JOURNAL OF COMPLEXITY 3, 146–182 (1987)

Algorithmic and Complexity Issues of Robot Motion in an Uncertain Environment*

VLADIMIR J. LUMELSKY

Center for Systems Science, Department of Electrical Engineering, Yale University, New Haven, Connecticut 06520

This paper presents a survey of one approach to planning collision-free paths for an automaton operating in an environment with obstacles. Path planning is one of the central problems in robotics. Typically, the task is presented in the two- or three-dimensional space, with the automaton being either an autonomous vehicle or an arm manipulator with a fixed base. The multiplicity of approaches one finds in this area revolves around two basic models: in one, called *path planning with complete information,* perfect information about the geometry and positions of the robot and the obstacles is assumed, whereas in the other, called *path planning with incomplete information,* an element of uncertainty about the environment is present. The approach surveyed here, called *dynamic path planning,* has been developed in the last few years; it is based on the latter model and gives rise to algorithmic and computational issues very different from those in the former model. The approach produces provable (nonheuristic) path planning algorithms for an automaton operating in a highly unstructured environment where no knowledge about the obstacles is available beforehand and no constraints on the geometry of the obstacles are imposed. © 1987 Academic Press, Inc.

1. INTRODUCTION

The problem of planning collision-free paths for an automaton operating in an environment (a scene) with obstacles has become one of the key issues in robotics in recent years. This interest is not surprising given the direct relationship of the problem to many robot applications and to the more general issue of programming complex automatic systems. The automaton can be an autonomous vehicle traveling along a two-dimensional surface with hills and valleys, or it can be an industrial robot arm manipulator which is required to avoid obstacles that are present in its work space. Two basic models have emerged based on different assumptions

* Supported in part by the National Science Foundation under Grant DMC-8519542.

about the information available for motion planning. This paper studies algorithmic and complexity issues of robot motion in an uncertain environment, in the context of a specific approach, called *dynamic path planning*. In this introductory section, the two basic models used in the literature on path planning are described, followed by a brief review of the state of affairs in the two corresponding categories of works, and by the basic idea of the approach of dynamic path planning; the approach is surveyed in more detail in the following sections.

1.1 *Path Planning Problem—Basic Models*

Depending on which of the two following basic models is being used, the current research on robot path planning can be classified into two large categories. In the first model, called *path planning with complete information* (another popular term is the *Piano Movers problem*), perfect information about the moving automaton and the obstacles is assumed. In the second model, called *path planning with incomplete information*, an element of uncertainty is present, and the missing data is typically provided by some source of local information (e.g., sensory feedback, such as an ultrasound range finder or a vision module). Another important distinction can be made between the provable (other terms—nonheuristic, exact, algorithmic) and heuristic approaches.

The model with complete information (Piano Movers model) is formulated as follows.[1] Given a solid object (or a combination of such objects) in two- or three-dimensional space, whose size, shape, and initial and target position and orientation are fully described, and given a set of obstacles whose shapes, positions, and orientations in space are likewise known, the task is to find a continuous path for the object from the initial to the target position while avoiding collisions with obstacles along the way. An important assumption used in the model is 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. Reaching a solution means either finding a path or concluding in finite time that no path exists. Given the fact that the solution is always conceptually feasible, cases of arbitrary complexity can be considered. Another apparent advantage to dealing with complete information is that various optimization criteria (finding the shortest path, or the minimum-time path, or the safest path, etc.) can be introduced easily.

[1] A good survey of the work on provable algorithms for the Piano Movers problem can be found in [17]; specialized maze search algorithms are considered in [15].

The price to be paid for dealing with perfect information is quite high, in terms of both the computational load and ensuing limitations. The requirement that all the surfaces be algebraic comes from the fact that the computational vehicle used in the Piano Movers model is that of the *connectivity graphs*: the problem in question is first reduced to a finite list of analytical entities (polynomial patches, sides of the polyhedra, space cells free of obstacles, etc.), from which an appropriate connectivity graph is produced. Then, a path is declared to exist if the start and the destination nodes on the graph are connected. The computational complexity of the problem is measured in terms of the structure and processing characteristics of the corresponding connectivity graph.

From the application standpoint, unless there is reason to believe that the obstacle boundaries are algebraic, an appropriate approximation must be performed before the connectivity graph can be built. Because the approximation itself depends on considerations that are secondary to the path planning problem (for example, the accuracy of the presentation of actual obstacles by polygons, or—a conflicting criterion—computational costs of processing the resulting connectivity graph), it can introduce problems of its own. For example, the operation of approximating nonlinear surfaces with linear constraints itself requires time exponential in the prescribed accuracy of approximation [21]. Also, the space of possible approximations 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 positions of the nodes of the approximated surfaces and eventually in the generated paths.

Measuring the computational burden in terms of complexity of the connectivity graphs can create peculiar situations when the derived computational complexity of a given task contradicts our intuitive notion of problem complexity. Consider, for example, a circular obstacle A shown in Fig. 1a. Assume that the algorithm to be used requires polygonal obstacles, and so the obstacle is first approximated—say, by one of the polygons B or C (Fig. 1). Now, according to Piano Movers algorithms, planning a path around the obstacle C is computationally more difficult than

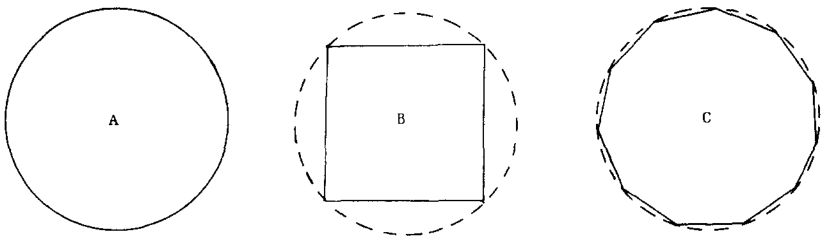

FIG. 1. Which of the three obstacles, A, B, and C, would be easier to pass around?

planning a path around the obstacle B, because of the greater number of nodes in C. Moreover, in the limit, increasing the accuracy of the polygon approximation makes the computational burden go to infinity. On the other hand, from the human and from the robotics control viewpoints, walking around the circle A is actually easier than walking around the obstacles B or C, because the latter require different decision-making along the edges and at the nodes of the obstacles. Also, from the dynamics standpoint, there is an undesirable sharp change in the velocity vector at the corners of the obstacles B and C.

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 to remove the requirement that the obstacles be stationary. The fact that processing is distributed over time and little information has to be processed at each step makes the computational burden insignificant. Most of the work on this model has been proceeding under an assumption that the incoming partial information about the environment is of local character—for example, robot sensors can provide information only about the robot's immediate surroundings, and so their operation naturally falls into this category. Replacing global information by local information 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].

One problem of dealing with incomplete information is that, because of the dynamic character of the incoming (sensor) data, 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 it is locally, or how "reasonable" it appears 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. Continuous path planning requires choosing a preferred direction of motion at each step. This dictates either availability of global information, or a limit on the number of alternatives. Because the information about the obstacles is local, the former option is not available. Instead, by imposing a constraint on the problem dimensionality, one limits the number of alternatives available to the automaton at each step.

1.2. *Path Planning With Complete Information: Previous Work*

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.[2] 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 to the original *work space*—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 cylinder 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 of its obvious relation to path generation and coordinate transformation problems of robot arms with multiple degrees of freedom. Recently, new approaches for this version were considered in [9, 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.

[2] 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.

1.3. *Path Planning With Incomplete Information: Previous Work*

Works related to the model with incomplete information have come primarily from the studies on autonomous vehicle navigation; so far, they have been limited to various heuristics. In [11–13,23,25,26], 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. Often, the navigation problem is treated as a hierarchical problem [12,23], with the upper level concerned with global navigation for which the information is assumed available, and the lower level doing local navigation based on sensory feedback. A heuristic procedure for moving a robot arm among unknown obstacles is described in [24].

Although most of these algorithms are intended for use in unstructured environments, no work has been done on handling more "natural" obstacles of arbitrary shape, or the minimum resources (sensory feedback, memory, etc.) that are required to guarantee reasonable navigation. Consequently, the range of applicability of various heuristic procedures for robot motion planning is not clear. It should also be mentioned that, having no theoretical assurances of the algorithm convergence, many heuristic procedures actually rely on so-called common sense which, in turn, is founded on the assumption that humans are good at orienting in space and at solving geometrical search problems. This assumption is questionable, however. There are many indications that human space orientation capabilities are rather limited.[3]

The only nonheuristic algorithm for path planning in an uncertain environment that this author is aware of is the Pledge algorithm described by Abelson and diSessa [16]. The algorithm is shown to converge; no performance estimates are presented. The algorithm addresses, however, a problem different from ours—namely, how to escape from an arbitrary

[3] This was also the conclusion reached in the experiments conducted at the Robotics Laboratory at Yale University. The experiments involved two specially designed computer games in which the subject had to move on the graphics screen a little autonomous vehicle or a planar robot arm manipulator, from the starting to the target position, in a scene filled with invisible obstacles. The system simulates tactile sensing—when the robot hits an obstacle, the subject can see a little part of the obstacle around the point of contact. When lacking global information and obvious directional clues, human subjects are confused and lose their sense of orientation. Although no direct experiments with vision have been done, the results suggest that in the orientation task vision would be of limited help.

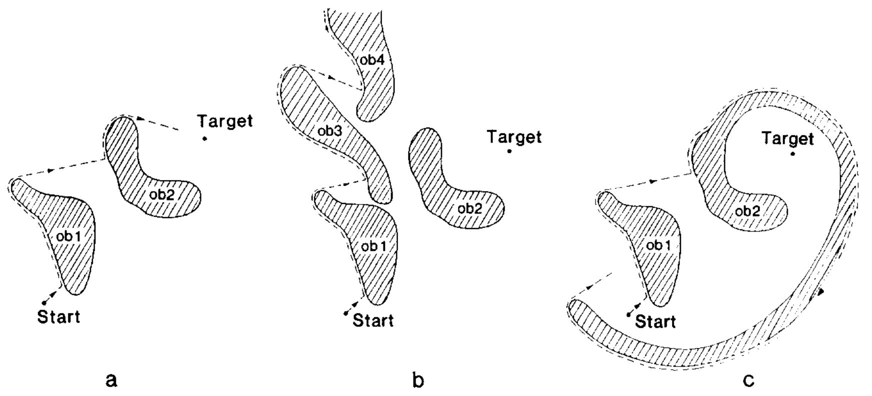

FIG. 2. This intuitively reasonable algorithm (walk toward the target whenever you can) will often work, (a), but it does not guarantee termination, (b), (c).

maze—and cannot be used to reach a specific point inside or outside the "maze."

The issue of convergence of motion planning algorithms with uncertainty is not a trivial one. As the following example shows, a seemingly reasonable strategy can suddenly produce rather disappointing results. Consider this algorithm[4] (see Fig. 2a):

1. Go directly to the target until one of the following occurs:
   (a) The 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) The target is reached. The procedure stops.
   (b) The direction toward the target clears. Go to Step 1.

As one can see, depending on the scene, this strategy may or may not succeed: in the scene shown in Fig. 2b, 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; in the scene of Fig. 2c, the strategy will result in infinite looping. It can be shown that attempts to fix this scheme with minor modifications (say, alternate, "left-right-left . . . ," instead of turning always left at an obstacle, etc.) can make things even worse.

1.4. *Dynamic Path Planning*

The approach of *dynamic path planning* is based on the model with incomplete information. In terms of the available information, the specific model it uses can be viewed as diametrically opposite to the Piano Movers

[4] The algorithm has frequently been suggested to me at various meetings.

model: instead of the full information about the obstacles assumed in the latter model, no information about the obstacles is given to the automaton, and no constraints are imposed on the shapes of the obstacles. The only available input information includes the automaton's own coordinates and those of the target. The automaton's capability for learning about the environment is limited to the local information provided by the automaton's "sensors." Although various noncontact sensors can fit the model (ultrasound range finders, vision systems, etc.), to understand the limits of the approach, it helps to consider the least informative "tactile sensor." In other words, in this version the automaton learns about the presence of an obstacle only when it hits it.

The purely local information assumed in the model is indeed sufficient for designing algorithms with guaranteed convergence. Using these algorithms, the automaton is continuously analyzing the incoming information about its surroundings and is continuously (dynamically) adjusting its path—hence the name of the approach. The approach is somewhat similar to that 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 appear. Since no reduction to a discrete space takes place, all points of the scene are available for the purpose of path planning.

The key idea of the approach is to reduce the problem of path planning to maneuvering a point automaton around simple closed curves representing the obstacles and lying in appropriate two-dimensional manifolds. When the automaton meets such a curve it has only two choices for going around it—turning left or turning right. Another important feature of the topology of a simple closed curve is that, no matter what direction is chosen for walking around the obstacle, it will eventually bring the automaton back to the starting point. Essentially, we 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]. The algorithms for a point automaton with a "tactile sensor" (called below *basic algorithms*) become the foundation for a series of path planning algorithms for an autonomous vehicle and various arm manipulators.

In the case of the autonomous vehicle, the surface along which it travels is homeomorphic to a plane, and so no additional constructs are needed. The final dimensions of the automaton can easily be accommodated since no assumptions about the obstacles are made and every point of the automaton's body is assumed to be sensitive to contact with an obstacle (see the next section). Handling the orientation of the automaton along its way is more difficult, because adding orientation increases the dimensionality of the parameter space from two to three, and consequently the problem cannot be reduced to maneuvering around simple closed curves. If one assumes that the gaps between the obstacles are such that the automaton can pass them without changing its orientation, then the basic algorithms can be used to design converging path planning procedures.

In the case of a robot arm manipulator, every point of the robot body is subject to collision. Similar to the human arm, each point of the robot "skin" is sensitive to contact (direct or at a distance, depending on the sensory system) with an obstacle. Again, as with the autonomous vehicle, assume that the proper orientation—in this case, of the arm end effector—can be done later, when the arm arrives in the vicinity of the target position. Then, the planning is limited to that of position planning, or *gross motion planning*. For the two-dimensional case, gross motion involves two degrees of freedom, and for the three-dimensional case it involves three degrees of freedom.

First, the problem is transformed into that of moving a point in the *image space*. For a planar arm, the image space presents a two-dimensional manifold in three-dimensional space. For a three-dimensional arm, the image space is forcibly reduced to a two-dimensional manifold, using the natural constraints imposed by the arm kinematics (e.g., the fact that the arm links are connected sequentially and that the arm base is fixed). The next step is to show that, given the kinematics of a specific arm manipulator and independent of the shape of the actual obstacles in the *work space,* the images of the obstacles in the image space form simple closed curves. Once this is done, the basic algorithms can be utilized and their convergence guaranteed provided that they are modified to take into account the topology of the image space.

In studying the performance characteristics of the motion planning algorithms, it is instructive to separate the issue of the computational complexity of the algorithms from the issue of the quality of the produced paths. In the Piano Movers model, the former issue is of utmost importance, whereas the latter issue is rarely addressed directly; the reason for this is that, within the framework of the model, the best path can in principle always be produced. In the model with incomplete information, however, the importance of the two issues is reversed. Because only very little information is available at each step, the computational burden of the algorithms is rather modest; on the other hand, the quality of the produced paths can vary greatly, and so the path efficiency of the algorithms is very important.

Because the model used in dynamic path planning is continuous, the criteria typically used for evaluating the 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.

In the following sections, dynamic path planning is reviewed in more detail. Most of the surveyed results have been presented in [28–40]. For proofs of specific statements and other details, the reader is referred to an appropriate publication. The simplest case of planar motion for a point automaton with the tactile sensor [28,33,40] is considered in Section 2. For this case, the worst-case lower bound on the length of generated paths is given. The bound applies to any path planning algorithm with uncertainty and presents a powerful means for comparing performance of various algorithms. This is followed by two *basic algorithms* for path planning, for which upper bounds on their performance are given. Finally, some details related to handling the final dimensions of the automaton and the noncontact sensors, such as vision, are discussed.

In Section 3, the problem of dynamic path planning is formulated for the case of two-dimensional arm manipulators, and two examples of how the methodology can be used in designing path planning algorithms for arms of different kinematics are shown; various parts of this work are described in [29,32,37,38]. Section 4 describes briefly how the method can be further generalized to simple three-dimensional arm manipulators [34,39]. Section 5 addresses the experimental work, both in computer simulation and in real robot systems [35,36].

2. Moving a Point Automaton in the Plane

2.1. *Model*

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 a finite number of 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. The automaton is a point; thus, any opening between two distinct obstacles is passable. The only information the automaton is provided with by its sensors is (1) its current coordinates, (2) the fact of contacting an obstacle. The automaton is also given the position of the

Target, and so it can always calculate its direction toward and distance from the Target. The memory available for storing data or intermediate results is limited to a few computer words. The motion capabilities of the automaton include three possible actions: move toward the Target on a straight line; move along the obstacle boundary; stop.

DEFINITION 2.1. A *local direction* is a once-and-for-all determined 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 the automaton meets an obstacle, there is no information or criteria which could help it decide whether it should turn left or right to go around the obstacle. For the sake of clarity, assume that the local direction of the automation is always *left* (as in Fig. 2).

DEFINITION 2.2. The automaton is said to *define a hit point H* on the obstacle, when, while moving along a straight line toward the Target, the automaton 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, Fig. 3).

If the automaton 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—the automaton 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.

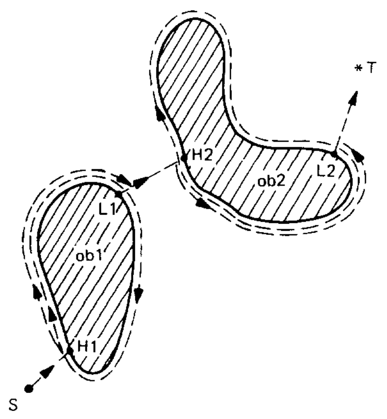

FIG. 3. Algorithm Bug1.

Notation. D is the (Euclidean) distance from the Start to the Target; $d(A, B)$ is the distance between points A and B of the same scene (thus, $d(\text{Start, Target}) = D$); $d(A)$ 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 the automaton on its way to the Target; P is the total length of the path generated by the automaton on its way from the Start to the Target; p_i is the perimeter of the ith obstacle met by the automaton.

2.2. *The Lower Bound for the Path Planning Problem*

This lower bound, formulated in Theorem 2.1 below, determines the best performance that can be expected in the worst case from any path planning algorithm operating within the framework of our model [33,40]. The bound is formulated in terms of the length of the path generated by the automaton on its way from the point Start to the point Target. The bound is a powerful means for measuring performance of various path planning procedures.

THEOREM 2.1. *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 such that*

$$P \geq D + \sum_i p_i - \delta,$$

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.

One point is worth mentioning. The information requirements of the model above are formulated such that the same model could be used in different statements and algorithms. Depending on the case in question, the necessary and sufficient information can be less than that assumed in the model. Specifically, for Theorem 2.1 to hold, the constraints on the information available to the automaton can be relaxed significantly; the required constraint is that at any time moment the automaton does not have complete information about the scene.

2.3. *First Basic Algorithm: Bug1*

The procedure Bug1 [33] is to be executed at any point of a continuous path. Figure 3 demonstrates the behavior of the automaton. When meeting an ith obstacle, the automaton defines a hit point H_i, $i = 1, 2, \ldots$. When leaving the ith obstacle, to continue its travel toward the Target, the automaton defines a leave point L_i; initially, $i = 1$; $L_0 =$ 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 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. The test for target reachability mentioned in Step 3 of the procedure is explained later in this section. 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 reach the Target, 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. Based on the content of the registers R_2 and R_3, determine the shorter way along the boundary to L_i, and use it to reach L_i. Apply the test for target reachability. If the Target is not reachable, the procedure stops. Otherwise, set $i = i + 1$ and go to Step 1.

LEMMA 2.1. *Under Bug1, after the automaton leaves a leave point of an obstacle in order to continue its way toward the Target, it never returns to this obstacle again.*

The lemma guarantees that the algorithm will never create cycles. A corollary to the lemma suggests that, under Bug1, independent of the geometry of an obstacle, the automaton defines on it not more than one hit and not more than one leave point.

To produce an upper bound on the length of paths generated by Bug1, an assurance is needed that on its way to the Target the automaton always encounters only a finite number of obstacles. This is not obvious since, while following the algorithm Bug1, the automaton 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. Hence the following lemma.

LEMMA 2.2. *Under Bug1, the automaton can meet only a finite number of obstacles on its way to the Target.*

A corollary to the lemma states that the only obstacles that can be met by the automaton are those which intersect the disc of radius D centered at the Target. Together, Lemma 2.1, Lemma 2.2, and the corollary guarantee convergence of the algorithm Bug1. The following theorem gives an upper bound on the length of paths produced by Bug1.

THEOREM 2.2. *The length of a path produced by the procedure Bugl will never exceed the limit*

$$P = D + 1.5 \cdot \sum_i p_i, \tag{2}$$

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

It can be shown that the following necessary and sufficient condition holds: if the automaton, after having arrived at the point L_i in Step 3 of the algorithm, discovers that the straight line (L_i, Target) crosses some obstacle at point L_i, then the Target is not reachable—either the Start or the Target point is *trapped* inside the ith obstacle. Based on that, the test for target reachability used in Step 3 of the procedure is formulated as follows.

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

Analysis of the procedure Bugl shows that the requirement that the automaton has to know its own coordinates at any instance can be eased. It suffices if the automaton is capable of positioning itself at the circle of a given radius centered at the Target. In other words, what the automaton needs in order to use Bugl is only 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, the register R_1 stores the minimum distance itself. In Step 3 of Bugl, then, the automaton can reach point Q_m by comparing its current distance from the Target with the content of register R_1. If more than one point of the current obstacle lies 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.

2.4. *Second Basic Algorithm: Bug2*

The procedure Bug2 is executed at any point of a continuous path. Unlike the procedure Bugl, in Bug2 the automaton can meet the same obstacle i more than once, and consequently generate more than one hit and leave point; actually, the algorithm has no way of distinguishing between different obstacles. Because of this, the subscript i will be used only when referring to more than one obstacle; in addition, the super-

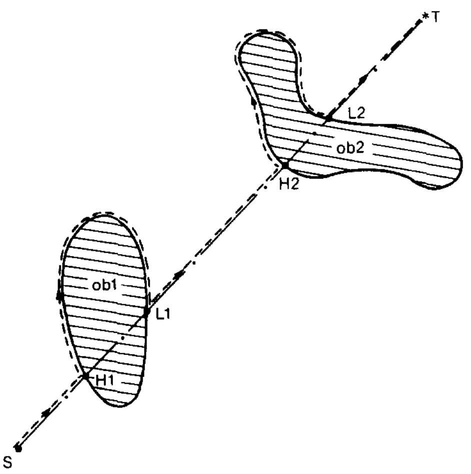

FIG. 4. Automaton's path under the procedure Bug2.

script 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 2b and 2c of the procedure is explained later in this section. One can follow the procedure using an example shown in Fig. 4. The algorithm consists of the following steps.

1. From 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.
   (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.

The relationship between obstacle perimeters and the length of paths generated by Bug2 is not as simple as in the case of Bug1. Note that in Bug1 the perimeter of an obstacle met by the automaton is covered at least once, and never more than 1.5 times. In Bug2, however, the range is

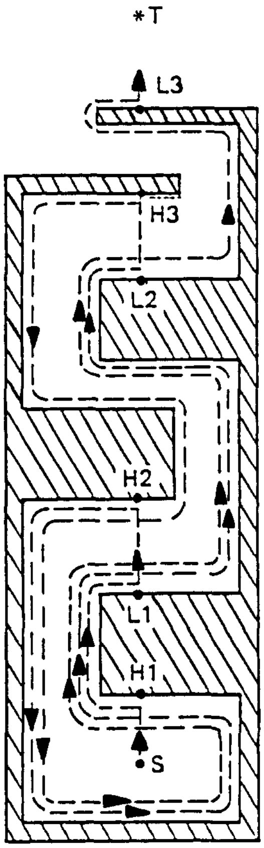

FIG. 5. Automaton's path in a maze-like obstacle, procedure Bug2. The obstacle complexity is measured by the number of times, n_i, the straight line (S, T) crosses it; here, n_i = 10. At most, the path passes one segment, ($H1$, $L1$), three times; that is, there are at most two local cycles in this path.

much wider. A path segment around an obstacle generated by the automaton is often shorter than the obstacle perimeter (as in Fig. 4). Sometimes, when a straight line segment of the path meets the obstacle almost tangentially and the automaton goes around the obstacle in a "wrong" direction, the path can actually be equal to the obstacle's full perimeter. Finally, as Fig. 5 demonstrates, the situation can become even worse, and the automaton may have to pass along some segments of a maze-like obstacle more than once.[5] The performance of Bug2 is analyzed in the following statements.

[5] A procedure that combines positive characteristics of both algorithms, while compensating for tneir negative sides, is described in [28,40].

LEMMA 2.3. *Under Bug2, on its way to the Target the automaton can meet only a finite number of obstacles.*

COROLLARY. *The only obstacles that can be met by the automaton under the algorithm Bug2 are those which intersect the disc of radius D centered at the Target. Moreover, the only obstacles that can be met by the automaton are those that intersect the straight line (Start, Target).*

DEFINITION 2.3. For a given local direction, a *local cycle* is created when the automaton has to pass some point of its path more than once. (In the example in Fig. 4, no cycles are created; in Fig. 5, the path contains local cycles.)

DEFINITION 2.4. The term *in-position* refers to such a mutual position of the pair of points (Start, Target) and a given obstacle where (i) the straight line segment (Start, Target) crosses the obstacle boundary at least once, and (ii) either the Start or the Target lie inside the convex hull of the obstacle. The 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 points Start and Target, creates an in-position condition; otherwise, the scene presents an out-position case.

Let n_i be 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 (Fig. 4, obstacle ob2). It can become more complicated if $n_i > 2$. In Fig. 5, 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 2.4. *Under Bug2, the automaton will pass any point of the ith obstacle boundary at most $n_i/2$ times.*

The lemma thus guarantees that the procedure terminates, and puts 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 2.3. *The length of a path generated by Bug2 will never exceed the limit*

$$P = D + \sum_i \frac{n_i p_i}{2}, \tag{3}$$

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

The theorem suggests that in some special scenes the procedure Bug2 may force the automaton to go around an obstacle any (large albeit finite) number of times. An important question is, then, how typical such scenes are, and, in particular, what characteristics of the scene influence the length of the path. The following theorem and its corollary suggest that the mutual position of the Start point, the Target point, and the obstacles can affect the path length rather dramatically.

THEOREM 2.4. *Under the procedure Bug2, in the case of an out-position scene the automaton will pass any point of the obstacle boundary at most once.*

In other words, if the mutual position of the obstacles and of the points Start and Target corresponds to an out-position scene, the estimate on the length of the path for the procedure Bug2 reaches the lower bound (1). If 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*

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

and, on the average,

$$P = D + 0.5 \cdot \sum_i p_i, \tag{5}$$

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

A necessary and sufficient condition similar to that presented for the procedure Bug1 can be shown for Bug2; the condition helps establish the test for target reachability incorporated in the algorithm.

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

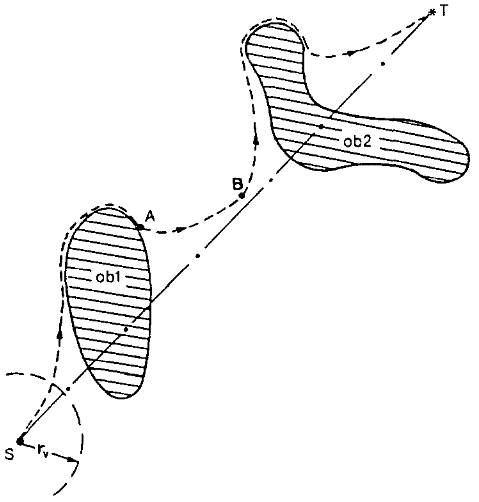

FIG. 6. Automaton's path in the scene of Fig. 4 when additional (vision) information is available.

2.5. *Handling Noncontact Sensors and the Automaton's Dimensions*

Consider a case where, instead of tactile feedback, the automaton is equipped with a sensor system that provides it with information within a disc of radius r_v ("radius of vision") centered at the current position of the automaton. Thus, within the disc r_v, the automaton can observe obstacles or the nearby Target, and estimate distances to them. Various sensors provide this capability: vision systems, ultrasonic and infrared proximity sensors, time-of-flight range finders, etc.

To assure convergence, the sensors have to be incorporated properly in the path planning procedure. For example, for the algorithm Bug2, an assurance is needed that the automaton will always be able to come back to the straight line (Start, Target). With this in mind, incorporating a "vision" sensor feedback in Bug2 is done as follows. At its current position, the automaton reconstructs "mentally," in the area it can observe, 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 of length r_v exactly. Then, the automaton makes a step in the chosen direction. The operation is repeated at each point of the path, resulting in a curved path which smooths out the path of a "blind" automaton (Fig. 6). With such a modification, convergence of the algorithm Bug2 is preserved.[6] As one would expect,

[6] Incidentally, the segment (A, B) of the path in Fig. 6 forms a curve called *tractrix*; this curve had been studied independently by Leibnitz and Huygens in the late 17th century. If

increasing the radius r_v produces shorter paths. In the limit, with r_v going to infinity, locally optimum paths are produced; this results from the simple fact that the distance between the current position of the automaton and its next intermediate visible goal is always covered along a straight line.

When obstacles interfere with vision, the generated straight line segment becomes shorter, and, in theory, can eventually be reduced to zero by a "bad" set of obstacles. Consequently, all the estimates for path lengths developed for the automaton with tactile sensing remain true for the case with vision.

Now, consider an automaton of finite dimensions. Instead of the single feedback sensor of a point automaton, the whole body of the "finite" automaton is covered with sensors, so that every point of the body is capable of detecting an obstacle. Assuming that no changes in the automaton orientation along the path is allowed, the algorithms described above can be used and their convergence is assured. This is primarily because nowhere in the algorithms are the dimensions, shape, or positions of the obstacles used.

For the actual algorithm, slight modifications in the definitions will be needed. Define some point of the automaton—say, one of its corners or its center of gravity—as the *core point*. Define the motion of the automaton along the line (S, T) as that of moving the core point along (S, T). A hit point is then defined as a point on the obstacle boundary which the automaton, while moving along (S, T), meets first. A leave point—for example, in the algorithm Bug2—is defined as a point on the obstacle boundary at which the automaton, while 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 automaton will differ from that of a point automaton. It is possible, for example, that, given a set of obstacles and S and T positions, a finite automaton will produce local cycles whereas a point automaton will not.

3. Moving a Planar Arm Manipulator

3.1. *General*

In this and the next sections, we will be concerned with robot arm manipulators consisting of links connected by sliding and revolute joints.

one puts one end of a stick of the length r_v at the point A and its second end at the point where it meets the line (S, T) and then pulls the second end of the stick along (S, T), then the first end of the stick will move along a tractrix.

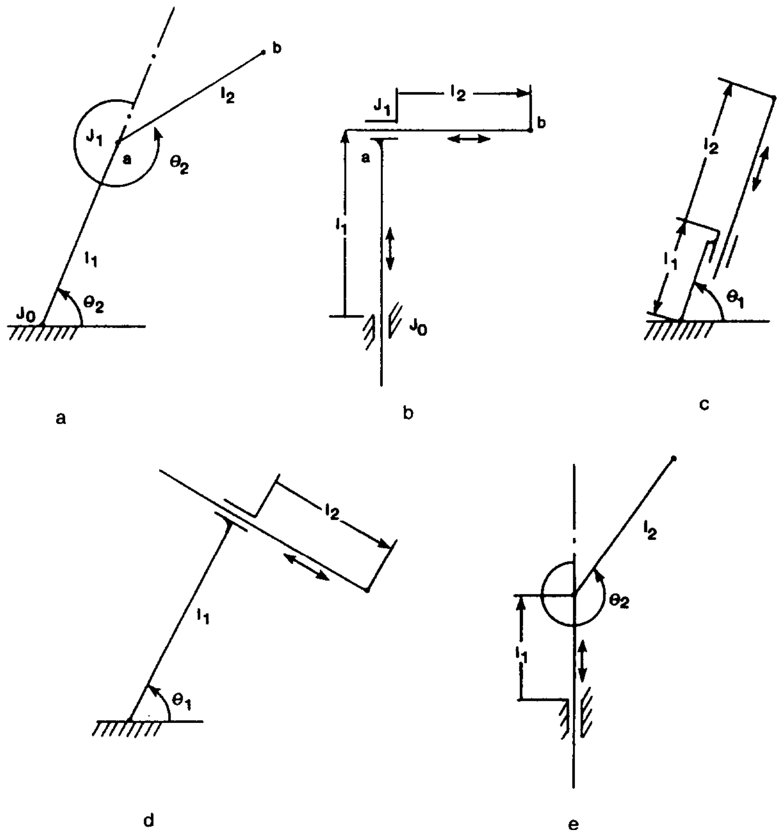

FIG. 7. Five kinematically distinct planar robot arms with revolute and sliding joints: (a) Arm 1; (b) Arm 2; (c) Arm 3; (d) Arm 4; (e) Arm 5.

As was mentioned above, realization of the gross motion for a three-dimensional arm requires a minimum of three degrees of freedom (that is, three links and three joints), and for a planar arm it requires two degrees of freedom. Out of 36 theoretically possible three-dimensional configurations of arms with revolute and/or sliding joints, 12 are kinematically useful and distinct [27]. The rest are not admissible, either because they degenerate into one- or two-dimensional cases, or because they are equivalent to some others. Out of these 12 types, only the following five combinations are meaningful for the planar case: two revolute joints (an articulated arm, Fig. 7a); two sliding joints (typically referred to as a Cartesian arm, Fig. 7b); a revolute joint followed by a parallel sliding joint (Fig. 7c); a revolute joint followed by a perpendicular sliding joint (Fig. 7d); and a sliding joint followed by a revolute joint (Fig. 7e).

The basic algorithms presented in the previous section can be shown to be applicable to these arm manipulators and their convergence is preserved provided that important modifications of the path planning procedure are introduced for each of the types. As an example, the method will be demonstrated on two of these arms (Figs. 7a and 7e).

The arm's objective is to move from the position Start (S) to the position Target (T); both S and T lie within the arm *work space* (*W-space*). Only continuous motion of the robot links is allowed. The boundaries of the W-space are defined by the arm configuration and by the dimensions of the arm links. The arm body consists of two links, l_1 and l_2, and two joints, J_0 and J_1 (Figs. 7a and 7e). Joint J_0 is fixed and is the origin of the reference system. Although the approach is applicable to links of arbitrary shape, assume for presentation purposes, that each link is a segment of a straight line; the lengths of the links are l_1 and l_2, respectively. Depending on the arm configuration, the length of a link may be constant or variable.

An *arm solution* (or arm position) corresponding to a given point P in the W-space is defined by a pair of variables, *joint values*, which are either angles (as in Fig. 7a), or linear translations (as in Fig. 7b), or both (as in Fig. 7e). An equivalent presentation for the same solution P is given by the coordinates of the link endpoints, a_p and b_p; b_p also designates the position of the arm endpoint.

Each of the obstacles present in the W-space is a simple closed curve. The shapes of the obstacles are not known and are not constrained. The number of obstacles in the W-space is finite. Any circle of a limited radius or a straight line passing through the W-space has a finite number of intersections with obstacles. Being rigid bodies, obstacles cannot intersect each other; two or more obstacles may touch each other, in which case for the arm they effectively present one obstacle. At any position of the arm with respect to a set of obstacles, at least some arm motion is assumed to be feasible.

The only information available to the arm includes its current position and the target position T. The starting position S is known to be feasible. Because of the obstacles, position T may or may not be feasible; also, it may or may not be reachable from position S. The arm body is covered with sensors that make it sensitive to contact with an obstacle; again, assume that the sensors are tactile. The arm is capable of performing the following actions: (i) move the arm endpoint through a prescribed simple curve, called *main line* (*M-line*), connecting S and T; (ii) when the arm body contacts an obstacle, identify the points of the arm body that are in contact; (iii) follow the obstacle boundary.

Realizing the first of these actions may require computing coordinates of consecutive points along the M-line and transforming them into the corresponding joint values (using, for example, well-known procedures of inverse kinematics). The sole purpose of the second operation is to provide information needed to pass around the obstacle. Such identification is a local operation that does not require global information about the environment. (Recall that a blindfolded person can easily identify the point of his body that touched an object.) When the arm endpoint follows an obstacle boundary up to the W-space boundary, it is not clear whether at the boundary the arm is still in contact with the obstacle. To avoid this limit case, assume that no point of the W-space boundary may be a point of contact between an obstacle and the arm endpoint.

DEFINITION 3.1. *Passing around an obstacle* is a continuous motion of the arm during which the arm is in constant contact with some obstacle.

Because of the arm/obstacle interaction, some areas of the W-space, though not occupied by the actual obstacles, may be inaccessible to the arm endpoint. Such an area creates a *shadow* of the obstacle (see, e.g., the shaded area behind the obstacle A, Fig. 8a); for the arm endpoint, a shadow presents as real an obstacle as points of the actual obstacle.

DEFINITION 3.2. A *virtual obstacle* X is an area (or areas) in W-space, no points of which can be accessed by the arm endpoint because of the arm's possible interference with the actual obstacle(s) X. A virtual obstacle consists of the corresponding actual obstacles and of their shadows.

DEFINITION 3.3. A *virtual line* is a curve in W-space which the arm endpoint follows when the arm is passing around the obstacle.

The *image space* (*I-space*) is a representation space in which the arm is shrunk to a point.[7] Any trajectory (path) and any virtual obstacle has its corresponding image in I-space. To define uniquely the image of a virtual line in I-space, the corresponding arm positions have to be added.

DEFINITION 3.4. A *virtual boundary* is a curve in I-space which represents the image of the corresponding virtual line. The virtual boundary separates the virtual obstacle from the rest of the I-space.

In the path planning procedure, the arm will attempt to move its endpoint from point S to point T along the M-line. When, during the motion, some point of the arm body meets an obstacle, in I-space this corresponds to a point of intersection between the M-line image and the obstacle virtual boundary. The point of intersection is said to define a *hit point, H*. At the hit point, the arm has a choice of moving along the virtual boundary in one of two directions: if facing the obstacle at the hit point, these directions are "right" and "left." The direction chosen for passing

[7] To avoid confusion, we choose not to use the term "configuration space" employed in a number of works (see, e.g., [2]); configuration space is meant to represent some Euclidean space, whereas I-space represents a manifold. For example, defining a metric in I-space may be quite different from that of the configuration space.

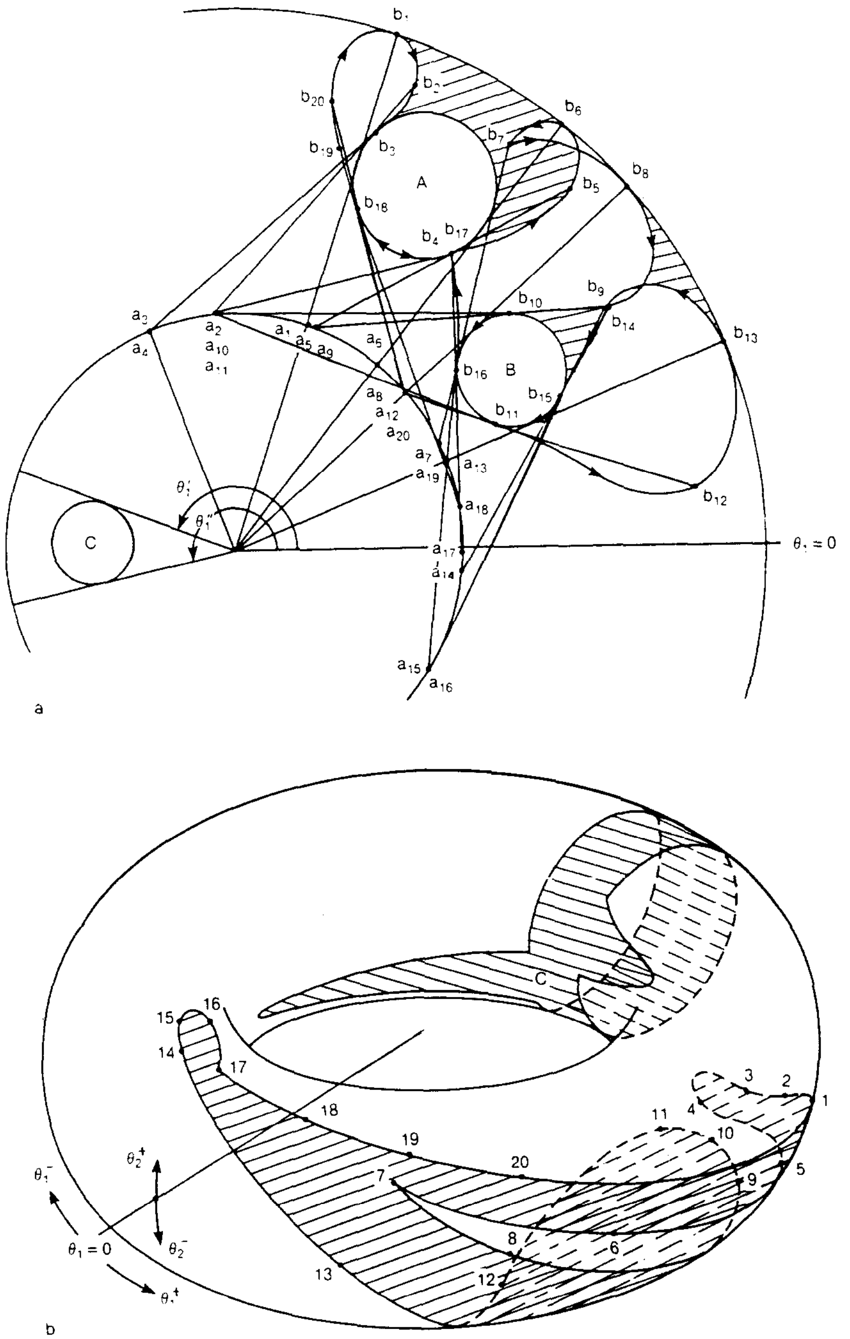

FIG. 8. Arm 1. (a) W-space. A, B, C—actual obstacles. (b) I-space images of the same obstacles.

around the obstacle is called the *local direction*. While following the virtual boundary, the arm may meet the M-line again. If this occurs at a distance from T which is shorter than that from the hit point to T (as measured along the M-line) then the arm is said to define a *leave point, L*. Hit and leave points come in pairs, $(H^j, L^j), j = 1, 2, \ldots$. Denote Start $= L^0$, with no corresponding H^0.

A pivotal point in the design of the dynamic path planning algorithms for an arm manipulator is the proof that no matter how complicated the virtual obstacles are, the corresponding virtual boundaries in I-space always form simple (the *proof of simplicity*) and closed (the *proof of closedness*) curves. Once this is done, the mechanisms of the basic algorithms for moving a point in the plane presented in the previous section can be utilized, and their convergence is assured. Additional modifications in the path planning procedure might be needed, to reflect the topology of the I-space in question. Specifically, one question to address is how many simple closed curves can the image of an actual obstacle form in the corresponding I-space.

3.2. *Arm 1: Two Revolute Joints*

The outer boundary of the W-space of this arm (Fig. 8a) presents a circle of radius $(l_1 + l_2)$; its inner boundary creates a circular "dead zone" around the origin of radius $|l_1 - l_2|$. Both joint values, angles θ_1 and θ_2, can increase or decrease indefinitely, $\theta_k = \theta_k \pm n \cdot 2\pi$; $n = 0, 1, \ldots$; $k = 1, 2$. In general, for any position of the arm endpoint in the W-space, except for points along the W-space boundaries, there are two arm solutions.

The combination of the arm kinematics and the obstacles creates a complicated pattern of areas inaccessible to the arm endpoint. Even with full information on the obstacles, these areas would present difficulties for direct analysis. When the arm is passing around an obstacle, it may create a single shadow or more than one subshadows. The latter are shown in Fig. 8a as two disconnected shaded areas behind the circular obstacle B. Note that with the same position but a smaller diameter of B, or with B of the same diameter positioned slightly further from the origin, both subshadows would merge into a single shadow. Obstacles may interact in creating shadows; this occurs, for example, in Fig. 8a (obstacles A and B, point b_7; here, point b_p is the end of a (straight line) segment of the length l_2 which starts at a_p and is tangential to the obstacle). Therefore, as far as the arm is concerned, obstacles A and B in Fig. 8a present one virtual obstacle. The virtual line of this obstacle is defined by the points b_p, $p = 1, 2, \ldots, 20$, and its virtual boundary is defined by the corresponding link positions (a_p, b_p). It is easy to see how a set of quite simple actual obstacles can produce extremely complicated virtual obstacles and virtual lines.

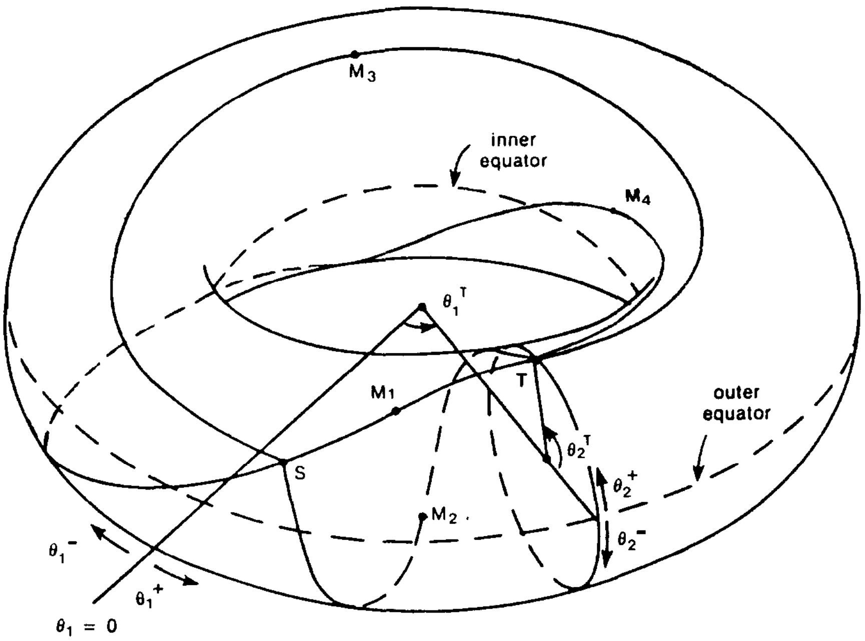

FIG. 9. Image space of Arm 1.

A set of two independent angular variables can be represented by a common torus, with a pair of values of the variables uniquely defining a point on the surface of the torus. The I-space of the Arm 1 (Fig. 9) is represented by the surface of a torus whose two independent variables are θ_1 and θ_2. Since one point in the W-space corresponds, in general, to two arm solutions, this creates two image points in I-space. The following two statements form the base of the path planning procedure for Arm 1.

THEOREM 3.1. *Any virtual boundary in the I-space of Arm 1 can consist only of simple closed curves.*

LEMMA 3.1. *Any virtual boundary in the I-space of Arm 1 can be formed by no more than two simple closed curves.*

The proofs of these statements [37] are based on the Jordan Curve Theorem and on the topology of a common torus (specifically, on the *first connectivity number* of the torus [22]). In Fig. 8b, images of the actual obstacles of Fig. 8a are shown. Note that in I-space the obstacles A and B form a single closed curve (these are called *Type I* obstacles) whereas the obstacle C forms a band-like structure limited by two simple closed curves (these are called *Type II* obstacles). The path planning procedure has to be able to recognize and appropriately handle each type of obstacle. This is necessary because if the arm, while passing around an obstacle, completes a full circle without ever meeting the M-line, it may mean that the target cannot be reached, or that "somewhere" there is another closed curve belonging to the same virtual boundary, which has not yet been investigated but which has to be investigated before any conclusions are drawn.

To do the obstacle type recognition, two counters, C_1 and C_2, corresponding to the angles θ_1 and θ_2, respectively, are used. When the arm is traveling in free space, the contents of the counters are zeros. Once the arm hits an obstacle, both counters are turned on, and, while the arm follows a closed curve of the virtual boundary, each counter integrates its angle, taking into account the sign. After completing a closed curve, the content of the counter C_k must be $n_k \cdot 2\pi$, $|n_k| = 0, 1, 2, \ldots$ (Fig. 8b). For a given closed curve, the resulting values of the pair (C_1, C_2) define its *arm joints range* (or, simply, *range*) (n_1, n_2). The range defines the type of the obstacle as follows: for a Type I obstacle, its range is (0, 0); for a Type II obstacle, its range is (n_1, n_2), with either $n_k = 0$ and $|n_{3-k}| = 1$, or $|n_k| = 1$ and $|n_{3-k}| = 1, 2, \ldots$; $k = 1, 2$.

To locate the second closed curve of a Type II obstacle, the notion of *complementary* M-lines is introduced. Although no restrictions are imposed on the choice of the M-line, for specificity, a straight line in the plane of variables θ_1 and θ_2 is being used. In I-space, this M-line forms a line segment corresponding to an imaginary tight thread connecting points S and T. There are, however, three more ways to have a tight thread between S and T on the torus surface (Fig. 9). These three are obtained by switching the direction of change of one or both angles θ_1 and θ_2. In Fig. 9, the segment M_1 corresponds to the global minimum, and the other three segments, M_2, M_3, and M_4, to local minima of the Euclidean distance (in the plane (θ_1, θ_2)) between the points S and T. Two segments, M_p and M_q, are said to be *complementary over the angle* θ_k if the θ_k components of M_p and M_q add to 2π. In the algorithm, to find the second closed curve of a Type II obstacle, an M-line complementary to the current M-line is selected; altogether, no more than two M-lines are ever used to produce the path.

The test for target reachability is based on the following facts. For a Type I obstacle, completing a closed curve of the obstacle virtual boundary without ever meeting the M-line suggests that the target is "inside" the obstacle (e.g., the obstacle forms a ring in the torus surface, with the target point inside the ring) and thus cannot be reached. A similar conclusion is made for a Type II obstacle if two closed curves have been passed without ever meeting the M-line.

Similar to the case of an autonomous vehicle, the algorithm for Arm 1 can be shown to create *local cycles* in some special cases when the relative positions of the points S, T, and of the obstacles form in I-space a combination similar to the in-position case (see the previous section). A local cycle is created when the arm image point in I-space comes back to a previously defined hit point and at this moment the content of one or both of its counters, C_k, is differen from $n_k \cdot 2\pi$, $|n_k| = 0, 1, 2, \ldots$; $k = 1, 2$. In such cases, complicated virtual obstacles similar to the maze-like obstacle shown in Fig. 5 can appear—albeit in I-space. The number of local cycles is always finite, and the convergence of the algoritm is preserved [31,37].

Now, the whole path planning procedure for Arm 1 can be formulated. The hit and leave points, H^j and L^j, are numbered in the order of their occurrence; $L^0 = S$. If a complementary M-line is introduced, the arm starts again at point S, and the numbering starts over. Distance $d(P, Q)$ between points P and Q is Euclidean distance in the plane of variables θ_1 and θ_2. A flag is used to indicate that, in the case of a Type II obstacle, one of the two closed curves of the current virtual boundary has been processed. The procedure consists of the following steps.

1. All four complementary M-lines are ordered as follows: M_1 is the shortest of the segments M_1, M_2, M_3, M_4; M_2 complements M_1 over the angle θ_2; M_3 complements M_1 over θ_1; M_4 complements M_1 over both θ_1 and θ_2. Go to Step 2.
2. M_1-line is designated as the M-line. The flag is set down. Set $j = 1$. Go to Step 3.
3. Counters C_1 and C_2 are set to zero. From point L^{j-1}, the arm moves along the M-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^j, is defined. Go to Step 4.
4. Counters C_1 and C_2 are turned on. The arm follows the virtual boundary until one of the following occurs:
   (a) The target is reached. The procedure stops.
   (b) M-line is met at a distance d from T such that $d < d(H^j, T)$; point L^j is defined. Increment j. Go to Step 3.
   (c) The arm returns to H^j (i.e., a closed curve along the virtual boundary has been completed) without ever meeting the M-line. Go to Step 5.
5. Examine the obstacle range accumulated in the counters C_1 and C_2. One of the following takes place:
   (a) The range is (0, 0) (i.e., this is a Type I obstacle). The target cannot be reached. The procedure stops.
   (b) The range is not (0, 0) and the flag is up (i.e., this is the second closed curve of the virtual boundary of a Type II obstacle). The target cannot be reached. The procedure stops.

The remaining three events relate to the case when the range is not (0, 0) and the flag is down (i.e., the first closed curve of the virtual boundary of a Type II obstacle is being processed).

(c) The range is $(0, n_2)$; $|n_2| \geq 1$ – an integer; designate the shorter of M_3 and M_4 as the M-line. Go to Step 6.
(d) The range is $(n_1, 0)$; $|n_1| \geq 1$; designate the shorter of M_2 or M_4 as the M-line. Go to Step 6.
(e) The range is (n_1, n_2); $|n_1|$, $|n_2| \geq 1$; designate the shortest of M_2, M_3, or M_4 as the M-line. Go to Step 6.

6. The arm moves back to Start. Set the flag up. Set $j = 1$. Go to Step 3.

3.3. *Arm 5: Sliding Link Followed by a Revolute Link*

A more detailed sketch of the arm of Fig. 7e is shown in Fig. 10. The first joint value is the variable length of the first link, l_1; $0 \leq l_1 \leq l_{1\max}$. The second joint value is the angle θ_2. The length of the second link, l_2, is constant. The boundary of the W-space of this arm is a combination of a rectangle whose sides are equal to $l_{1\max}$ and $2l_2$, respectively, and of two semicircles of radius l_2 attached to the rectangle as shown in Fig. 10. Assume that no obstacle can interfere with the arm outside of the W-space.

Two circles of radius l_2 centered at the limit positions O or O_1 of the link l_1 are called the *limit areas* of the W-space. Note that any point belonging to a limit area has only one corresponding arm solution, whereas any point outside the limit areas has two possible arm solutions (compare points P and P_1, Fig. 10). For the path planning purposes, this peculiarity makes the arm distinct from the other planar arms.

For a given actual obstacle, the corresponding virtual obstacle, virtual line, and virtual boundary are defined as above. An obstacle is considered to be *inside the limit area* if only one arm solution exists for any point of the obstacle virtual line. It is considered to be *outside the limit area* if two arm solutions exist for any point of the obstacle virtual line. An intermediate situation, when some points of the virtual line have one solution, and some other points have two solutions, is referred to as being *partially inside the limit area.*

The I-space of Arm 5 presents the surface of a cylinder whose height is equal to the maximum length of the first link, $l_{1\max}$. Vertical motion along the cylinder surface corresponds to changing the first joint value, l_1; horizontal motion, parallel to the cylinder *base circles*, corresponds to changing the second joint value, θ_2 (Fig. 11b). No matter whether or not the virtual line of an obstacle has self-intersections, the corresponding virtual boundary in I-space is formed by one or two simple curves. Depending on

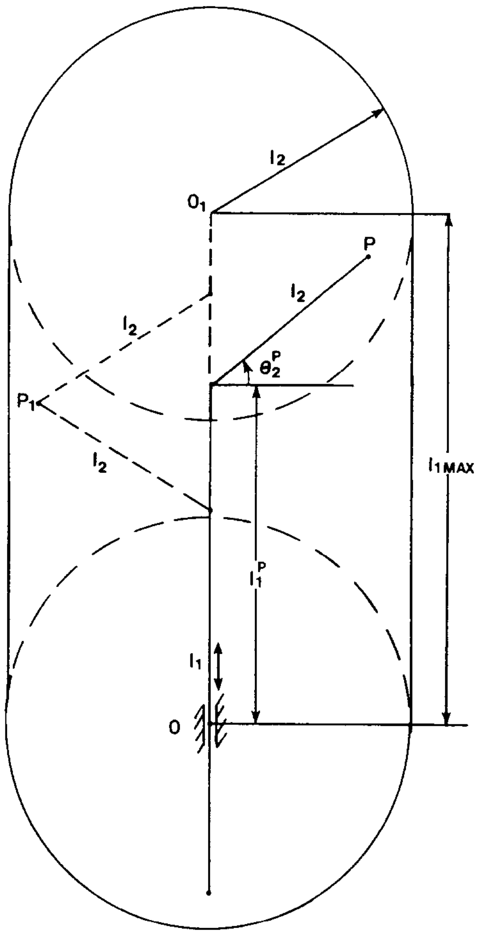

FIG. 10. W-space of Arm 5. Any point (such as P) within the circles whose centers are O and O_1, and whose radius is l_2, has one corresponding arm solution. Any point (such as P_1) outside these circles has two corresponding arm solutions, except on the border of W-space.

whether the obstacle in question is inside, partially inside, or outside the limit areas, its virtual boundary may or may not include segments of the base circles as its part (Fig. 11b). A segment of the virtual boundary which is a part of a base circle cannot be accessed by the arm. Therefore, some virtual boundaries (such as A in Fig. 11) form closed curves which can be traced by the arm fully, and some others (such as B and C, Fig. 11) form open simple curves whose endpoints correspond to one of the limit values of l_1.

Similar to Arm 1, define the M-line as a straight line in the plane of the joint variables l_1 and θ_2, that is, as a function $\theta_2 = p \cdot l_1 + q$, with the

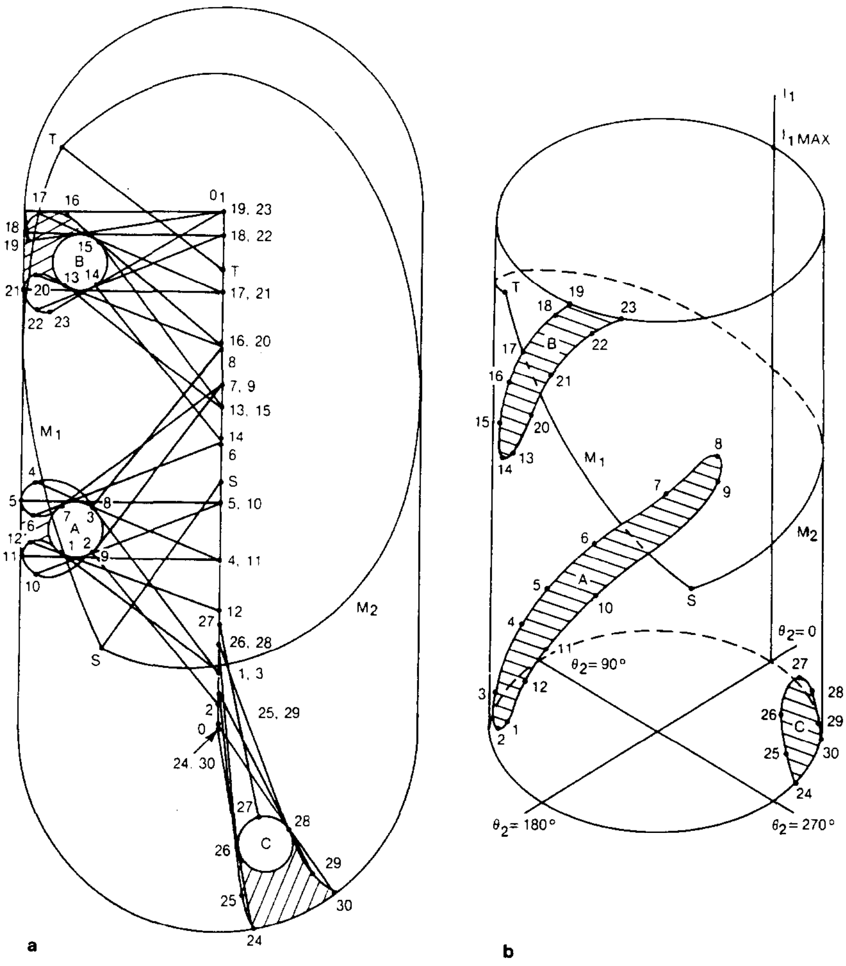

FIG. 11. Arm. 5. (a) W-space; shown are M-lines M_1 and M_2, and link positions during passing around obstacles A, B, and C. (b) I-space images of the same M-lines and of virtual obstacles.

coefficients p and q determined from the coordinates of the points S and T. The image of this M-line,[8] denoted as the M_1-line in Figs. 11a and 11b, is a geodesic line between the points S and T in the surface of the I-space clyinder. If, because of the obstacles, T cannot be reached from S using this M-line, a complementary M-line (denoted as the M_2-line) can be tried;

[8] From a practical standpoint, a straight line in W-space is less convenient because often a continuous motion of both links cannot be maintained. For example, in Fig. 11a, it is not possible to move the arm endpoint between the points S and T along a straight line; somewhere, a discontinuity takes place in the motion of one or both links.

the M_2-line is defined similarly to the M_1-line, with the second coordinate of the point T being $(\theta_2 - 2\pi)$ instead of θ_2.

A number of special cases of interaction between the arm and the obstacles can appear, which require separate handling in the algorithm. These cases are as follows [32,38]:

• The virtual obstacle forms a swath that connects both base circles of the I-space cylinder; in this case, the virtual boundary includes two simple open curves.

• The virtual obstacle forms a band around the I-space cylinder; the virtual boundary includes two simple closed curves, one of which is the circumference of one of the base circles.

• Two bands similar to the one above are formed, adjoining the opposite base circles of the I-space cylinder; depending on the obstacles, the bands may or may not be connected.

• The combination of the topology of the virtual obstacle(s) and their mutual position in respect to the M-line and the points S and T is such that local cycles appear (see Section 2).

To save space, the resulting path planning procedure for Arm 5 is not presented here. The algorithm is somewhat different from that for Arm 1 above; its complexity is about the same, and its performance, in terms of the length of paths in I-space, is given by the estimates presented in Section 2.

4. Three-Dimensional Arm Manipulators

The fact that maneuvering a body around another body in three-dimensional space presents an infinite number of alternatives, precludes direct application of the strategy of following simple closed curves in the image space. However, natural constraints imposed by the arm kinematics (e.g., the fact that the arm links are connected sequentially and that the arm base is fixed), may still allow one to reduce the problem to the cases above.

One special case of three-dimensional arm manipulators is a class of arms with two links and two joints, connected such that the arm body moves in three-dimensional space. One example from this class would be an arm with one link and a spherical joint at the arm base; this joint is equivalent to two revolute joints. The work space of this arm—that is, a set of points reachable by the arm endpoint—is the surface of a sphere of radius equal to the length of the link. On the other hand, the body of the arm moves in the three-dimensional space inside the sphere and thus can interact with three-dimensional obstacles. (Such arms have been dubbed $2\frac{1}{2}$-*dimensional arms*.) The I-space of this arm is similar to that of Arm 1 in

Section 3; there is also similarity in how one proves simplicity and closedness of the virtual boundaries required for assuring convergence of the resulting path planning procedure [39].

Another relatively simple type of three-dimensional robot arms is a Cartesian robot. This arm consists of three links sliding along three mutually perpendicular axes. Assume that the desired motion (the M-line) of the arm endpoint between the starting and target positions is along a straight line. The various ways that the links interact with the obstacles can be divided into three cases, with the actual motion presenting some combination of these cases. The resulting trajectory of the arm endpoint is a three-dimensional curve [34]. The three cases are as follows:

• The first link (counting from the arm base) is in contact with an obstacle. This degenerate case simply means that the target cannot be reached.

• The second link is in contact with an obstacle. In this case, only the first two links must participate in the path planning operation, thus reducing the problem to the two-dimensional case; the motion of the third link can, in principle, be arbitrary.

• The third link, or any combination of links including the third link, is in contact with obstacles. All three links must participate in the path planning. In this case, the motion is planned such that the arm endpoint moves in the plane containing the third link and the M-line, and perpendicular to the plane of the axes of the other two links. One additional assumption introduced here is that a fully retracted link cannot interfere with obstacles. With this assumption, which is plausible, from the practical standpoint, the advantage of the plane above is that it assures convergence of the planning procedure.

5. Experimental Validation of the Algorithms

The dynamic path planning algorithms have been extensively studied using a software package ROPAS (stands for RObot PAth Simulation) developed at the Yale University Robotics Laboratory for simulating sensor-based robot motion planning and collision avoidance. ROPAS is implemented in C language on the DEC MicroVAX II Workstation. It can simulate the behavior of an autonomous vehicle or a multi-link robot arm manipulator equipped with a tactile or proximity (vision) sensing capability; in the case of the arm, the whole "skin" of the arm body is sensitive. The user can interactively build and modify the environment, define the robot structure and its start and target positions, choose and run one of the motion planning algorithms, and document different aspects of the current run. Simultaneously with an animated real-time motion of the robot on the graphics screen, the user can observe the corresponding motion in the image space (see Section 3).

The package includes two computer games (see Introduction) whose purpose is to test human capabilities for path planning and collision avoidance in an uncertain environment. For example, immediately after an experiment with a human subject, the experimenter can check how the same problem would be solved by a robot equipped with the same sensing capabilities and having the same information about the environment. With one or two exceptions, the computer algorithm performed consistently better than human subjects. Interestingly, but not surprisingly, moving a robot arm was, for the subjects, a significantly more difficult task than moving a compact autonomous "bug"—although, from the standpoint of dynamic path planning, these tasks are roughly of comparable complexity. Human subjects are especially confused when multiple points of the robot body are simultaneously in contact with obstacles.

Our experiments with hardware are in their preliminary stage, both for the autonomous vehicle and for the robot arm manipulator. In the latter case, the body of the arm (General Electric robot arm, model P-5) is covered with panel modules each containing 8 to 16 infrared proximity sensors. Each sensor is an active device consisting of an infrared LED, which modulates a signal into the space to be sensed, and a photodetector, which detects the signal reflected from a near object. The resulting sensing capability is, therefore, different from those in humans; it amounts to a limited "vision" capability within a distance of 3–7 in., available at every point of the robot body. In other words, the sensor

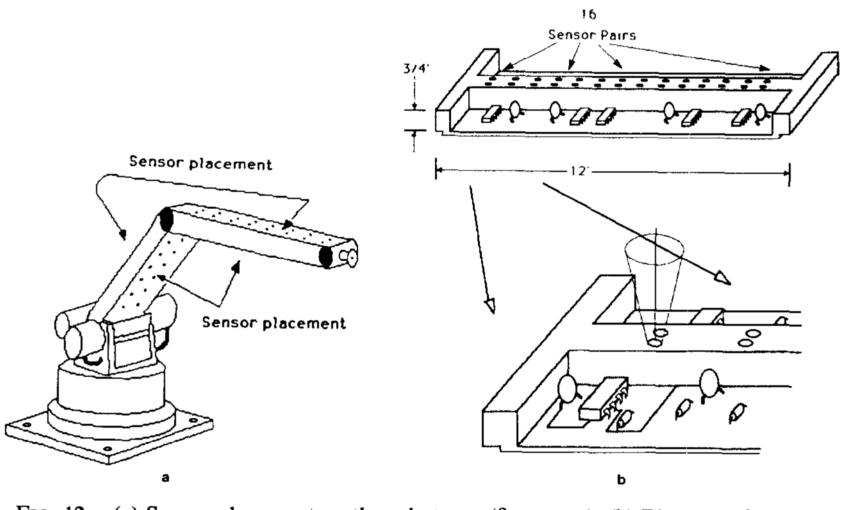

FIG. 12. (a) Sensor placement on the robot arm (first stage). (b) Blowups of one sensor module.

system forms a kind of aura similar to that provided by the hairs on the leg of an insect. A sketch of the robot arm with the sensors on it and of one sensor module is shown in Fig. 12.

Computationally, operating a multi-sensor system in real time has two distinct modes—one corresponding to "passive sensing" computations for the whole array of sensors during the arm motion far from obstacles, and the other corresponding to the processing of data for the sensors that are in the vicinity of obstacles. Although neither of these modes is computationally expensive, handling a large number of sensors presents a separate control problem.

Conclusion

Until recently, provable (nonheuristic) algorithms have been described only for the path planning model with complete information (Piano Movers model). The approach of dynamic path planning demonstrates that the model with incomplete information also presents a rich source of theoretically sound algorithms. Moreover, the ability of dynamic path planning algorithms to naturally incorporate sensory feedback and to operate in real time makes the approach an attractive candidate for robotics applications in unstructured and changing environments.

The development of the two basic models of motion planning—one with complete information and the other with incomplete information—has so far been proceeding along profoundly different avenues. Nearly every characteristic of the suggested methods (algorithms themselves, ways of measuring the algorithm performance, computational complexity issues, etc.) in both categories of works is remarkably distinct. A natural question then is: Why is this so?

There are two possible answers. First, the model with incomplete information accepts a continuous formulation, according to which an obstacle is a continuous curve or surface, and not a discretized set of nodes, faces, cells, etc., which are mandatory in the Piano Movers model. This changes the underlying methodology from the graph search to the exploitation of the topological characteristics of appropriate manifolds. It also results in a dramatic difference in the tools utilized and the methods of analysis of the algorithm performance.

Second, the two models exhibit an interesting intolerance to the addition or subtraction of extra information. The Piano Movers problem cannot be properly formulated in the presence of even "tiny" uncertainty. On the other hand, the algorithms of dynamic path planning as presented above do not seem to allow a natural mechanism for taking advantage of additional information—for example, using global partial information from a map stored in the robot's memory. Hence, as they stand at present, the two models are incompatible, although much would be gained if they could be combined. This raises an array of questions, which are likely to be addressed in future research.

REFERENCES

1. SCHWARTZ, J. T., AND SHARIR, M. (1983), On the "Piano Movers" problem. I. The case of a two-dimensional rigid polygonal body moving amidst polygonal barriers, *Comm. Pure Appl. Math.* **36,** 345–398.
2. LOZANO-PEREZ, T., AND WESLEY, M. (1979), An algorithm for planning collision-free paths among polyhedral obstacles, *Comm. ACM* **22,** 560–570.
3. MORAVEC, H. (1983), The Stanford cart and the CMU rover, *Proc. IEEE* **71,** No. 7, 872–874.
4. BROOKS, R. A. (1983), Solving the find-path problem by good representation of free space, *IEEE Trans. Systems Man Cybernet.* **13,** No. 3.
5. BINFORD, T. O. (1971), Visual perception by computer, *in* "Proceedings, IEEE Conf. on Systems, Science, and Cybernetics, Miami, Florida, December."
6. REIF, J. (1979), Complexity of the Mover's Problem and generalizations, *in* "Proceedings, 20th Symposium of the Foundations of Computer Science (FOCS).
7. PIEPER, D. L. (1968), "The Kinematics of Manipulators Under Computer Control," Ph.D. thesis, Stanford University, October.
8. PAUL, R. (1972), "Modeling Trajectory Calculation and Servoing of a Computer controlled Arm," Ph.D. thesis, Stanford University, Nov.
9. SCHWARTZ, J. T., AND SHARIR, M. (1983), On the "Piano Movers" problem. II. General techniques for computing topological properties of real algebraic manifolds, *Adv. in Appl. Math.*, No. 4, 298–351.
10. HOPCROFT, J., JOSEPH, D., AND WHITESIDES, S. (1982), On the movement of robot arms in 2-dimensional bounded regions, *in* "Proceedings, IEEE Foundations of Computer Science Conf., Chicago, November."
11. BULLOCK, B., KEIRSEY, D., MITCHELL, J., NUSSMEIER, T., AND TSENG, D. (1983), Autonomous vehicle control: An overview of the Hughes project, *in* "Proceedings, IEEE Computer Society Conf. Trends and Applications, 1983: Automating Intelligent Behavior, Gaithersburg, Maryland, May."
12. THOMPSON, A. M. (1977), The navigation system of the JPL robot, *in* "Proceedings, 5th Joint International Conf. on Artificial Intelligence, Cambridge, Massachusetts, August."
13. KERSEY, D. M., KOCH, E., MCKISSON, J., MEYSTEL, A. M., AND MITCHELL, J. S. B. (1984), Algorithm of navigation for a mobile robot, *in* "Proceedings, IEEE International Conf. on Robotics, Atlanta, Georgia, March."
14. BLUM, M., AND KOZEN, D. (1978), On the power of the compass (or, Why mazes are easier to search than graphs), *in* "Proceedings, 19th Annual Symposium on Foundation of Computer Science (FOCS), Ann Arbor, Michigan."
15. LIPSKI, W., AND PREPARATA, F. (1981), Segments, rectangles, contours, *J. Algorithms* **2,** 63–76.
16. ABELSON, H., AND DISESSA, A. (1980), "Turtle Geometry," pp. 179–199, MIT Press.
17. YAP, C. (1986). Algorithmic Motion Planning (Survey), *In* "Advances in robotics, Vol. 1: Algorithmic and Geometric Aspects" (J. Schwartz and C. Yap, Eds.), Lawrence Erlbaum Assoc., Hillsdale, N.J.
18. MEIJDAM, L., AND DE ZEEUW, A. (1986), On expectations, information, and dynamic game equilibria, *in* "Dynamic Games and Applications in Economics" (T. Basar, Ed.), Springer-Verlag.
19. MOORE, E. (1964), The firing squad synchronization problem, *in* "Sequential Machines" (E. Moore, Ed.), Reading, MA.

20. TRAUB, J., WASILKOWSKI, G., AND WOZNIAKOWSKI, H. (1983), "Information, Uncertainty, Complexity," Addison–Wesley.
21. REIF, J. (1986), A survey on advances in the theory of computational robotics, *in* "Adaptive and Learning Systems" (K. Narendra, Ed.), Plenum, New York.
22. MASSEY, W. S. (1967), "Algebraic Topology," Harcourt, Brace, & World, New York.
23. CROWLEY, G. L. (1985), Navigation for an intelligent mobile robot, *IEEE J. Robotics Automation* **RA-1,** No. 1, March.
24. PETROV, A. A., AND SIROTA, I. M. (1981), Control of a robot manipulator with obstacle avoidance under little information about the environment, *in* "Proceedings, VIII Congress of IFAC, Kyoto, Japan, v.XIV."
25. CHATILA, R. (1982), Path planning and environment learning in a mobile robot system, *in* "Proceedings, European Conf. on Artificial Intelligence, Torsey, France."
26. CHATTERGY, R. (1985), Some heuristics for the navigation of a robot, *Internat. J. Robotics Res.* **4,** No. 1, 59–66.
27. MILENKOVIC, V., AND HUANG, B. (1983), Kinematics of major robot linkage, *in* "Proceedings, 13th International Symposium on Industrial Robots and Robots 7 Conf., Chicago, April."
28. LUMELSKY, V., AND STEPANOV, A. (1984), Effect of uncertainty on continuous path planning for an autonomous vehicle, *in* "Proceedings, 23rd IEEE Conf. on Decision and Control, Las Vegas, Nevada, December."
29. LUMELSKY, V. (1985), On path planning for a planar robot arm with uncertainty, *in* "SIAM Conf. on Geometric Modeling and Robotics, Albany, New York, July."
30. LUMELSKY, V. (1985), On non-heuristic motion planning in unknown environment, *in* "Proceedings, IFAC Symposium on Robot Control, Barcelona, Spain, November."
31. LUMELSKY, V. (1986), Continuous robot motion planning in unknown environment, *in* "Adaptive and Learning Systems: Theory and Applications" (K. Narendra, Ed.), Plenum, New York.
32. LUMELSKY, V. (1985), Effect of robot kinematics on motion planning in unknown environment, *in* "Proceedings, 24th IEEE Conf. on Decision and Control, Fort Lauderdale, Florida, December."
33. LUMELSKY, V., AND STEPANOV, A. (1986), Dynamic path planning for a mobile automaton with limited information on the environment, *IEEE Trans. Automat. Control* **AC-31,** No. 11, November.
34. LUMELSKY, V. (1986), Continuous motion planning in unknown environment for a 3D cartesian robot arm, *in* "Proceedings, IEEE International Conf. on Robotics and Automation, San Francisco, April."
35. SUN, K., AND LUMELSKY, V. (1986), Simulating sensor-based robot motion amongst unknown obstacles, *in* "Proceedings, NATO Workshop on Languages for Sensor-Based Control in Robotics, Pisa, Italy, September."
36. SUN, K., AND LUMELSKY, V. (1987), Computer simulation of sensor-based robot collision avoidance in an unknown environment, *Robotica,* Oxford Univ. Press, to appear.
37. LUMELSKY, V. (1987), Dynamic path planning for a planar articulated robot arm moving amidst unknown obstacles, *Automatica, J. IFAC (International Federation of Automatic Control),* Sept.
38. LUMELSKY, V. (1987), Effect of kinematics on dynamic path planning for planar robot arms moving amidst unknown obstacles, *IEEE J. Robotics Automation* **RA-3,** No. 3, June.
39. LUMELSKY, V., AND SUN, K. (1987), Gross motion planning for a simple 3D articulated robot arm moving amidst unknown arbitrarily shaped obstacles, *in* "Proceedings, IEEE International Conf. on Robotics and Automation, Raleigh, North Carolina, April."
40. LUMELSKY, V., AND STEPANOV, A. (1987), Path planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape, *Algorithmica,* Springer-Verlag, to appear.