedure for walking around the obstacle—MA just continues its straight line walk toward the Target. In other words, no H or L points will be defined in this case. Because of that, no point of an obstacle can be defined as both an H and an L point. In order to define an H or an L point, the corresponding straight line has to produce a "real" crossing of the obstacle; that is, in the vicinity of the crossing, a finite segment of the line should lie inside the obstacle, and a finite segment of it should lie outside the obstacle.

Throughout, the following notation is used:

D is the (Euclidean) distance from the Start to the Target;

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

$d(A)$ is used as a shorthand notation for $d(A, \text{Target})$;

$d(A_i)$ signifies the fact that the point A is located on the boundary of the ith obstacle met by MA on its way to the Target;

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

p_i is the perimeter of the ith obstacle met by MA.

The performance of the path-planning algorithms will be evaluated based on the quantity $\sum_i p_i$, the sum of perimeters of obstacles met by MA on its way to the Target, or of obstacles contained in a specific area of the scene. This quantity will allow us to compare various path-planning procedures in terms of the length of the paths they produce.

3. The Lower Bound for the Path-Planning Problem.

This lower bound, formulated in Theorem 1 below, determines what performance can be expected in the worst case from any path-planning algorithm operating within the framework of our model. The bound is formulated in terms of the length of the path generated by the automaton on its way from the point Start (S) to the point Target (T). The bound is a powerful means for measuring performance of various path-planning procedures.

THEOREM 1. *For any path-planning algorithm satisfying the assumptions of our model, any (however large) $P > 0$, any (however small) $D > 0$, and any (however small) $\delta > 0$, there exists a scene for which the algorithm will generate a path of length P, and*

$$P \geq D + \sum_i p_i - \delta, \tag{1}$$

where D is the distance between the points Start and Target, and p_i are perimeters of the obstacles intersecting the disc of radius D centered at the Target.

PROOF. We want to prove that for any given algorithm a scene can be designed for which the length of the path generated by this unknown algorithm—say, Algorithm X—will satisfy (1). Algorithm X can be of any type: it can be deterministic or random; its intermediate steps may or may not depend on intermediate results; etc. The only information known about Algorithm X is that it operates within the framework of our model of the automaton MA and the environment (Section 2). The proof consists of designing a special scene (a set of obstacles) and then proving that the scene will force Algorithm X to generate a path not shorter than P in (1).

The following scheme, consisting of two stages, is suggested for designing the required scene (called the *resultant scene*). At the first stage, a *virtual obstacle* is introduced; this is an obstacle parts or all of which, but not more, will eventually produce, once the second stage is completed, the *actual obstacle(s)* of the resultant scene.

Consider a virtual obstacle shown in Figure 1(a). It presents a corridor, of finite width $2W > \delta$, and of finite length L. One end of the corridor is closed. The corridor is positioned such that the point S is located at the middle point of the closed end; the corridor opens in the direction opposite to the line (S, T). The thickness of the corridor walls is negligible compared with δ. Still in the first stage, MA is let walk from S to T along the path prescribed by Algorithm X. On its way, MA may or may not touch the virtual obstacle.

When the path is complete, the second stage starts. A segment of the virtual obstacle is said to be *actualized* if all points of the inside wall of the segment have been touched by MA. If MA touched the inside wall of the virtual obstacle at some length l, then the actualized segment is exactly of length l. If MA was continuously touching the virtual obstacle at a point and then bouncing back, the corresponding actualized area is considered to be a segment of length δ

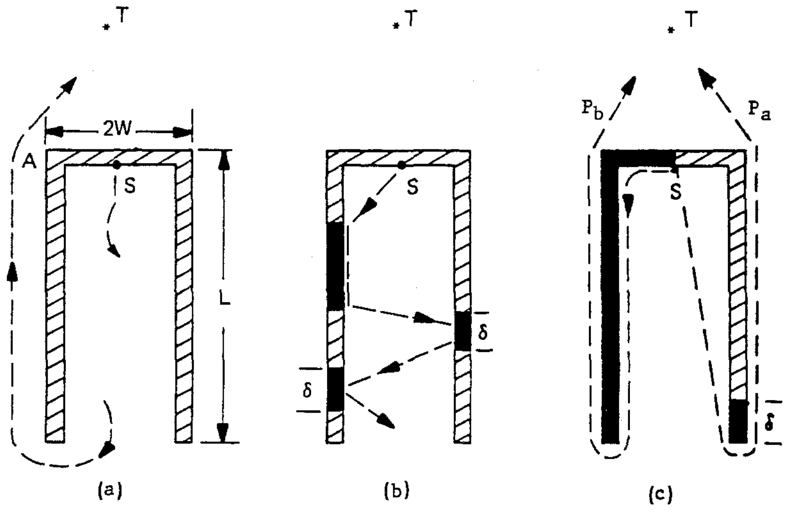

Fig. 1. Illustration for Theorem 1. Actualized segments of the maximum obstacle are shown as solid (S, Start point, T, Target point).

around the point of contact. If two segments of MA's path along the virtual obstacle are separated by an area of the virtual obstacle which MA did not touch, then MA is said to have actualized two separate segments of the virtual obstacle.

We produce the resultant scene by designating as actual obstacles only those areas of the virtual obstacle which have been actualized. Thus, if an actualized segment is of length l, then the perimeter of the corresponding actual obstacle is equal to $2l$; this takes into account the inside and the outside walls of the segment, and also the fact that the thickness of the wall is negligible.

This method for producing the resultant scene can be justified by the fact that, under the accepted model, the behavior of MA is affected only by those obstacles which it touches along its way. Indeed, the produced path could have appeared, under Algorithm X, in two different scenes: in the scene with the virtual obstacle, and in the resultant scene. One can argue, therefore, that the areas of the virtual obstacle, which MA did not touch along its way, might never have existed, and Algorithm X produced its path not in the scene with the virtual obstacle but in the resultant scene. This means that the performance of MA in the resultant scene can be judged against (1). This completes the design of the scene. Note that depending on MA's behavior under Algorithm X, zero, one, or more actual obstacles can be created in the scene of Figure 1.

Next, we have to prove that MA's path in the resultant scene satisfies (1). Since MA starts at a distance $D = d(S, T)$ from T, it obviously cannot avoid the term D in (1). We concentrate, then, on the second term in (1). One can see by now that the main idea behind the described process of designing the resultant scene is to force MA to generate, for each actual obstacle, a segment of the path at least as long as the total length of the boundary of that obstacle. Note that this characteristic of the path is independent of Algorithm X.

MA's path in the scene can be divided into two parts, $P1$ and $P2$; $P1$ corresponds to MA's traveling inside the corridor, and $P2$ corresponds to its traveling outside the corridor; the same notation is used to indicate the length of the corresponding part. Both parts can become intermixed since, after having left the corridor, MA can temporarily return into it. Since part $P2$ starts at the exit point of the corridor then

$$P2 \geq L + C, \tag{2}$$

where $C = \sqrt{D^2 + W^2}$ is the hypotenuse AT of the triangle ATS (Figure 1(a)). As for part $P1$ of the path inside the corridor, it can be, depending on Algorithm X, any curve. Observe that in order to defeat the bound (1), Algorithm X has to decrease the "path per obstacle" ratio as much as possible. What is important for the proof is that, from the "path per obstacle" standpoint, every segment of $P1$ that does not result in the creation of the equivalent segment of an actualized obstacle makes the path worse. All possible alternatives for $P1$ can be clustered to three groups. These groups are discussed separately below.

1. Part $P1$ of the path never touches the walls of the virtual obstacle (Figure 1(a)). As a result, no actual obstacles will be created, $\sum_i p_i = 0$; but, as one can see, the resulting path is $P > D$. Therefore, for this kind of Algorithm X the theorem holds. Moreover, at the final evaluation, where only actual obstacles count, this MA strategy will not be judged as being very efficient: it creates an additional path component at least equal to $(2 \cdot L + (C - D))$—in a scene with no obstacles!
2. MA touches one or both inside walls of the virtual obstacle more than once (Figure 1(b)). In other words, between the consecutive touchings MA is temporarily "out of touch" with the virtual obstacle. As a result, part $P1$ of the path will produce a number of disconnected actual obstacles; the smallest of these, of length δ, correspond to point touchings. Observe that, in terms of the "path per obstacle" assessment, this kind of strategy is not very wise either. First, for each actual obstacle, a segment of the path at least as long as the obstacle perimeter is created; besides, additional segments of $P1$, due to traveling between the actual obstacles, are produced. Each of these additional segments is at least not smaller than $2W$, if the two consecutive touchings correspond to the opposite walls of the virtual obstacle, or at least not smaller than the distance between two sequentially visited disconnected actual obstacles on the same wall. Thus, the length P of the path exceeds the right side in (1), and the theorem holds.
3. MA touches the inside walls of the virtual obstacle at most once. This case includes various possibilities, from a point touching, which creates a single actual obstacle of length δ, to the case when MA closely follows the inside wall of the virtual obstacle. As one can see in Figure 1(c), this case contains the most interesting paths. The shortest possible path would be created if MA went directly from S to the furthest point of the virtual obstacle and then directly to T (path P_a, Figure 1(c)). (Given the fact that MA knows nothing

about the obstacles, this kind of a path can be produced only by accident.) The total perimeter of the obstacle(s) here is 2δ, and the theorem clearly holds.

Finally, the most efficient path, from the "path per obstacle" standpoint, is produced if MA closely follows the inside wall of the virtual obstacle, and then goes directly to the point T (path P_b, Figure 1(c)). Here MA is doing its best in trying to compensate each segment of the path with an equivalent segment of the actual obstacle. In this case, the generated path P is equal to

$$P = \sum_i p_i + \sqrt{D^2 + W^2} - W \tag{3}$$

(in our case, there is only one term in $\sum_i p_i$). Since no constraints have been imposed on the choice of the lengths D and W, take them such that

$$\delta \geq D + W - \sqrt{D^2 + W^2} \tag{4}$$

which is always possible because the right-hand side in (4) is nonnegative for any D and W. Reverse the sign (and the inequality) in (4), and add $(D + \sum_i p_i)$ to both its sides; this produces

$$\sum_i p_i + \sqrt{D^2 + W^2} - W \geq D + \sum_i p_i - \delta. \tag{5}$$

Comparing (3) with (5), observe that (1) is satisfied. This exhausts all possible cases of path generation by Algorithm X. □

We conclude this section with two remarks. First, by appropriately selecting multiple virtual obstacles, Theorem 1 can be extended to an arbitrary number of obstacles. Second, for the lower bound to hold, the constraints on the information available to MA can be relaxed significantly. Namely, the only required constraint is that MA does not have complete information about the scene at any time.

In the following sections, three path-planning algorithms are introduced, their performances analyzed, and the upper bounds on the length of their generated paths derived.

4. First Basic Algorithm: Bug1

4.1. Procedure. The procedure Bug1 is to be executed at any point of a continuous path. Figure 2 demonstrates the behaviour of MA. The goal is to generate a path from the Start to the Target. When meeting an ith obstacle, MA defines a hit point H_i, $i = 1, 2, \ldots$. When leaving the ith obstacle, to continue its travel toward the Target, MA defines a leave point L_i; initially, $i = 1$; $L_0 = \text{Start}$. The procedure uses three registers, R_1, R_2, R_3, to store intermediate information; all three are reset to zero when a new hit point, H_i, is defined. Specifically, R_1 is used to store the coordinates of the current point, Q_m, of the minimum distance

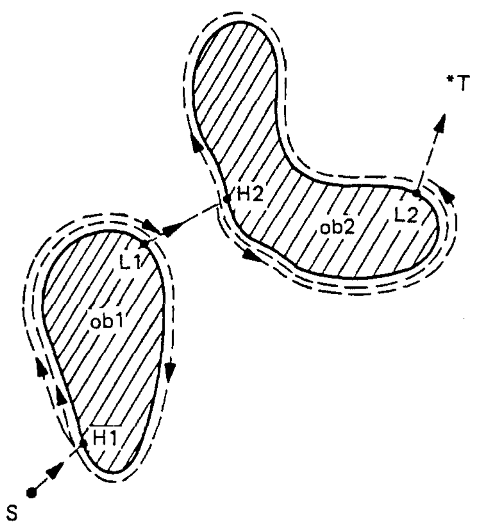

Fig. 2. Automaton's path (dotted lines), Algorithm Bug1 (*ob*1, *ob*2, obstacles; *H*1, *H*2, hit points; *L*1, *L*2, leave points).

between the obstacle boundary and the Target (this takes one comparison at each path point); R_2 integrates the length of the obstacle boundary starting at H_i; and R_3 integrates the length of the obstacle boundary starting at Q_m. (In case of many choices for Q_m, any one of them can be taken.) The test for target reachability mentioned in step 3 of the procedure is explained in Section 4.3. The procedure consists of the following steps:

1. From the point L_{i-1}, move toward the Target along a straight line until one of the following occurs:
   (a) The Target is reached. The procedure stops.
   (b) An obstacle is encountered and a hit point, H_i, is defined. Go to step 2.
2. Using the local direction, follow the obstacle boundary. If the Target is reached, stop. After having traversed the whole boundary and having returned to H_i, define a new leave point $L_i = Q_m$. Go to step 3.
3. Apply the test for target reachability. If the Target is not reachable, the procedure stops. Otherwise, using the content of the registers R_2 and R_3, determine the shorter way along the boundary to L_i, and use it to get to L_i; set $i = i + 1$ and go to step 1.

*4.2. Characteristics of Bug*1. In this section the characteristics and performance of the algorithm are analyzed.

LEMMA 1. *When, under Bug*1, MA *leaves a leave point of an obstacle in order to continue its way toward the Target, it never returns to this obstacle again.*

Proof. Assume that on its way from the Start to the Target MA does meet some obstacles. We number those obstacles in the order in which MA meets them. Then, the 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 the point Start happened to be on the boundary of an obstacle and the line (Start, Target) crosses that obstacle then $D = d(H_1)$.

It has been mentioned in Section 2 that if MA's path touches an obstacle tangentially then there is no need to invoke the procedure for walking around an obstacle—MA just continues its straight line walk toward the Target. In all other cases of meeting the ith obstacle, unless the Target lies on the boundary of the obstacle, a relation holds: $d(H_i) > d(L_i)$. This is because, on the one hand, according to the model, any straight line (except a line that touches the obstacle tangentially) crosses the obstacle at least in two distinct points (finite "thickness" of obstacles), and, on the other hand, according to Algorithm Bug1, the point L_i is the closest point from the obstacle to the Target. Starting from L_i, MA walks straight to the Target until it meets the $(i+1)$th obstacle. Since, according to the model, obstacles do not touch one another, then $d(L_i) > d(H_{i+1})$. Therefore, our sequence of distances satisfies the relation,

$$(6) \qquad d(H_1) > d(L_1) > d(H_2) > d(L_2) > d(H_3) > d(L_3) > \cdots,$$

where $d(H_1)$ is or is not equal to D. Since $d(L_i)$ is the shortest distance from the ith obstacle to the Target, and since (6) guarantees that Algorithm Bug1 monotonically decreases the distances $d(H_i)$ and $d(L_i)$ to the Target, Lemma 1 follows. □

The lemma thus guarantees that the strategy will never create cycles.

Corollary. *Under Bug*1, *independent of the geometry of an obstacle,* MA *defines not more than one hit and not more than one leave point on it.*

To produce an upper bound on the length of the paths generated by Bug1, an assurance is needed that on its way to the Target MA always encounters only a finite number of obstacles. This is not obvious since, while following Algorithm Bug1, MA can "look" at the Target not only from different distances but also from different directions; that is, besides moving toward the Target, it may also rotate around the Target (see Figure 3). Hence the following lemma.

Lemma 2. *Under Bug*1, *on its way to the Target* MA *can meet only a finite number of obstacles.*

Proof. Although, while walking around an obstacle, MA can, at some moments, be at distances much larger than D from the Target (see Figure 3), the straight line segments of its path toward the Target are always within the same circle of radius D centered at the Target; this is guaranteed by inequality (6). Since, according to our model, any disc of finite radius can intersect with only a finite number of obstacles, the lemma follows. □

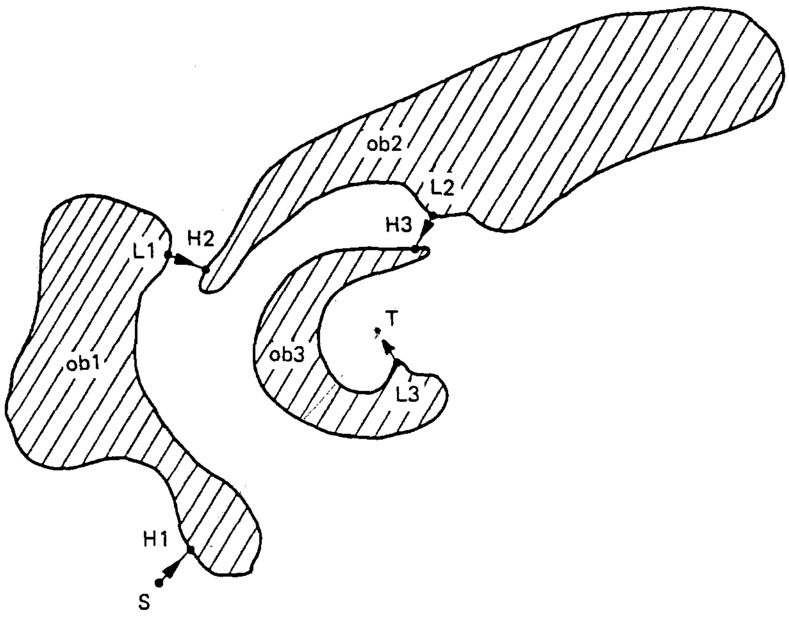

Fig. 3. Algorithm Bug1. Arrow lines indicate straight line segments of the automaton's path. Segments around obstacles are not shown; these are similar to the ones shown in Figure 2.

COROLLARY. *The only obstacles that can be met by* MA (*under Algorithm Bug*1) *are those which intersect the disc of radius D centered at the Target.*

Together, Lemma 1, Lemma 2, and the corollary guarantee convergence of Algorithm Bug1.

At this point we can establish the performance of the procedure Bug1, in terms of the length of the paths it generates. The following theorem gives an upper bound on the path lengths produced by Bug1.

THEOREM 2. *The length of the path produced by the procedure Bug*1 *will never exceed the limit*

$$P = D + 1.5 \cdot \sum_i p_i, \tag{7}$$

where $\sum_i p_i$ refer to the perimeters of the obstacles intersecting the disc of radius D centered at the Target.

PROOF. Any path can be looked at as consisting of two parts: straight line segments of the path between the obstacles, and the path segments related to walking around the obstacles. Due to inequality (6), the sum of the straight line segments will never exceed D. As to the path segments around the obstacles, Algorithm Bug1 requires that, in order to define a leave point on the ith obstacle, MA has to make a full circle around it; this produces a path segment equal to one perimeter, p_i, of the ith obstacle. By the time MA is prepared to walk from the hit to the leave point, in order to depart for the Target, it knows the direction (go left or go right) of the shorter path to the leave point. Thus, its path segment between the hit and the leave points along the boundary of the ith obstacle will not exceed $0.5 \cdot p_i$. Summing up the partial estimates for the straight line segments of the path and for the segments around the obstacles met by MA on its way to the Target, we obtain (7). □

Analysis of the procedure Bug1 shows that the requirement that MA has to know its own coordinates at any instance can be eased. It suffices if MA is capable of positioning itself at the circle of a given radius centered at the Target. In other words, what is important for Bug1 is not the actual position of MA but its direction toward and its distance from the Target. Assume that instead of the coordinates of the current point Q_m of the minimum distance between the obstacle and the Target, we store in the register R_1 the minimum distance itself. Then, in step 3 of Bug1, MA can reach the point Q_m by comparing its current distance from the Target with the content of the register R_1. If more than one point of the current obstacle lie at the minimum distance from the Target, any one of them can be used as the leave point, without affecting the convergence of the procedure.

4.3. Test for Target Reachability. Every time MA completes the exploration of a new obstacle i, it defines on it a point L_i. Then, MA leaves the ith obstacle (according to Lemma 1, it will never return to it) and starts moving from L_i, to the Target along the straight line segment (L_i, Target). Since the point L_i is by definition the closest point of the ith obstacle to the Target, then normally there should be no points of the ith obstacle between L_i and the Target. Because of the model assumption that the obstacles do not touch each other, the point L_i cannot belong to any other obstacle but i. Therefore, if MA, after having arrived at L_i in step 3 of the algorithm, discovers that the straight line (L_i, Target) crosses some obstacle at point L_i, this can only mean that the crossed obstacle is i and that the Target is not reachable—either the Start or the Target point is *trapped* inside the ith obstacle.

To show that this is true, let O be a simple closed curve, X be a point in the scene not belonging to O, L the point on O closest to X, and (L, X) the straight line segment connecting L and X; all of these are defined in the plane. The segment (L, X) is said to be *directed outward* if a finite part of it in the vicinity of the point L is located outside of the curve O; otherwise, if (L, X) penetrates inside the curve O in the vicinity of L, it is said to be *directed inward*.

The following statement holds: if the segment (L, X) is directed inward then X is inside O. The condition is necessary because if X were outside the curve O then some other point of O would appear in the intersection of (L, X) and O which would be closer to X than L; by definition of the point L, this is impossible. The condition is also sufficient because if (L, X) is directed inward and L is the point of the curve O closest to X then (L, X) cannot cross any other point of O and, therefore, X must lie inside O. This fact is used in the following test.

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

5. Second Basic Algorithm: Bug2

5.1. Procedure. The procedure Bug2 is executed at any point of a continuous path. Again, the goal is to generate a path from the Start to the Target. As will be seen, in its travel under Bug2, on the one hand, MA can meet the same obstacle i more than once, but, on the other hand, the algorithm has no way of distinguishing between different obstacles. Because of that, the subscript i will be used only when referring to more than one obstacle; in addition, the superscript j will be used to indicate the jth occurrence of the hit or leave points on the same or on a different obstacle. Initially, $j = 1$; $L^0 =$ Start. The test for target reachability built into steps 2(b) and 2(c) of the procedure is explained in Section 5.3. One can follow the procedure using the example shown in Figure 4. The algorithm consists of the following steps:

1. From the point L^{j-1}, move along the straight line (Start, Target) until one of the following occurs:
   (a) The Target is reached. The procedure stops.
   (b) An obstacle is encountered and a hit point, H^j, is defined. Go to step 2.
2. Using the accepted local direction, follow the obstacle boundary until one of the following occurs:
   (a) The Target is reached. The procedure stops.

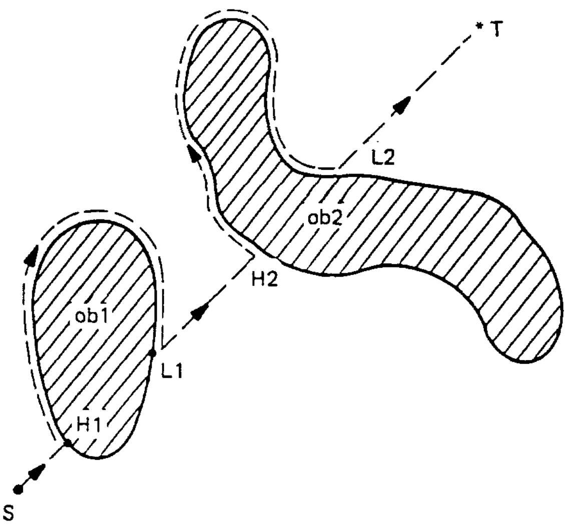

Fig. 4. Automaton's path (dotted line) under Algorithm Bug2.

(b) The line (Start, Target) is met at a point Q such that the distance $d(Q) < d(H^j)$, and the line (Q, Target) does not cross the current obstacle at the point Q. Define the leave point $L^j = Q$. Set $j = j + 1$. Go to step 1.

(c) The automaton returns to H^j and thus completes a closed curve (the obstacle boundary) without having defined the next hit point, H^{j+1}. The target is trapped and cannot be reached. The procedure stops.

Unlike Algorithm Bug1, more than one hit and more than one leave point can be generated on a single obstacle (see, for example, Figure 5). Also, note that the relationship between the perimeters of the obstacles and the length of the paths generated by Bug2 is not as clear as in the case of Bug1. In Bug1 the perimeter of an obstacle met by MA is covered at least once, and never more than 1.5 times. In Bug2, however, more options appear. A path segment around an obstacle generated by MA is sometimes shorter than the obstacle perimeter (compare Figures 2 and 4). In some other cases, when a straight line segment of

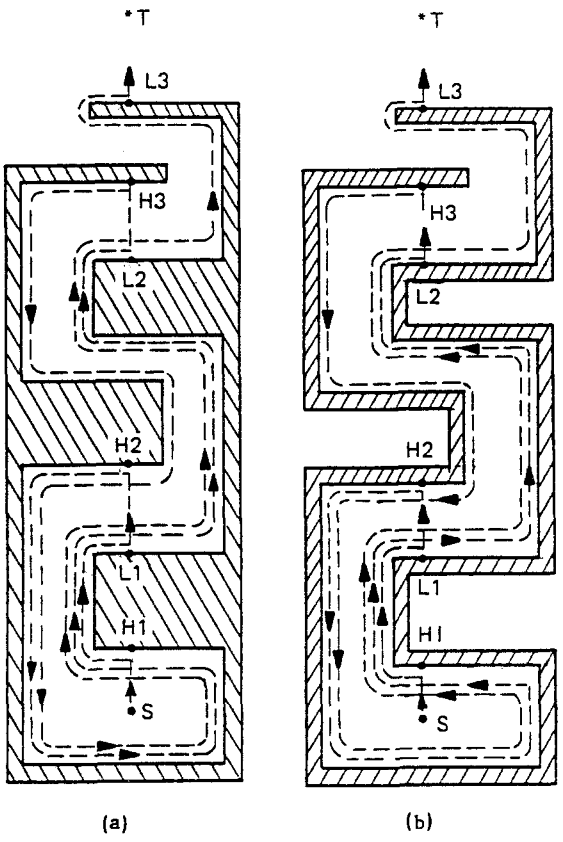

Fig. 5. Automaton's path around a maze-like obstacle (in-position case) under Algorithm Bug2. In terms of path complexity, both obstacles (a) and (b) are the same, whereas for (a) the straight line (S, T) crosses the obstacle 10 times, $n_i = 10$, and for (b), $n_i = 16$. At most, the path passes one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

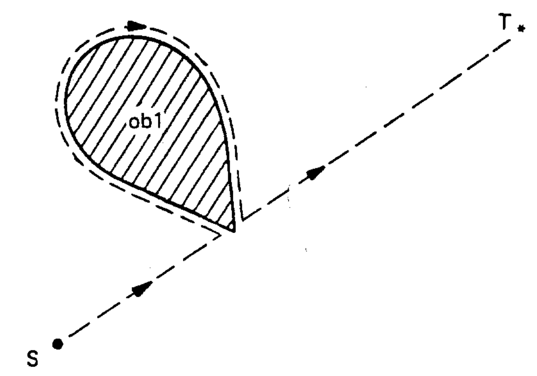

Fig. 6. A case when, under Algorithm Bug2, the automaton will have to make almost a full circle around a convex obstacle.

the path meets the obstacle almost tangentially and MA goes around the obstacle in a "wrong" direction, the path can actually be equal to the obstacle's full perimeter (Figure 6). And finally, as Figures 5 and 7 demonstrate, the situation can get even worse, and MA may have to pass along some segments of a maze-like obstacle more than once (see more on this case in the next section).

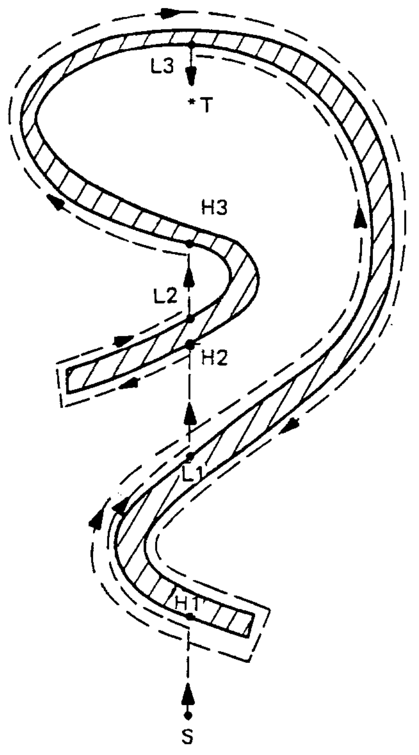

Fig. 7. Automaton's path in case of an in-position scene; here S is outside the obstacle, and T is inside.

5.2. Characteristics of Bug2

LEMMA 3. *Under Bug2, on its way to the Target* MA *can meet only a finite number of obstacles.*

PROOF. Although, while walking around an obstacle, MA can, at some moments, be at distances much larger than D from the Target, its straight line segments toward the Target are always within the same circle of radius D centered at the Target. This is guaranteed by the algorithm condition that $d(L^j, \text{Target}) > d(H^j, \text{Target})$ (see step 2 of Algorithm Bug2). Since, according to our model, any disc of finite radius can intersect with only a finite number of obstacles, the lemma follows. □

COROLLARY. *The only obstacles that can be met by* MA *under Algorithm Bug*1 *are those which intersect the disc of radius D centered at the Target. Moreover, the only obstacles that can be met by* MA *are those that intersect the straight line (Start, Target).*

DEFINITION 3. For a given local direction, a *local cycle* is created when MA has to pass some point of its path more than once.

In the example in Figure 4, no cycles are created; in Figures 5 and 7, there are some local cycles.

DEFINITION 4. A term *in-position* refers to such a mutual position of the pair of points (Start, Target) and a given obstacle where (1) the straight line segment (Start, Target) crosses the obstacle boundary at least once, and (2) either the Start or the Target lie inside the convex hull of the obstacle. A term *out-position* refers to such a mutual position of the pair (Start, Target) and the obstacle in which both points Start and Target lie outside the convex hull of the obstacle. A given scene is referred to as an in-position case if at least one obstacle in the scene, together with the Start and the Target points, creates an in-position condition; otherwise, the scene present an out-position case.

For example, the scene in Figure 3 presents an in-position case; without the obstacle *ob*3, it would have presented an out-position case. Below, n_i is the number of intersections between the straight line (Start, Target) and the ith 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 but still $n_i = 2$, the path generated by Bug2 can be as simple as that for a convex obstacle (Figure 4, obstacle *ob*2). It can become more complicated if $n_i > 2$. In Figure 5(a) and (b), the segment of the boundary 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.

LEMMA 4. *Under Bug*2, MA *will pass any point of the* ith *obstacle boundary at most* $n_i/2$ *times.*

Proof. As one can see, the procedure Bug2 does not distinguish whether two consecutive obstacle crossings by the straight line (Start, Target) correspond to the same or to different obstacles. Without loss of generality, assume that only one obstacle is present; then, the index i can be dropped. For each hit point, H^j, the procedure will make MA 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). Because, under the accepted model, obstacles are of finite "thickness," for each pair (H^j, L^j) an inequality holds: $d(H^j) > d(L^j)$. After leaving L^j, MA walks along a straight line to the next hit point, H^{j+1}. Since, according to the model, the distance between two crossings of the obstacle by a straight line is finite then $d(L^j) > d(H^{j+1})$. This produces an inequality for all the H and L points,

$$d(H^1) > d(L^1) > d(H^2) > d(L^2) > d(H^3) > d(L^3) > \cdots. \tag{8}$$

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

The lemma guarantees that the procedure terminates, and gives a limit on the number of generated local cycles. Using the lemma, an upper bound on the length of the paths generated by Bug2 can be produced.

Theorem 3. *The length of a path generated by the procedure Bug2 never exceeds the limit*

$$P = D + \sum_i \frac{n_i p_i}{2}, \tag{9}$$

where p_i refer to the perimeters of the obstacles intersecting the straight line segment (Start, Target).

Proof. Any path can be looked at as consisting of two parts: straight line segments of the line (Start, Target) (between the obstacles intersecting the line) and the path segments related to walking around the obstacle boundaries. Because of inequality (8), the sum of the straight line segments will never exceed D. As to the path segments around the obstacles, there is an upper bound guaranteed by Lemma 4 for each obstacle met by MA: that is, not more than $n_i/2$ passings along the same segment of the obstacle boundary will take place. Because of Lemma 3 (see the proof of the lemma), only those obstacles that intersect the straight line (Start, Target) should be counted. Summing up the straight line segments and those corresponding to walking around the obstacles, we obtain (9). □

Theorem 3 suggests that in some special scenes, while under the procedure Bug2, MA may have to go around the obstacles any (large albeit finite) number of times. An important question, then, is how typical such scenes are, and, in particular, what characteristics of the scene influence the length of the path. Theorem 4 and its corollary below address this question. They suggest that the mutual position of the Start point, the Target point, and the obstacles can affect the path length rather dramatically. Together, they significantly improve the upper bound on the length of the paths generated by Bug2—in out-position scenes in general and in scenes with convex obstacles in particular.

THEOREM 4. *Under the procedure Bug2, in the case of an out-position scene* MA *will pass any point of the obstacle boundary at most once.*

In other words, if the mutual position of the obstacle and of the points Start and Target satisfies the out-position requirements, the estimate on the length of the path for the procedure Bug2 reaches its lower bound (1).

PROOF. Figure 8 illustrates the proof. Shaded areas in the figure correspond to one or many obstacles. The boundaries of these areas of the obstacles can be of any shape which is indicated by dotted lines.

Consider an obstacle met by MA on its way to the Target, and consider an arbitrary point Q on the obstacle boundary; assume that Q is not a hit point. Because the obstacle boundary is a simple closed curve, the only way that MA can reach point Q is to come to Q from a previously defined hit point. Move from Q along the already generated part of the path in the direction opposite to

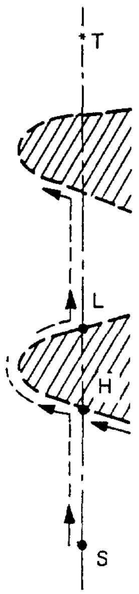

Fig. 8. Illustration for Theorem 4.

the accepted local direction, until the closest hit point along the path is encountered—say, this is H^j. We are interested only in those cases where Q is involved in at least one local cycle, that is, when MA passes the point Q more than once. For this event to occur, MA has to pass H^j at least as many times. In other words, if MA does not pass H^j more than once, it cannot pass Q more than once.

According to the procedure Bug2, the first time MA reaches the point H^j is along the straight line (Start, Target), or, more precisely, along the straight line segment (L^{j-1}, Target). Then, MA turns left and starts walking around the obstacle. To form a local cycle on this path segment, MA has to return to the point H^j again. Since a point can be defined as a hit point only once (see the proof for Lemma 4), the next time MA returns to the point H^j it must approach it from the right (see Figure 8), along the obstacle boundary. Therefore, after having defined H^j, in order to reach it again, this time from the right, MA somehow must cross the straight line (Start, Target) and enter its right semiplane. This can take place in one of only two ways—outside or inside the interval (Start, Target). Consider both cases.

1. The crossing occurs outside the interval (Start, Target). This case can correspond only to an in-position configuration (see Definition 4). Theorem 4, therefore, does not apply.
2. The crossing occurs inside the interval (Start, Target). We now want to prove that such a crossing of the path with the interval (Start, Target) cannot produce local cycles. Notice that the crossing cannot occur anywhere within the interval (Start, H^j) because otherwise at least a part of the straight line segment (L^{j-1}, H^j) would be included inside the obstacle. This is impossible because MA is known to have walked along the whole segment (L^{j-1}, H^j). If the crossing occurs within the interval (H^j, Target) then at the crossing point MA would define the corresponding leave point, L^j, and start moving along the line (Start, Target) toward the Target until it defined the next hit point, H^{j+1}, or reached the Target. Therefore, between H^j and L^j, MA could not have reached into the right semiplane of the line (Start, Target) (see Figure 8).

Since the above argument holds for any Q and the corresponding H^j, we conclude that in an out-position case MA will never cross the interval (Start, Target) into the right semiplane, which prevents it from producing local cycles. □

So far, no constraints on the shape of the obstacles have been imposed. In a special case when all the obstacles in the scene are convex, no in-position configurations can appear, and the upper bound on the length of the path can be improved as follows.

COROLLARY. *If all the obstacles in the scene are convex then, in the worst case, the length of the path produced by the procedure Bug2 is*

$$P = D + \sum_i p_i \tag{10}$$

and, on the average,

$$P = D + 0.5 \cdot \sum_i p_i, \tag{11}$$

where p_i refers to the perimeters of the obstacles intersecting the straight line segment (Start, Target).

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 MA will have to walk around them so as to cover the bigger part of their perimeters (as in case of the obstacle *ob*1, Figure 4), and, for some other obstacles, MA will cover only a smaller part of their perimeters (as in case of the obstacle *ob*2, Figure 4). On the average, one would expect a path that satisfies (11). As for (10), Figure 6 presents an example of such a "bad" scene. The corollary thus assures that for a wide range of scenes the length of paths generated by Algorithm Bug2 will not exceed the universal lower bound (1).

5.3. Test for Target Reachability. As Lemma 4 suggests, under Bug2 MA may pass the same point H^j of a given obstacle more than once, thus producing a finite number p of local cycles, $p = 0, 1, 2, \ldots$. The proof to the lemma indicates that, after having defined a point H^j, MA will never define this point again as an H or an L point. Therefore, on each of the subsequent local cycles (if any), the point H^j will be passed not along the straight line (Start, Target) but along the obstacle boundary. Every time after leaving the point H^j MA can expect one of the following to occur:

MA will never return again to H^j; this happens, for example, if it leaves the current obstacle altogether, or reaches the Target,

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

MA will come back to the point H^j without having defined on the previous cycle a point L^j. In other words, MA could find no other intersection point Q of the line $(H^j$, Target) with the current obstacle such that Q would be closer to the Target than H^j, and the line (Q, Target) would not cross the current obstacle at Q. This can happen only if either MA or the Target are trapped inside the current obstacle (see Figure 9). The condition is both necessary and sufficient, which can be shown similarly to the proof in the target reachability test for the procedure Bug1, Section 4.3.

Based on this observation, a test for Target reachability for the procedure Bug1 can be formulated as follows.

TEST FOR TARGET REACHABILITY. If, on the pth local cycle, $p = 0, 1, \ldots$, after having defined a point H^j, MA returns to this point before it defines at least the

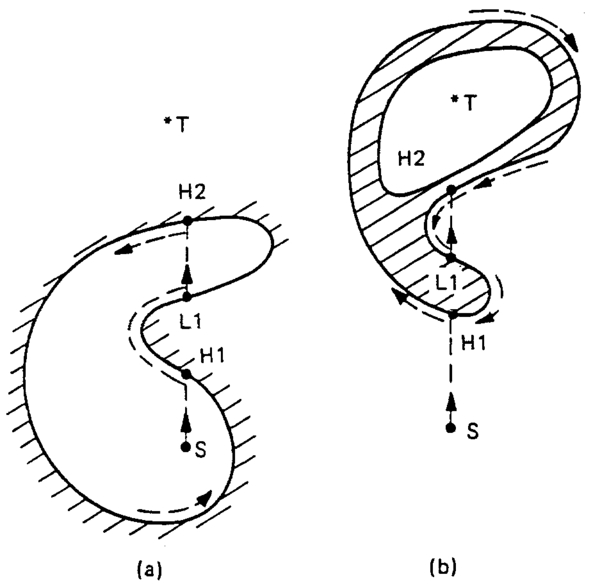

Fig. 9. Examples of traps. The path (dotted line) is executed under Algorithm Bug2. After having defined the point $H2$, the automaton returns to it before it defines any new L point. Therefore, the Target is not reachable.

first two out of the possible set of points L^j, $H^{j+1}, \ldots, H^k$, it means that MA has been *trapped* and hence the Target is not reachable.

6. Improving the Performance of the Basic Algorithms. Each of the algorithms Bug1 and Bug2 has a clear and simple underlying idea; each has its pluses and minuses. Namely, Bug1 never creates any local cycles, but it tends to be "over-cautious" and never covers less than the full perimeter of an obstacle. The procedure Bug2, on the other hand, is more "human" in that it takes advantage of the simple situations, but may become quite inefficient in more difficult cases. The better features of both procedures are combined in the following procedure, called BugM1 (for "modified"). The procedure BugM1 combines the efficiency of the procedure Bug2 in simpler scenes (where MA will pass only portions, instead of the full perimeters, of the obstacles, see Figure 4) with the more conservative strategy of the procedure Bug1 (which limits the corresponding segment of the path around an obstacle to 1.5 of its perimeter, see the bound (7)). In BugM1, for a given point on the path, the number of local cycles containing this point is never larger than two; in other words, MA will never pass the same point of the obstacle boundary more than three times. Although the flow of action in BugM1 is not as "clean" as in the basic algorithms, their termination properties are retained.

The procedure BugM1 is executed at any point of the continuous path. Instead of using the fixed straight line (Start, Target), as in Bug2, BugM1 uses a straight line (L_i^j, Target), with a changing point L_i^j; here, L_i^j indicates the jth leave point on an obstacle. The procedure uses three registers, R_1, R_2, R_3, to store intermediate information; all three are reset to zero when a new hit point, H_i^j, is defined. Specifically, R_1 is used to store the coordinates of the current point, Q_m, of the minimum distance between the obstacle boundary and the Target; R_2 integrates the length of the obstacle boundary starting at H_i^j; and R_3 integrates the length of the obstacle boundary starting at Q_m. (In case of many choices for Q_m, any one of them can be taken.) The test for target reachability mentioned in step 2(d) is explained in Section 5.3. Initially, $i=1, j=1$; $L_1^0=$ Start. The procedure consists of the following steps:

1. From the point L_i^{j-1}, move along the line (L_i^{j-1}, Target) toward the Target until one of the following occurs:
   (a) The Target is reached. The procedure stops.
   (b) An obstacle is encountered and a hit point, H_i^j, is defined. Go to step 2.
2. Using the accepted local direction, follow the obstacle boundary until one of the following occurs:
   (a) The Target is reached. The procedure stops.
   (b) The line (L_i^{j-1}, Target) is met inside the interval (L_i^{j-1}, Target), at a point Q such that the distance $d(Q) < d(H_i^j)$, and the line (Q, Target) does not cross the current obstacle at the point Q. Define the leave point $L_i^j = Q$. Set $j=j+1$. Go to step 1.
   (c) The line (L_i^{j-1}, Target) is met outside the interval (L_i^{j-1}, Target). Go to step 3.
   (d) The automaton returns to H_i^j and thus completes a closed curve (the obstacle boundary) without having defined the next hit point. The Target cannot be reached. The procedure stops.
3. Continue following the obstacle boundary. If the Target is reached, stop. Otherwise, after having traversed the whole boundary and having returned to H_i^j, define a new leave point $L_i^j = Q_m$. Go to step 4.
4. Using the content of R_2 and R_3, determine the shorter way along the obstacle boundary to L_i^j, and use it to get to L_i^j. Apply the test for target reachability (as in Section 4.3). If the Target is not reachable, the procedure stops. Otherwise, designate $L_i^0 = L_i^j$, set $i=i+1$, $j=1$, and go to step 1.

Why does the procedure BugM1 converge? Depending on the scene, the flow of the algorithm fits one of the following two cases:

1. For a given scene, if the condition in step 2(c) of the procedure is never satisfied then the actual flow of the algorithm is that of Bug2, for which convergence has already been shown. In this case, the straight lines (L_i^j, Target) always coincide with the straight line (Start, Target), and no local cycles appear.
2. If, on the other hand, the scene presents an in-position case then the condition in step 2(c) is satisfied at least once; that is, MA crosses the straight line (L_i^{j-1}, Target) outside the interval (L_i^{j-1}, Target). This indicates that there is a danger of multiple local cycles. At this point, MA switches to a more conservative

approach offered by Algorithm Bug1, instead of risking an uncertain number of local cycles it might now expect from the procedure Bug2 (see Lemma 4). MA does this by executing steps 3 and 4 of BugM1 which are identical to steps 2 and 3 of the procedure Bug1.

After one execution of steps 3 and 4, the last leave point on the obstacle is defined, L_i^j, which is guaranteed to be closer to the Target than the corresponding hit point H_i^j (see inequality (7), Lemma 1). Then, MA leaves the ith obstacle and never returns to it again (Lemma 1). From now on, the algorithm (specifically, its steps 1 and 2) will be using the straight line (L_i^0, Target) as the "leading thread." (Note that, in general, the line (L_i^0, Target) does not coincide with the straight lines (L_{i-1}^0, Target) or (Start, Target).) One execution of the sequence of steps 3 and 4 of BugM1 is equivalent to one execution of steps 2 and 3 of Bug1, which guarantees the reduction by one of the number of obstacles that MA will meet on its way. Therefore, as in Bug1, the convergence of this case is guaranteed by Lemma 1, Lemma 2, and its corollary. Since cases 1 and 2 are independent and they exhaust all possible cases, the procedure BugM1 converges.

7. Concluding Remarks.

The two bounds given by (1) and (7) indicate a gap in the path length estimates for the general problem of path planning among unknown obstacles, between at least $\sum p_i$ given by the lower bound (1), and at most $1.5 \cdot \sum p_i$ assured by the upper bound (7) of Algorithm Bug1. This poses an interesting problem of narrowing the gap either by finding a higher lower bound, or by introducing a better path-planning algorithm and lowering the upper bound.

The only work that these authors are aware of on convergence of motion-planning algorithms with uncertainty is the Pledge algorithm [16], which addresses a problem different from ours—namely, how to escape from a maze (and not how to find a given point inside or outside the maze). Very little is known about the performance of such algorithms; there are no performance estimates for the Pledge algorithm. In this paper we limit the performance of such algorithms by the worst-case lower bound (1), the worst-case upper bounds (7), (9), and (10), and by the average estimate (11) above.

On the theoretical level, therefore, the question "How reasonable can the paths generated by the algorithms for robot path planning among unknown obstacles be?" has a simple answer: Algorithm Bug1 is the best that can be offered today. Another advantage to Bug1 is that it does not require knowledge of the robot's current coordinates; it is sufficient if the robot can measure its distance from and its direction toward the target.

On a more practical level, Bug1 is a rather "conservative" algorithm, whose thoroughness in investigating each obstacle may or may not fit our notion of "being reasonable." Algorithm Bug2, on the other hand, is more "aggressive" and more efficient in many cases. Its behavior seems more reasonable, more "human." And, as happens sometimes with humans when they desperately try to reach their target and instead get lost in the woods, Bug2 pays a high price on those rare occasions when the set (obstacles, Start, Target) presents an in-position arrangement (see Section 5.2). In this sense, Algorithm BugM1 provides a compromise between the pluses and minuses of Algorithms Bug1 and Bug2.

One gets an additional insight into the operation and the "area of expertise" of the basic algorithms by trying them in maze search problems. The problem of search in an unknown or a known maze can be formulated in a number of ways. In one version (see, e.g., [14]), the automaton, after starting at an arbitrary cell of an unknown maze, must eventually visit every single cell of the maze, without passing through any barriers. Any pair of cells in the maze is assumed to be connected via other cells. Note that in this version there is no notion of a specific target cell, and no sense of direction is present. Because of that, neither of the basic algorithms can be used.

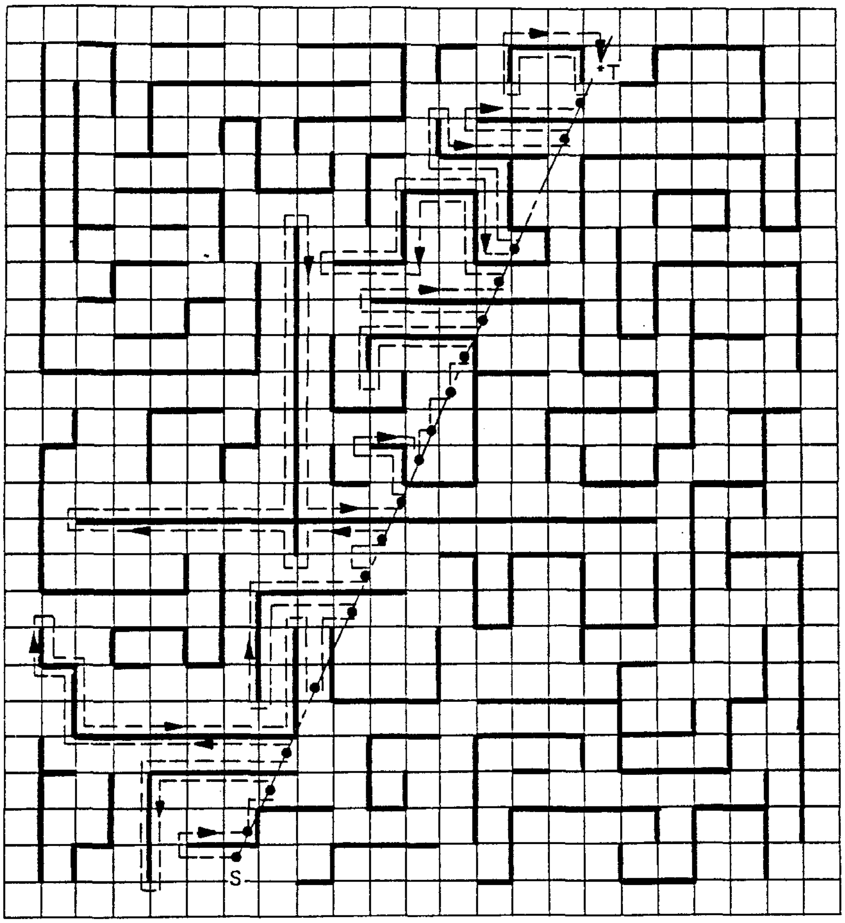

Fig. 10. Example of a walk in a maze using Algorithm Bug2; S = Start, T = Target. Points in which the automaton's path (dotted line) crosses the imaginary straight line (S, T) are indicated by dots. Maze barriers are shown as heavy lines.

In another version of the maze search problem, given a starting cell, the automaton has to find an exit from an unknown maze; the coordinates of the exit are not known. Although no target is explicitly presented here, either of our algorithms can be used. Namely, the automaton can choose any point somewhere in infinity as its target position, and then use the basic algorithms as usual. Then, if the exit exists, it is guaranteed to be found.

In still another version of the maze search problem [15], the barriers of the maze are formed by a set of horizontal and vertical line segments; unlike the two previous versions, full information about the maze is assumed. Given the coordinates of two points (cells) in a maze, the problem is to find a route from one to the other. Assuming that the barriers have nonzero thickness, our work shows that full information about the maze is redundant; local information (from any mechanism simulating a "tactile sensor") is sufficient to solve the problem. It is not clear whether this observation can have interesting consequences for applications—for example, for the routing problem in VLSI design.

Since the performance of Algorithm Bug2 can differ remarkably, depending on whether it deals with an in-position or an out-position case (see the estimates (9)-(11)), its behavior in a maze depends largely on the local topology of the barriers and, specifically, on whether the barriers in the maze form connected or disconnected patterns. This is demonstrated in an example shown in Figure 10. (To fit a typical convention of the maze literature, we present a discrete version of the continuous path-planning problem: the maze is a rectangular cell structure, with each cell being a little square; any cell crossed by the straight line (S, T) is considered to be lying on the line.) At first, one might think that the automaton is dealing here with an in-position scene and, therefore, Algorithm Bug2 is likely to produce an inefficient path with local cycles. Observe that this is not the case: given the fact that the automaton knows nothing about the maze, it performs quite well.

References

[1] J. T. Schwartz and M. Sharir, On the "piano movers" problem: I. The case of a two-dimensional rigid polygonal body moving amidst polygonal barriers, *Comm. Pure Appl. Math.*, **36** (1983), 345-398.

[2] T. Lozano-Perez and M. Wesley, An algorithm for planning collision-free paths among polyhedral obstacles, *Comm. ACM*, **22** (1979), 560-570.

[3] H. Moravec, The Stanford cart and the CMU rover, *Proc. IEEE*, **71** (1983), 872-874.

[4] R. A. Brooks, Solving the find-path problem by good representation of free space, *IEEE Trans. Systems Man Cybernet.*, **13** (1983).

[5] T. O. Binford, Visual perception by computer, *Proceeding of the IEEE Systems Science and Cybernetics Conference*, Miami, FL, 1971.

[6] J. Reif, Complexity of the mover's problem and generalizations, *Proceedings of the 20th Symposium on the Foundations of Computer Science*, 1979.

[7] D. L. Pieper, The kinematics of manipulators under computer control, Ph.D. Thesis, Stanford University, 1968.

[8] R. Paul, Modeling trajectory calculation and servoing of a computer controlled arm, Ph.D. Thesis, Stanford University, 1972.

[9] J. T. Schwartz and M. Sharir, On the "piano movers" problem: II. General techniques for computing topological properties of real algebraic manifolds, *Adv. in Appl. Math.*, **4** (1983), 298-351.

[10] J. Hopcroft, D. Joseph, and S. Whitesides, On the movement of robot arms in 2-dimensional bounded regions, *Proceedings of the IEEE Foundations of Computer Science Conference*, Chicago, IL, 1982.

[11] B. Bullock, D. Keirsey, J. Mitchell, T. Nussmeier, and D. Tseng, Autonomous vehicle control: an overview of the hughes project, *Proceedings of the IEEE Computer Society Conference "Trends and Applications, 1983: Automating Intelligent Behavior"*, Gaithersburg, MD, 1983.

[12] A. M. Thompson, The navigation system of the JPL robot, *Proceedings of the Fifth Joint International Conference on Artificial Intelligence*, Cambridge, MA, 1977.

[13] D. M. Kersey, E. Koch, J. McKisson, A. M. Meystel, and J. S. B. Mitchell, Algorithm of navigation for a mobile robot, *Proceedings of the International Conference on Robotics*, Atlanta, GA, 1984.

[14] M. Blum and D. Kozen, On the power of the compass (or, why mazes are easier to search than graphs), *Proceedings of the 19th Annual Symposium on Foundation of Computer Science*, Ann Arbor, MI, 1978.

[15] W. Lipski and F. Preparata, Segments, rectangles, contours, *J. Algorithms*, **2** (1981), 63-76.

[16] H. Abelson and A. diSessa, *Turtle Geometry*, MIT Press, Cambridge, MA, 1980, pp. 176-199.

[17] C. Yap, Algorithmic motion planning, in *Advances in Robotics*, Vol. 1 (J. Schwartz and C. K. Yap, eds.), Lawrence Erlbaum, NJ, 1986.

[18] L. Meijdam and A. de Zeeuw, On expectations, information, and dynamic game equilibria, in *Dynamic Games and Applications in Economics* (T. Basar, ed.), Springer-Verlag, New York, 1986.

[19] E. Moore, The firing squad synchronization problem, in *Sequential Machines* (E. Moore, ed.), Reading, MA, 1964.

[20] J. Traub, G. Wasilkowski, and H. Wozniakowski, *Information, Uncertainty, Complexity*, Addison-Wesley, Reading, MA, 1983.

[21] J. Reif, A survey on advances in the theory of computational robotics, in *Adaptive and Learning Systems* (K. Narendra, ed.), Plenum, New York, 1986.

[22] W. S. Massey, *Algebraic Topology*, Harcourt, Brace, & World, New York, 1967.

[23] V. Lumelsky, Effect of robot kinematics on motion planning in unknown environment, *Proceedings of the 24th IEEE Conference on Decision and Control,* Fort Lauderdale, FL, 1985.

[24] V. Lumelsky, Continuous motion planning in unknown environment for a 3D cartesian robot arm, *Proceedings of the IEEE International Conference on Robotics and Automation*, San Francisco, 1986.