T-AC/31/11/10441

# Dynamic Path Planning for a Mobile Automaton with Limited Information on the Environment

V. J. Lumelski
A. A. Stepanov

Reprinted from: IEEE TRANSACTIONS ON AUTOMATIC CONTROL, Vol. AC-31, No. 11, NOV. 1986.

# Dynamic Path Planning for a Mobile Automaton with Limited Information on the Environment

VLADIMIR J. LUMELSKY AND ALEXANDER A. STEPANOV

*Abstract*—**The problem of path planning is studied for the case of a mobile robot moving in an environment filled with obstacles whose shape and positions are not known. Under the accepted model, the automaton**

Manuscript received January 25, 1985; revised March 26, 1986 and June 2, 1986. This paper is based on a prior submission of July 18, 1984.

V. J. Lumelsky is with Yale University, New Haven, CT 06520.
A. A. Stepanov is with the Polytechnic University, Brooklyn, NY 11201.

IEEE Log Number 8610441.

**knows its own and the target coordinates, and has a "sensory" feedback which provides it with local information on its immediate surroundings. This information is shown to be sufficient to guarantee reaching a global objective (the target), while generating reasonable (if not optimal) paths. A lower bound on the length of paths generated by any algorithm operating with uncertainty is formulated, and two nonheuristic path planning algorithms are described. In the algorithms, motion planning is done continuously (dynamically), based on the automaton's current position and on its feedback. The effect of additional sources of information (e.g., from a vision sensor) on the outlined approach is discussed.**

## I. Introduction

To generate a path for a mobile automaton means to find a continuous trajectory leading from the initial position of the automaton to its target position, in the environment filled with obstacles. The path generation strategy is affected by the assumptions about geometric characteristics of the environment, as well as by information available to the automaton. For our purposes, we distinguish between those approaches where full information on the environment is given and approaches where an element of uncertainty is assumed.

One model with full information that has been extensively studied in recent years is the Piano Mover's problem, in which full information is assumed to be available on the geometry and positions of the obstacles and of the moving body. For this model, the complexity of the path generation problem has been investigated, and a number of heuristic and nonheuristic approaches involving moving rigid or hinged bodies in two- or three-dimensional space have been considered [1]–[5].

Works on path generation with uncertainty come primarily from studies on navigation of an autonomous vehicle moving along a two-dimensional surface [6], [7], [14] and from studies on local control (e.g., compliance control) of robot arms [8]. In these works, various heuristics are used for path generation in the limited area around the automaton for which information is available. Within this limited area, the problem is actually treated as one with full information.

The purpose of this study is to design a methodology for nonheuristic motion planning in an uncertain environment. Situations with uncertain or unknown environments are typical in navigation of autonomous vehicles. In applications involving industrial robot arms, where the environment is easier to control, an assumption of full information is still unwarranted in many cases, for example, when the control relies on the real-time information from sensors.

We consider a case opposite to that of the Piano Mover's model: it is assumed that no information on the geometry or positions of obstacles is available. An idealized two-dimensional model with a point automaton is considered. (Actually, any surface homeomorphic to the plane suits this approach.) Any obstacle is of finite area, and its boundary is of finite length; a scene of finite size contains a finite number of obstacles. The coordinates of the target and of the obstacles are fixed. The automaton knows its own and the target coordinates, but has no information about the obstacles. The lack of information about the environment is compensated in the model by the automaton's ability to recognize the fact of meeting an obstacle, using a feedback from some contact or noncontact sensors, and to walk along the obstacle boundary. In this context, "avoiding obstacles" means maneuvering around the most recently discovered obstacle in some reasonable way. The question being asked is whether the purely local information provided by the feedback is sufficient to reach a global goal (here, the target position).

A somewhat similar problem is addressed by the Pledge algorithm [11] which uses the same model of the environment and of the automaton and guarantees escaping from a maze. However, the Pledge algorithm cannot be used to find a fixed point inside or outside the maze.

When analyzing the problem, one can see that seemingly reasonable strategies do not guarantee success. Consider, for example, this algorithm [see Fig. 1(a)].

1) Go directly to the target until one of the following occurs.
   a) Target is reached. The procedure stops.
   b) An obstacle is encountered. Go to Step 2.
2) Turn left and follow the obstacle boundary until one of the following occurs.
   a) Target is reached. The procedure stops.
   b) The direction on the target clears. Go to Step 1.

Unfortunately, depending on the scene, this strategy can create problems. For example, in Fig. 1(b), in spite of the fact that each of the obstacles is of finite size, the strategy will take the automaton to infinity, instead of the target, whereas in Fig. 1(c) the strategy will result in infinite looping.

To analyze the problem, we first introduce a formal model of the automaton and the environment and develop a lower bound for the length of generated paths. The bound is expressed in terms of perimeters of obstacles which intersect the straight line between the starting and the target points. Then, we show that the local information provided to the automaton by its sensors is indeed sufficient for the path planning. Two nonheuristic algorithms are introduced, each of which guarantees reaching the target or concluding that the target cannot be reached. The general idea is to arrange a sequence of steps, each having in it some special critical points such that they correspond to monotonically decreasing (and converging to zero) distances between the automaton and the target. Along the path, however, no monotonic behavior of the distance to the target is expected. Path planning is done, therefore, continuously (dynamically), based on the automaton's current position, and also on the local information from the feedback.

The algorithms are described in the context of a "tactile" feedback, followed by a discussion on using the same ideas in the context of a noncontact (e.g., vision) feedback, and on handling finite dimensions of the automaton. More detailed analysis, examples, and proofs of lemmas and theorems can be found in [9], [10]. Although our approach does not generalize easily to three or more dimensions, a case-by-case study reveals that a number of applications do lend themselves into it (for applications to two- and three-dimensional arm manipulators, see [12], [13]).

In the algorithms, the automaton is continuously generating the path, based on its current coordinates and on the incoming local information. The general idea is similar to the local methods of studying the geometrical phenomena developed in [11]. No approximation of obstacles (e.g., by polygons) is done, and no explicit reduction to a discrete space (e.g., to a set of nodes of the polygons) takes place. As a result, the whole space (and not only those points that lie along certain subspaces, e.g., along the edges of the connectivity graph representing the nodes of the polygons) are available for path generation purposes. Since, unlike the Piano Mover's problem, the only information to be processed at any given moment is limited to the automaton's immediate surroundings, the algorithms are rather efficient and suitable for real-time implementation.

## II. Model

The goal of the mobile automaton ($MA$) is to generate a continuous path from the point start ($S$) to the point target ($T$). Both $S$ and $T$ are fixed points. A scene is a set of nonintersecting simple closed curves in the plane, such that any disk of a finite radius intersects only a finite set of curves. Each simple closed curve represents the boundary of an obstacle. To avoid some degenerate cases, any obstacle boundary is assumed to cross any straight line a finite number of times; also, any disk intersects a finite number of obstacles.

The following notation is used: $d(A, B)$ is the distance between any two points $A$ and $B$ in the scene; specifically, $d(S, T) = D$, where $D$ is a constant; $d(A_i, B)$ signifies the fact that the point $A$ is located on the boundary of the $i$th obstacle met by $MA$ on its way to $T$; $P$ is the total length of the path generated by $MA$ on its way from $S$ to $T$; $p_i$ is the perimeter (the length of the boundary) of the $i$th obstacle. The line $(S, T)$ is called the main line, or $M$-line.

$MA$ is a point; thus, any opening between two obstacles is considered passable. $MA$ is given the position of target and its own current position (coordinates); using its "sensors," it can recognize the fact of hitting an obstacle. Also, $MA$ is capable of moving toward $T$ along a straight line, and of moving along the obstacle boundary. Realizing this capability requires special algorithms which are beyond the scope of this note; it is important, however, that the only information on the environment that it requires is very small since it is limited to the current immediate surroundings of $MA$.

A *local direction* is defined as a once-and-for-all decided upon direction

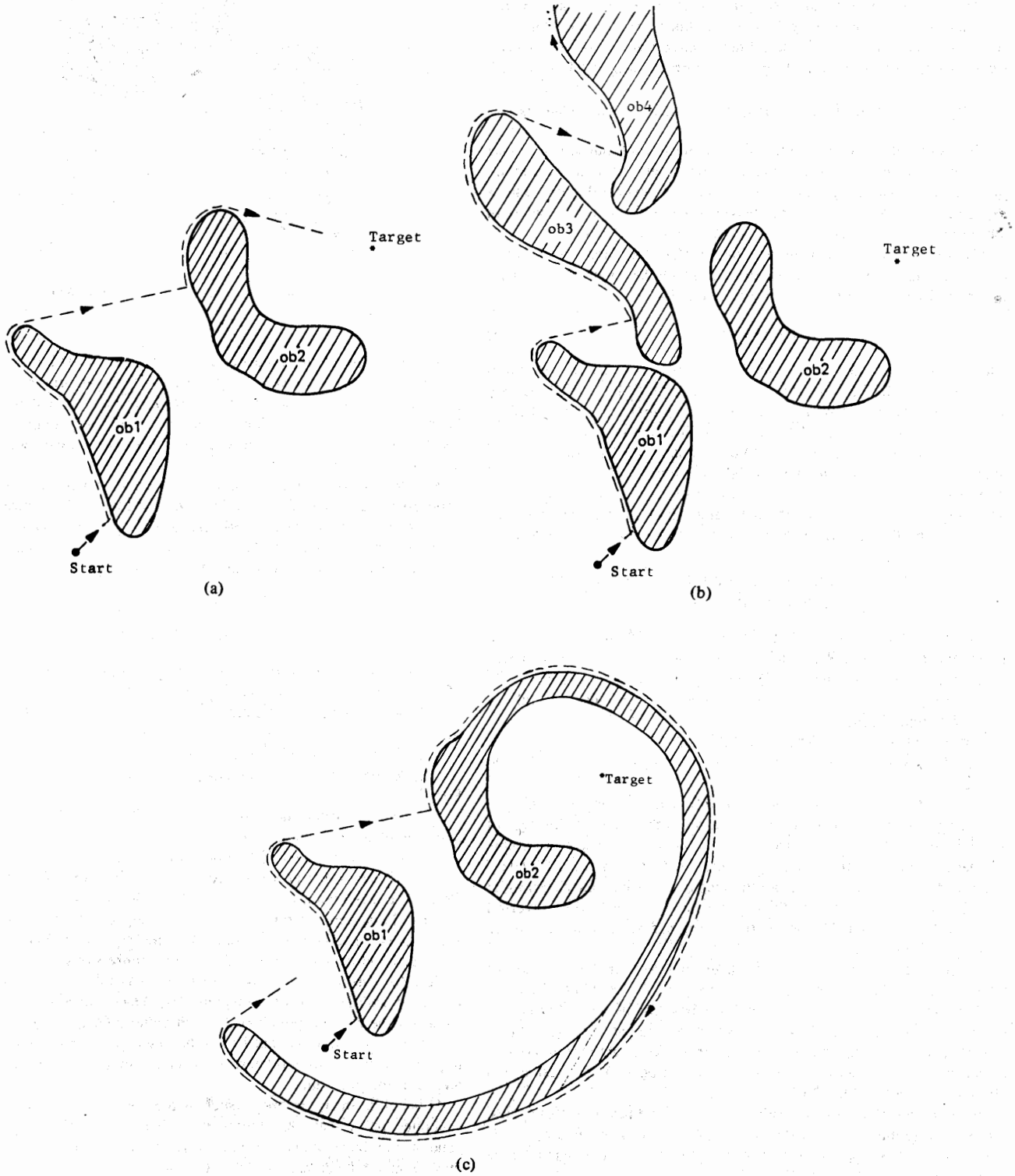


Fig. 1. This intuitively reasonable algorithm (walk toward the target whenever you can) will often work (a), but it does not guarantee termination (b), (c).

of passing around an obstacle. For the two-dimensional problem, it can be either left or right. Because of incompleteness of information, where *MA* hits an obstacle, there is no information or criteria that could help it decide which local direction it should use. For the sake of clarity and without losing generality, assume that the local direction of *MA* is always *left* (as in Fig. 3, dashed line). *MA* is said to define a *hit point* $H$ on an obstacle boundary when, while moving along the $M$-line toward target, *MA* meets the first point of an obstacle; it defines a *leave point* $L$ on the obstacle boundary when it leaves the obstacle and starts moving in "free space."

## III. A Lower Bound for the Path Generation Problem

The bound determines, within the framework of our model, what ultimate performance can be expected from any path generation algorithm. The bound applies to the length of generated paths and is expressed in terms of the distance between $S$ and $T$ and of perimeters of the obstacles that intersect the $M$-line. It is given by the following.

*Theorem 1:* For any path generating algorithm and any positive (however small) $\epsilon$, there exists a scene for which the length $P$ of the path generated by the algorithm will obey the relationship

$$P \geq D + \Sigma p_i - \epsilon \tag{1}$$

where $P$, $D > 0$, and $p_i$ have been defined above, and $\Sigma p_i$ is the sum of perimeters of all the obstacles intersecting the $M$-line. The theorem suggests that no matter what algorithm is being used, a nontrivial scene can be designed such that *MA* will have to pass around each obstacle at least once. (This is true no matter how "small" the uncertainty is—as long as it is present—and how good the sources of additional local information are.) The theorem thus describes a major characteristic of operating in an environment with uncertainty; it also presents a good way for measuring the performance of various path generating procedures.

## IV. An Upper Bound for the Path Generation Problem: Algorithm Bug1

This procedure is executed at any point of a continuous path. It uses three registers, $R_1$, $R_2$, $R_3$, to store intermediate information, and consists of the following steps (see Fig. 2); initially, $j = 1$; $L_0 =$ start.

1) From point $L_{j-1}$, move toward target along a straight line until one of the following occurs.

a) 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. If reach target, stop. Use $R_1$ to store coordinates of the current point, $Q_m$, of a minimal distance from target (this computation takes a simple comparison at each path point), $R_2$ to integrate the length of the boundary starting at $H_j$, and $R_3$ to integrate the length of the boundary starting at $Q_m$. (In case of many choices for $Q_m$, take any.) After having traversed the whole boundary and having returned at $H_j$, define a new leave point as $L_j = Q_m$. Go to Step 3.

3) Using the content of $R_2$ and $R_3$, determine the shorter way along the boundary to $L_j$, and use it to get to $L_j$. Set $j = j + 1$. Go to Step 1.

*Test for Target Reachability* for Bug1 is as follows. If, after having defined a leave point $L_j$, *MA* discovers that the straight line segment ($L_j$, target) crosses an obstacle at $L_j$, then either *MA* or target is *trapped* and, therefore, target cannot be reached (see [10] for details). The test should be used as a part of Step 3 of the procedure.

It can be shown that the algorithm converges and never creates cycles. The following theorem estimates the length of paths generated by Bug1; this is also our best upper bound on the length of paths generated under the accepted model.

*Theorem 2:* The length of a path generated by the algorithm Bug1 never exceeds the limit

$$P = D + 1.5 \cdot \Sigma p_i. \tag{2}$$

Both bounds developed above indicate a gap in the length estimates for the path planning problem with uncertainty, between at least $\Sigma p_i$ given by the lower bound, and at most $1.5 \cdot \Sigma p_i$ assured by the algorithm Bug1. This poses an interesting problem of narrowing the gap by either finding a higher lower bound, or by introducing a better path planning algorithm and lowering the upper bound. (In spite of many attempts, we were not able to improve the bounds.)

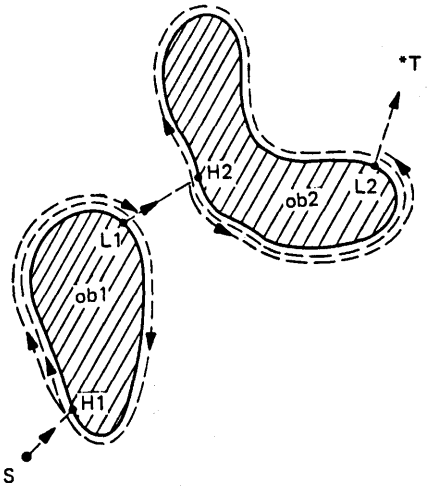


Fig. 2. Automaton's path under the algorithm Bug1.

Note that the requirement to know the current *MA* coordinates may be softened in case of Bug1. Namely, the algorithm will function and the convergence is guaranteed if only the current direction on and distance from the target are known. This may have important ramifications when choosing between the algorithms Bug1 and Bug2 based on available sensors.

## V. Algorithm Bug2

The procedure is executed at any point of a continuous path. It consists of the following steps (see Fig. 3); initially, $j = 1$; $L_0 =$ start.

1) From point $L_{j-1}$, move along the $M$-line until one of the following occurs.

a) 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) Target is reached. The procedure stops.

b) $M$-line is met at a distance $d$ from $T$ such that $d < d(H_j, T)$. Define the leave point $L_j$. Set $j = j + 1$. Go to Step 1.

c) The automaton returns to $H_j$ and thus completes a closed curve along the obstacle boundary without ever meeting the $M$-line. The target is trapped and cannot be reached. The procedure stops.

*Test for Target Reachability* for Bug2 is as follows. If, after having identified and left a given hit point $H_j$, *MA* returns to $H_j$ before it identifies the next hit point, $H_{j+1}$, then either *MA* or target is *trapped* and, therefore, target cannot be reached (see [10] for details). The test requires storing up to two latest hit points; it should be used as a part of Step 2 of the procedure.

Although in many instances (such as the one depicted in Fig. 3) Bug2 generates paths that are shorter than those defined by (1) or those generated by Bug1, it may create cycles and produce longer paths. A *local cycle* is said to be created when *MA* passes some segment of the obstacle boundary more than once. It can be shown that local cycles are created only in scenes with rather special obstacles and special initial positions of start and target relative to the obstacles. A case of an *in-obstacle* (Fig. 4) refers to a mutual position of the pair of points ($S$, $T$) and a given obstacle where i) the straight line segment ($S$, $T$) crosses an obstacle boundary at least once, and ii) either $S$ or $T$ lie inside the convex hull of the obstacle.

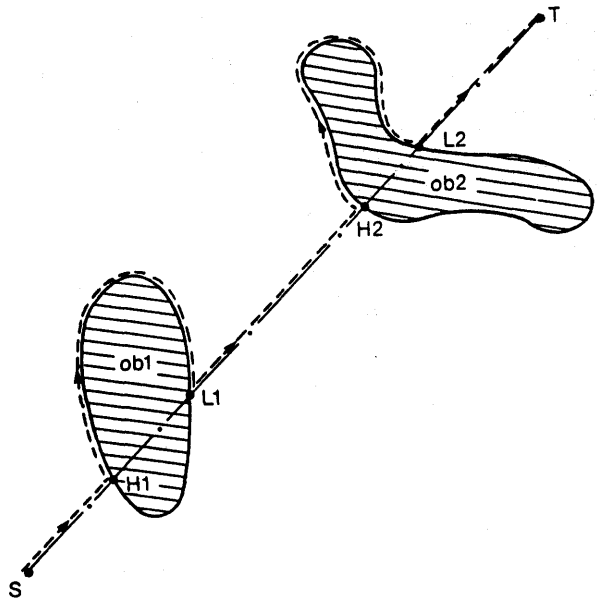


Fig. 3. Automaton's path under the algorithm Bug2.

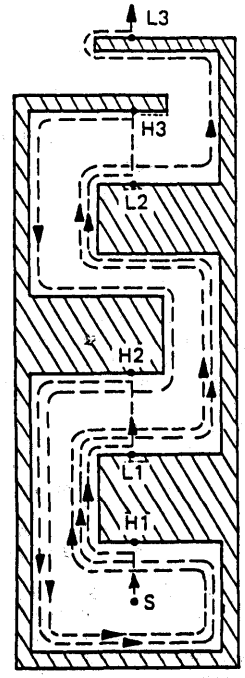


Fig. 4. Automaton's path in a maze-like obstacle, procedure Bug2. The obstacle complexity is measured by the number of times, $n_i$, that the straight line $(S, T)$ crosses it; here, $n_i = 10$. At most, the path passes one segment $(H1, L1)$, three times; that is, there are at most two local cycles in this path.

A case of an *out-obstacle* (Fig. 3) refers to a mutual position of the pair $(S, T)$ and an obstacle in which both $S$ and $T$ lie outside the convex hull of the obstacle.

Let $n_i$ be the number of intersections between the $M$-line and the $i$th obstacle; thus, $n_i$ is a characteristic of the set (obstacles, start, target) and not of a specific algorithm. Clearly, for any convex obstacle $n_i = 2$. If an obstacle is not convex, the situation still may be as simple; for example, a concave obstacle ob2 in Fig. 3 has $n_2 = 2$ and creates no local cycles. However, in Fig. 4 the segment of the boundary from $H1$ to $L1$, $(H1, L1)$, will be passed (under Bug2) three times; segments $(L1, L2)$ and $(H2, H1)$, twice each; and segments $(L2, L3)$ and $(H3, H2)$, once each.

*Lemma 1:* Under the procedure Bug2, $MA$ will pass any segment of the $i$th obstacle boundary at most $n_i/2$ times.

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

$$P = D + \sum \frac{n_i p_i}{2}. \tag{3}$$

Together, the lemma and the theorem guarantee termination, put a limit on the number of local cycles, and provide the upper bound on the length of paths generated by Bug2. The upper bound (3) is constructive, in the sense that a set of obstacles can be readily suggested on which the bound will be reached. It can be shown, however, that local cycles can appear only in rather special cases with in-obstacle positions. For the rest of the cases, paths generated by Bug2 never exceed the lower bound (1). Specifically, it can be shown that in the case of convex obstacles the upper bound for the length of path is, in the worst case,

$$P = D + \Sigma p_i \tag{4}$$

and, on the average,

$$P = D + 0.5 \cdot \Sigma p_i. \tag{5}$$

This suggests that the shape of the obstacles is not important in creating local cycles unless special positions of $S$ and $T$ relative to the obstacles are involved. In other words, situations similar to that shown in Fig. 4 are rare; local cycles do not appear in many scenes with in-obstacle positions, even in many mazes (such an example can be found in [10]).

## VI. Handling Noncontact Sensors and the Automaton's Dimensions

Consider a case when, instead of a tactile feedback assumed in the model, $MA$ is equipped with a sensor system which provides it with information within a disk of a radius $r_v$ ("radius of vision") centered at the current position of $MA$. Various sensors fall in this category: vision systems, ultrasonic and infrared proximity sensors, time-of-flight range finders, etc.

Consider, for example, the algorithm Bug2. When incorporating a noncontact feedback in the algorithm, it is important to assure that $MA$ will be able to always come back to the $M$-line. With this in mind, the modification is as follows. At its current position, $MA$ reconstructs, in the area that it can "see," the path segment that would have been generated if no vision were available, and replaces it with a straight line segment. If no obstacles interfere with vision, the resulting segment will be exactly of length $r_v$. The operation is repeated at each point of the path, resulting in a curved path which smooths out the path of a "blind" $MA$ (Fig. 5).[1] With such a modification, convergence of Bug2 is preserved. Increasing the radius $r_v$ produces, in general, shorter paths. In the limit, with $r_v$ going to infinity, it results in locally optimal paths; this comes from the simple fact that the distance between the current position of $MA$ and its next intermediate visible goal is always covered along a straight line.

When obstacles interfere with vision, the generated straight line segments become shorter, and, in theory, can be eventually reduced to zero by a "bad" set of obstacles. Consequently, all the estimates for path lengths developed for the "blind" $MA$ remain true for the case with vision.

Now, consider an $MA$ of finite dimensions. We replace the single feedback sensor of a point $MA$ with a set of sensors covering the $MA$ body such that any point of the body is capable of detecting an obstacle. Also, assume that no changes in $MA$ orientation along the path is allowed. Then, the algorithms described above can be used and their convergence is still assured. This is primarily because nowhere in the algorithms the

[1] Interestingly, the segment $(A, B)$ of the path in Fig. 5 forms a curve called *tractrix*; this curve has been studied independently by Leibnitz and Huygens in the late XVII century. One forms the tractrix here by putting one end of a stick of the length $r_v$ at the pont $A$ and its other end at the point where it meets the line $(S, T)$, and then pulling the second end along $(S, T)$.

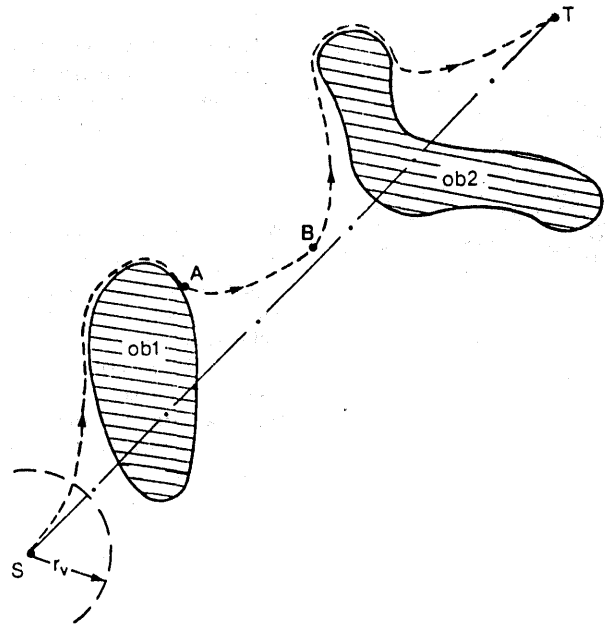


Fig. 5. Automaton's path in the scene of Fig. 3 when additional (vision) information is available.

dimensions, shape, or positions of the obstacles are used. The motion of a "finite" *MA* with fixed orientation among original obstacles is known to be reducible to the motion of a point *MA* among new, "grown" obstacles, whose dimensions are changed to reflect the corresponding shrinking of *MA* (for a detailed discussion, see, e.g., [2]).

For this case, slight modifications in the definitions will be needed. Define some point of *MA*—say, one of its corners or its center of gravity—as the *core point (C-point)*. Define the motion of *MA* along the line (*S*, *T*) as that of moving *C*-point along (*S*, *T*). A hit point is defined as a position of *C*-point at which *MA*, while moving along (*S*, *T*), meets the obstacle boundary first. A leave point (for example, in the algorithm Bug2) is defined as a position of *C*-point at which *C*-point, while *MA* is passing around the obstacle, meets the line (*S*, *T*) at a distance from the target shorter than that from the last hit point to the target. In general, the actual path of a finite *MA* (e.g., of its *C*-point) will differ from that of a point *MA*. It is possible, for example, that, given a set of obstacles and *S* and *T* positions, a finite *MA* will produce local cycles whereas a point *MA* will not.

## VII. Concluding Remarks

The only work that these authors are aware of on convergence of path planning algorithms operating with unknown obstacles is the Pledge algorithm [11], which addresses a different problem: namely, how to escape from a maze (and not how to find a given point inside or outside of the maze). Very little is known about the performance of such algorithms. There is no path length estimates even for the Pledge algorithm. In this note, we limit the algorithm performance by the lower bound (1) and by the upper bounds (2)–(5) above.

On the theoretical level, therefore, the question, "How reasonable are the paths generated by these algorithms?," the answer is simple: the algorithm Bug1 is the best that can be offered today. Also, unlike Bug2, Bug1 does not require knowing the automaton's current coordinates. On the practical level, Bug1 is a rather "conservative" algorithm, whose thoroughness is investigating each obstacle met on its way may or may not be related to our notion of "being reasonable." The algorithm Bug2, on the other hand, is more "aggressive" and more efficient in many cases. Its behavior (especially, in the version with vision) seems more reasonable, more "human." And, as it often happens with humans who get lost in the woods while desperately trying to get to their target, Bug2 pays a high price in those rare occasions when points start and target are positioned in a rather special way relative to a nontrivial scene (see the last paragraph in Section V).

The algorithms Bug1 and Bug2 are based on two quite different ideas. It would be natural to try to combine their better features, so as to avoid investigating the whole boundary of each obstacle, while limiting the number of local cycles. Such a version, which limits the number of local cycles to a constant, is described in [10].

## References

[1] J. T. Schwartz and M. Sharir, "On the "Piano Movers" problem. II. General techniques for computing topological properties of real algebraic manifolds," *Advances Appl. Math.*, no. 4, 1983.

[2] T. Lozano-Perez, "Spatial planning: A configuration space approach," *IEEE Trans. Comput.*, vol. C-10, Feb. 1983.

[3] R. A. Brooks, "Solving the find-path problem by representing free space as generalized cones," Artificial Intelligence Lab., Mass Inst. Technol., Cambridge, AI Memo 674, May 1982.

[4] J. Reif, "Complexity of the mover's problem and generalizations," in *Proc. 20th Symp. Foundations of Comput. Sci.*, 1979.

[5] J. Hopcroft, D. Joseph, and S. Whitesides, "On the movement of robot arms in 2-dimensional bounded regions," in *Proc. IEEE Foundations of Comput. Sci. Conf.*, Chicago, IL, Nov. 1982.

[6] B. Bullock, D. Keirsey, J. Mitchell, T. Nussmeier, and D. Tseng, "Autonomous vehicle control: An overview of the Hughes project," in *Proc. IEEE Comput. Society Conf.*; and *Trends and Applications, 1983: Automatic Intelligent Behavior*, Gaithersburg, MD, May 1983.

[7] A. M. Thompson, "The navigation system of the JPL robot," in *Proc. 5th Joint Int. Conf. Artificial Intelligence*, Cambridge, MA, Aug. 1977.

[8] M. T. Mason, "Compliance and force control for computer controlled manipulators," in *Robot Motion*. Cambridge, MA: M.I.T. Press, 1982.

[9] V. Lumelsky and S. Stepanov, "Navigation strategies for an autonomous vehicle with incomplete information on the environment," General Elec. Corp. Res. Center, Tech. Rep. 84CRD070, Apr. 1984.

[10] V. J. Lumelsky and A. A. Stepanov, "Path planning strategies for a traveling automaton in an environment with uncertainty," Center for Syst. Sci., Yale Univ., Tech. Rep. 8504, Apr. 1985.

[11] H. Abelson and A. diSessa, *Turtle Geometry*. Cambridge, MA: M.I.T. Press, 1980.

[12] V. J. Lumelsky, "Continuous robot motion planning in unknown environment," in *Adaptive Learning Systems: Theory and Applications*, K. Narendra, Ed. New York: Plenum, 1986.

[13] V. J. Lumelsky, "Continuous motion planning in unknown environment for a 3D Cartesian robot arm," in *Proc. IEEE Int. Conf. on Robotics and Automat.*, San Francisco, CA, Apr. 1986.

[14] H. P. Moravec, "The Stanford cart and the CMU rover," *Proc. IEEE*, no. 7, 1983.