Incorporating Range Sensing in the Robot Navigation Function

VLADIMIR LUMELSKY, SENIOR MEMBER, IEEE, AND TIM SKEWIS, STUDENT MEMBER, IEEE

Abstract—**A model of mobile robot navigation is considered whereby the robot is a point automaton operating in an environment with unknown obstacles of arbitrary shapes. The robot's input information includes its own and the target point coordinates, as well as local sensing information such as from stereo vision or a range finder. These algorithmic issues are addressed: 1) Is it possible to combine sensing and planning functions, thus producing, similar to the way it is done in nature, "active sensing" guided by the needs of planning? (The answer is "yes"). 2) Can richer sensing (e.g., stereo vision versus tactile) guarantee better performance, that is, resulting in shorter paths? (The general answer is "no.") A paradigm for combining range data with motion planning is presented. It turns out that extensive modifications of simpler "tactile" algorithms are needed to take full advantage of additional sensing capabilities. Two algorithms that guarantee convergence and exhibit different "styles" of behavior are described, and their performance is demonstrated in simulated examples.**

I. Introduction

EFFICIENT UTILIZATION of available information for the navigation purposes presents an important problem in robotics. It is assumed in the sequel that the robot is a point automaton operating in a two-dimensional (2-D) surface with unknown obstacles of arbitrary shape; its task is to produce a collision-free path between known start and target locations. The robot is equipped with a sensor that provides information on obstacles with some "radius of vision." Although the instantaneous navigation decisions in such a system are necessarily of local character, they must guarantee acceptable global performance. Our emphasis here is on global characteristics of underlying algorithms such as global convergence and the relationship between sensing media and the motion planning function.

More specifically, we pose the following questions. Compared, for example, with tactile sensing, does vision or range data really help in motion planning, and if so, in what way and under what conditions? Can this question be put in a formal way? What is the relationship, if any, between the type and basic characteristics of the sensors used and the convergence of the motion planning algorithms? Can sensing be algorithmically incorporated into the planning function, so as to result in active sensing guided by the needs of motion planning—the way it is done in nature, which is quite different from the way it is currently done in robotics?

Interestingly, although these questions are far from simple and are quite important for both theory and engineering practice, there have been no attempts, to our knowledge, of addressing them. It is partly for this reason, we believe, that today's experimental mobile robots, even when equipped with powerful computers and all kinds of sophisticated sensors, look so helpless when dealing with nontrivial obstacles—e.g., in a large room with partitions, where the robot may have to backtrack and visit some segments of its path more than once.

Current research on robot motion (path) planning revolves around two models that are based on different assumptions about the information available for planning. In the first model, called "path planning with complete information" (or the "piano movers problem"), perfect information about the robot and the obstacles is assumed. The obstacle boundary must be algebraic (in many works, polygonal). Under this model, the whole process of motion planning is a one-time off-line operation (see, e.g., [1]).

This paper is concerned with the second model, called "path planning with incomplete information," or "path planning with uncertainty." In this model, an element of uncertainty is present, and the missing data are typically provided by some source of local information, such as from a laser range finder or a vision sensor. The attractiveness of this model for robotics lies in the possibility to naturally introduce a notion of sensor feedback, and thus transform the operation of motion planning into a continuous dynamic on-line process. Also, the requirement of analytic representation of obstacle boundaries becomes unnecessary and can be dropped. Because little information is being processed at each step, the main difficulty is not in computational efficiency, as is the case in the piano mover problem, but rather in guaranteeing convergence.

One important issue in motion planning with incomplete information is the way the sensory data are incorporated into the planning function. A more traditional approach in this area is to separate the functions of scene

Manuscript received March 23, 1989; revised April 13, 1990. This work was supported in part by the National Science Foundation grants DMC-8519542 and DMC-8712357; in part by Unimation Inc.; and in part by Transitions Research Corporation. This paper was partially presented at the IEEE International Conference on Robotics and Automation, Philadelphia, PA, April 1988.

The authors are with the Department of Electrical Engineering, Yale University, New Haven, CT 06520.

IEEE Log Number 9036585.

0018-9472/90/0900-1058$01.00 ©1990 IEEE

reconstruction and motion planning. In such a system, the environment, or a part of it, is first reconstructed based on the sensor data, independent of how the produced information is going to be used later. Then an algorithm that operates under an assumption of complete information is used for motion planning. This approach gives rise to various techniques of scene reconstruction that are either weakly tied to the task of path planning [2]–[5], or are completely independent of it [6], [7].

Another approach to sensor-based motion planning, also considered in this paper, calls for an intimate integration of the sensory capability and the planning function. Under this approach, sensing becomes an active process: the robot is deciding at each point of its path what sensory information is required for generating its next step. A natural question that appears in this context is whether richer on-line sensory data, such as from a range finder or vision, can guarantee better path length performance, that is, result in shorter paths.

Our objective is to design algorithms capable of guiding a mobile robot in a nontrivial scene with arbitrary obstacles, given the simplest distance information that one can expect from a range finder or stereo vision. We introduce a simplified model of a "vision sensor," which mimics a typical range finder in that it provides the robot with coordinates of those points of obstacle boundaries that lie within a limited radius of vision around the robot. A paradigm is then considered for incorporating range data into the robot motion-planning function. By making use of theory developed in [8], motion-planning algorithms utilizing active range sensing are built next. It turns out that extensive modifications of "tactile" algorithms are needed to fully utilize additional sensing capabilities. Two principles for designing strategies with proven convergence are considered; the resulting algorithms exhibit different "styles" of behavior and are not, in general, superior to each other.

The first procedure is more conservative in that it simply uses range data to "cut corners" that would have been produced by a "tactile" algorithm. The difference between the two can be exemplified by comparing the behavior of two people—one sightful and the other blindfolded—who attempt to walk around the perimeter of a building of complex shape. The second procedure is more opportunistic in that it simply attempts to choose its intermediate goals closer to the target.

Since our emphasis is on global algorithmic issues of active sensing, numerous questions of local control that would arise in actual implementation—such as handling robot dynamics—are ignored here. In principle, any techniques of local control can be used on conjunction with our algorithms (see, e.g., [10]). Some results of an attempt to implement these algorithms can be found in [11].

The accepted model and required definitions are presented in section II. The basic idea of the algorithms is discussed in section III, followed, in sections IV and V, by relevant analysis, description of two algorithms, and examples demonstrating the algorithm performance.

II. Model

The environment (the scene) is a plane with a set of static obstacles and two points, start (S) and target (T), in it. Each obstacle is a simple closed curve of arbitrary shape and 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. Obstacles do not touch each other. The environment 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.

The robot is a point. Its input information includes coordinates of its current location, C, as well as the target T. The robot is capable of moving along a straight line and along obstacle boundaries. It also has a capability, referred to as vision, that allows it to detect an obstacle and the distance to it along any direction from point C within its field of vision. The field of vision presents a disc of radius r_v (radius of vision) centered at C. A point Q is visible if, first, it is located within the field of vision, and second, the straight-line segment CQ does not cross any obstacles.

The robot is capable of using its vision in a scanning operation, during which it identifies obstacles, or the lack thereof, that intersect the whole field of vision or appear in a specific direction. As will become clear in the following, the robot usually has no need to scan the entire 360° circumference every time; instead it scans only specific directions requested by the path planning algorithms. Thus the robot can, for example, identify some intermediate target point that lies within its field of vision and walk toward that point along a straight line. On the other hand, the robot can use its capability for walking along the obstacle boundary to maneuver around a convex obstacle when the visibility of the obstacle boundary shrinks to zero.

The robot has limited memory that allows it to store a few "interesting" points, but is not sufficient, for example, for storing incremental maps. (Recall that those might be very large, given obstacles of arbitrary shape.) Thus the robot is not able to recognize an obstacle that it passed before.

A desirable path to T, called the main line or M-line, is introduced as a straight-line segment that connects S and T. An elementary operation of defining the next intermediate target point, T_i, is executed by the robot at every moment i, given its current position C_i and the range data within the current field of vision. Then the robot makes a little step in the direction of T_i, and the process repeats. In the algorithms described in the following, every T_i lies either on the M-line or on an obstacle boundary. For a segment of the path where T_i moves along the M-line, the first defined T_i that lies at the intersection between the M-line and an obstacle is a special point called the hit point, H. For a segment of the

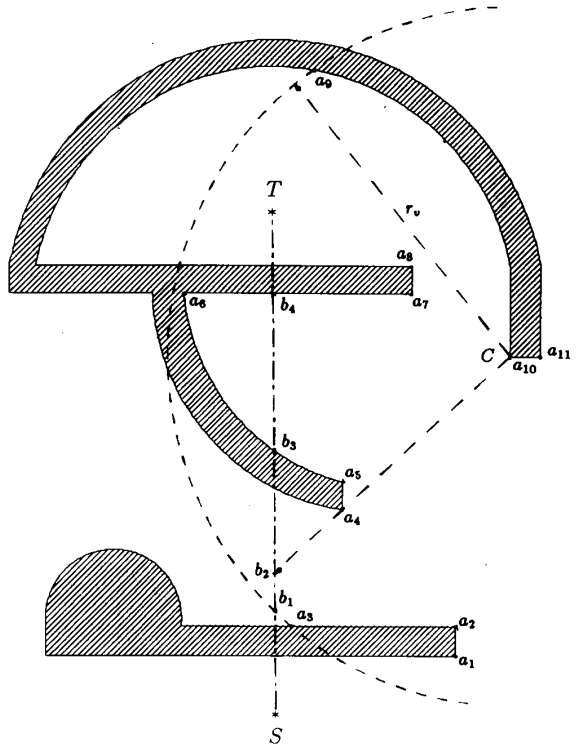

Fig. 1. Shaded areas represent obstacles. At its current location C, robot will see obstacles within its radius of vision r_v such as segments of obstacle boundaries $a_1a_2a_3, a_4a_5a_6a_7a_8, a_9a_{10}a_{11}$. It will also conclude that segments b_1b_2 and b_3b_4 of M-line are "visible."

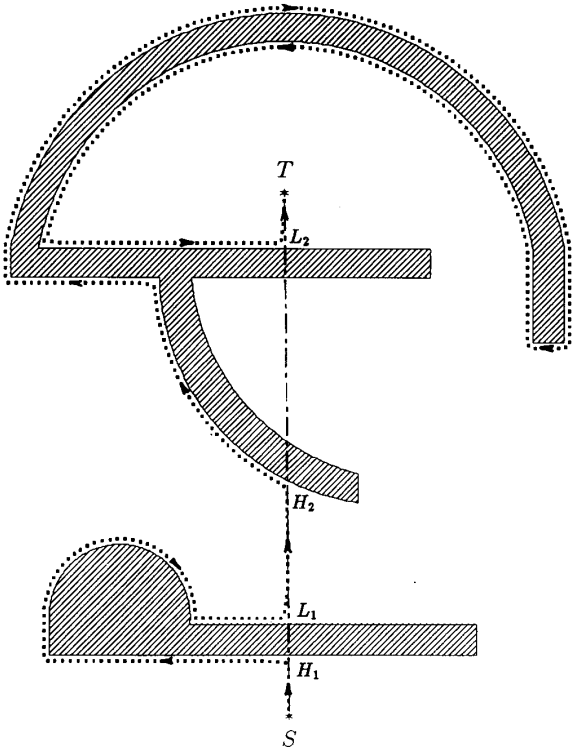

Fig. 2. Environment 1: path generated by algorithm Bug2.

path where T_i moves along an obstacle boundary, the first defined T_i that lies at the M-line is a special point called the leave point, L. The main difference between the two algorithms considered here is in how they define T_i. Naturally the current T_i is always at a distance from the robot not more than r_v.

While scanning the field of vision, the robot may be detecting some contiguous sets of visible points—for example, a segment of the obstacle boundary. A point Q is contiguous to another point S over the set $\{P\}$, if 1) $S \in \{P\}$, 2) Q and $\{P\}$ are visible, and 3) Q can be continuously connected with S using only points of $\{P\}$. A set is contiguous if any pair of its points are contiguous to each other over the set. As one will see, no memorization of contiguous sets will be needed—while "watching" a contiguous set, the robot's only concern will be whether two points are contiguous to each other.

A local direction is a once-and-for-all determined direction for passing around an obstacle; facing the obstacle, it can be either left or right (or clockwise and counterclockwise). For the sake of clarity, assume that the local direction is always left.[1]

[1]Although locally one direction may seem preferable to the other, it is known that from the global performance standpoint, as long as no complete information is available, neither local direction can be judged better than the other [8].

The M-line divides the environment into two half-planes. The half-plane that lies to the local direction side of the M-line is called the main semiplane, and the other half-plane is called the secondary semiplane. Thus if the local direction is "left," then the left half-plane, looking from S toward T, is the main semiplane.

The defined terms are exemplified in Fig. 1. The shaded areas represent obstacles. The straight-line segment ST is the M-line; the current location of the robot, C, is in the secondary (right) semiplane; its field of vision is of radius r_v. If, while standing at C, the robot performs the complete scanning operation, it will identify contiguous segments of the obstacle boundaries, $a_1a_2a_3$, $a_4a_5a_6a_7a_8$, and $a_9a_{10}a_{11}$, and contiguous segments of the M-line, b_1b_2 and b_3b_4.

III. Basic Schemes

In principle, planning algorithms with range sensing can be based on different maze-searching procedures. Our algorithms (in the following) make use of a procedure called Bug2 [8], which we briefly sketch first. The algorithm is based on the preceding model, except its sensing capability is limited to tactile sensing; this can be interpreted as zero vision, $r_v = 0$. Under the algorithm Bug2, the robot can meet the same obstacle more than once, and it has no way of distinguishing between different obstacles. The subscript j will 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 algorithm Bug2 consists of the following steps (follow Fig. 2).

1) From point L_{j-1}, move along the straight-line segment ST until one of the following occurs:
   a) T 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) T is reached. The procedure stops.
   b) The line segment ST is met at a point Q such that the distance from Q to T is less than that from H_j to T, and the line segment QT 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 robot 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 cannot be reached. The procedure stops.

Because we want to preserve convergence of the path planning procedures, introducing vision in an algorithm, such as Bug2, cannot be done in a direct way. For example, it can be shown that simply walking to the farthest visible "corner" of an obstacle that lies in the "right" direction can ruin convergence.

Since the procedure Bug2 is known to converge, one way of using vision is to organize the algorithm such that at each step of its path the robot "mentally" reconstructs in its current field of vision the segment of the path that would be produced by Bug2 (it is called below the Bug2 path), makes the farthest point of that segment its intermediate target, and makes a step directly toward that target. Notice that the only essential thing during such an operation is the continuity of the considered segment of the Bug2 path—the segment itself does not have to be remembered. As it turns out, deciding whether a given point lies on the path that would be generated by Bug2 (or, for short, whether a given point is a Bug2 point) is not a trivial task. The resulting algorithm is called below VisBug-21 (where the first numeral, "2," refers to Bug2), and the path it generates is referred to as the VisBug-21 path.

A different procedure, also based on the Bug2 mechanism but with a more opportunistic behavior pattern than in VisBug-21, can be designed. Instead of using the Bug2 path for generating its own path, the robot can deviate from what is dictated by the Bug2 path to take advantage of opportunities that look more promising (as long as convergence is preserved). This procedure is called VisBug-22. As one will see in the following sections, given the same initial conditions, both procedures, VisBug-21 and VisBug-22, can produce quite different paths. Furthermore, depending on the environment, one procedure may be more efficient (that is, produce a shorter path) than the other, and so both present viable options. Both algorithms include a test for target reachability that is

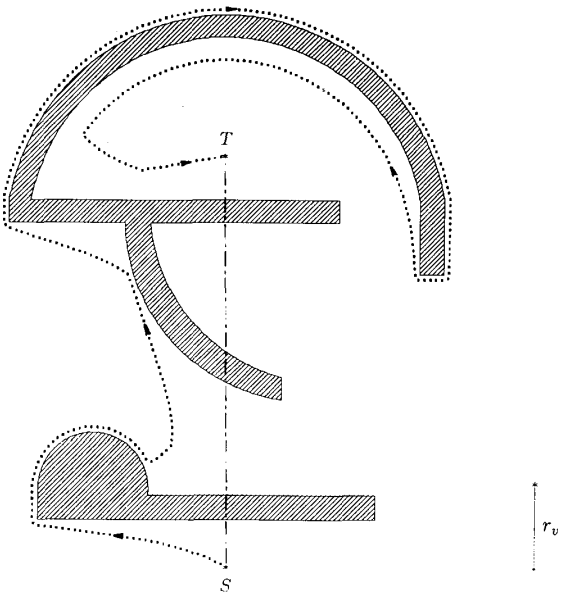

Fig. 3. Environment 1: path generated by algorithms VisBug-21 or VisBug-22; radius of vision is r_v.

based on the following necessary and sufficient condition: if, after having defined the last hit point as its intermediate target, the robot returns to it before it defines the next hit point, then either the robot or the target point is trapped, and hence the target is not reachable (for more details, see [8]).

The following notation will be used.

- C_i and T_i are the position and the intermediate target of the robot at the ith step.
- $|AB|$ is the straight-line segment whose endpoints are A and B; it may also designate the length of this segment.
- (AB) is the obstacle boundary segment whose endpoints are A and B, or the length of this segment.
- $[AB]$ is the path segment between the points A and B that would be generated by algorithm Bug2, or the length of this path segment.
- $\{AB\}$ is the path segment between the points A and B that would be generated by VisBug-21 or VisBug-22, or the length of this path segment.

It will be evident from the context whether a given notation refers to a segment or its length. Also, when more than one segment appears between points A and B, the context will resolve the ambiguity.

IV. Algorithm VisBug-21

A. Algorithm

The algorithm consists of the main body, which does the proper motion planning along the path, and a procedure called Compute T_i-21, which generates the next intermediate target T_i and performs the test for target

reachability. As will be seen in the following, in most cases the operation of the main body is confined to step S1. Step S2 is executed only in those cases when the robot is moving along a (locally) convex boundary of an obstacle, and so it cannot use its vision for defining the next intermediate target T_i. For reasons that will become clear later, the algorithm distinguishes the case when point T_i lies in the main semiplane from the case when T_i lies in the secondary semiplane. Initially, $C = T_i = S$.

1) Main Body: This procedure is executed at each point of the continuous path. It consists of the following steps:

- Step 1: Move towards T_i while executing Compute T_i-21 and performing the following test:
  If $C = T$ the procedure stops.
  Else if the target is unreachable the procedure stops.
  Else if $C = T_i$ go to step S2.
- Step 2: Move along the obstacle boundary while executing Compute T_i-21 and performing the following test:
  If $C = T$ the procedure stops.
  Else if the target is unreachable the procedure stops.
  Else if $C \neq T_i$ go to step S1.

2) Procedure Compute T_i-21: The procedure consists of the following steps.

- Step 1: If T is visible then define $T_i = T$; procedure stops.
  Else if T_i is on an obstacle boundary go to Step 3.
  Else go to Step 2.
- Step 2: Define point Q as the endpoint of the maximum length contiguous segment of the M-line, $|T_iQ|$, extending from T_i in the direction of T.
  If an obstacle has been identified crossing the M-line at point Q then define a hit point, $H = Q$; assign $X = Q$, define $T_i = Q$; go to Step 3.
  Else define $T_i = Q$; go to Step 4.
- Step 3: Define point Q as the endpoint of the maximum length contiguous segment of the obstacle boundary, (T_iQ), extending from T_i in the local direction.
  If the obstacle has been identified crossing the M-line at a point $P \in (T_iQ)$, $|PT| < |HT|$, then assign $X = P$ and if, in addition, $|PT|$ does not cross the obstacle at P then define a leave point, $L = P$; define $T_i = P$; and go to Step 2.
  If the lastly defined hit point, H, is again identified and $H \in (T_iQ)$ then the target is not reachable; procedure stops.
  Else define $T_i = Q$; go to Step 4.
- Step 4: If T_i is on the M-line define $Q = T_i$, otherwise define $Q = X$.
  If points $\{P\}$ on the M-line are identified such that $|S'T| < |QT|$, $S' \in \{P\}$, and C is in the main semiplane then find the point $S' \in \{P\}$ that produces the shortest distance $|S'T|$; define $T_i = S'$; go to Step 2.
  Else procedure stops.

In simpler terms, the procedure Compute T_i-21 operates as follows. Step 1 is executed at the (last) stage when target T becomes visible (e.g., at point A, Fig. 4). A special case, in which points of the M-line noncontiguous to the previously considered sets of points are tested as candidates for the next intermediate target T_i, is handled in Step 4. All the remaining situations relate to choosing the next T_i among the points of the Bug2 path contiguous to the previously defined T_i; these are treated in Steps 2 and 3. Specifically, in Step 2 candidate points along the M-line are processed, and hit points are defined. In Step 3, candidate points along obstacle boundaries are processed, and leave points are defined. The test for target reachability is also performed in Step 3. It is conceivable that, given a current location C_i of the robot, the procedure will execute, perhaps even more than once, some combination of Steps 2, 3, and 4. While doing that, contiguous and noncontiguous segments of the Bug2 path along the M-line and along obstacle boundaries are considered before the next intermediate target T_i is defined and the robot makes a physical step towards T_i.

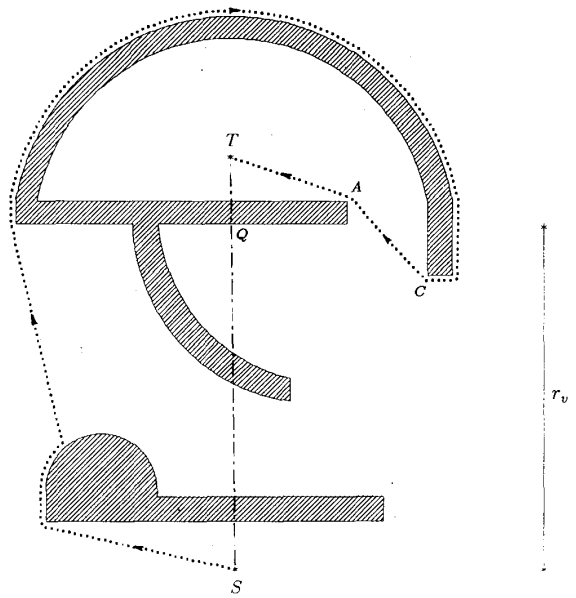

Fig. 4. Environment 1: path generated by VisBug-21; radius of vision r_v is larger than that in Fig. 3.

B. Analysis

Examples shown in Figs. 3 and 4 demonstrate the effect of radius of vision r_v on the performance of algorithm VisBug-21 (compare with Bug2 algorithm in the same environment, Fig. 2). In the following analysis, we first look at the global performance of the algorithm, and then address the issue of convergence. Since the path generated by VisBug-21 can diverge significantly from the path that would be produced under the same conditions by algorithm Bug2, it is to be shown that the path length performance of VisBug-21 is never worse than that of Bug2. One would expect such a behavior, and it is indeed assured by the following lemma.

Lemma 1: For a given environment and a given set of start and target points, the path produced by algorithm VisBug-21 is never longer than that produced by algorithm Bug2.

Proof: Assume that the environment and the start and target points, S and T, are fixed. Consider the position of the robot, C_i, and its corresponding intermediate target, T_i, at the step i of the path, $i = 0, 1, \cdots$. We wish to show that the lemma holds not only for the whole path from S to T, but also for an arbitrary step i of the path. This amounts to showing that the inequality,

$$\{SC_i\} + |C_iT_i| \leqslant [ST_i] \tag{1}$$

holds for any i. The proof is by induction. Consider first the initial stage when $i = 0$. This corresponds to $C_0 = S$. Clearly, $|ST_0| \leqslant [ST_0]$. This can be written as $\{SC_0\} + |C_0T_0| \leqslant [ST_0]$, which corresponds to (1) when $i = 0$. To proceed by induction, assume that (1) holds for the step $(i-1)$ of the path, $i > 1$:

$$\{SC_{i-1}\} + |C_{i-1}T_{i-1}| \leqslant [ST_{i-1}]. \tag{2}$$

Each step of the robot's motion takes place in one of two ways: either $C_{i-1} \neq T_{i-1}$, or $C_{i-1} = T_{i-1}$. The latter case takes place when the robot moves along the boundary of a (locally) convex obstacle, whereas the former case comprises all the remaining situations. Consider the first case, $C_{i-1} \neq T_{i-1}$. Here the robot will take a step of length $|C_{i-1}C_i|$ along a straight line towards T_{i-1}; thus (2) can be rewritten as

$$\{SC_{i-1}\} + |C_{i-1}C_i| + |C_iT_{i-1}| \leqslant [ST_{i-1}]. \tag{3}$$

In (3), the first two terms form $\{SC_i\}$, and so

$$\{SC_i\} + |C_iT_{i-1}| \leqslant [ST_{i-1}]. \tag{4}$$

At point C_i, the robot will define the next intermediate target, T_i. Now, add the obvious inequality, $|T_{i-1}T_i| \leqslant [T_{i-1}T_i]$, to (4):

$$\{SC_i\} + |C_iT_{i-1}| + |T_{i-1}T_i| \leqslant [ST_{i-1}] + [T_{i-1}T_i] = [ST_i]. \tag{5}$$

By the triangle inequality,

$$|C_iT_i| \leqslant |C_iT_{i-1}| + |T_{i-1}T_i| \tag{6}$$

Therefore it follows from (5) and (6) that

$$\{SC_i\} + |C_iT_i| \leqslant [ST_i] \tag{7}$$

which proves (1).

Now, consider the second case, $C_{i-1} = T_{i-1}$. Here the robot takes a step of length $(C_{i-1}C_i)$ along the obstacle boundary (the Bug2 path, $[C_{i-1}C_i]$). Then (2) becomes

$$\{SC_{i-1}\} + [C_{i-1}C_i] \leqslant [SC_{i-1}] + [C_{i-1}C_i] \tag{8}$$

where the left side amounts to $\{SC_i\}$ and the right side to $[SC_i]$. At point C_i, the robot will define the next intermediate target, T_i. Since $|C_iT_i| \leqslant [C_iT_i]$, (8) can be written as

$$\{SC_i\} + |C_iT_i| \leqslant [SC_i] + [C_iT_i] = [ST_i] \tag{9}$$

which, again, produces (1). Since, by the algorithm's design, at some finite i, $C_i = T$, then

$$\{ST\} \leqslant [ST] \tag{10}$$

which completes the proof.

One can also see from Fig. 4 that when r_t goes to infinity, algorithm VisBug-21 will generate locally optimal paths, in the following sense. Take two obstacles or parts of the same obstacle, k and $k+1$, that are visited by the robot, in this order. During the robot's interaction with (i.e., passing around) the obstacle k, once the obstacle $k+1$ is identified as the next intermediate target, the area between k and $k+1$ will be traversed along the straight line—which presents the locally shortest path.

To define its next intermediate target, T_i, the algorithm VisBug-21, as presented previously, sometimes uses points on the M-line that are not necessarily contiguous to the previous intermediate targets. Such operation results in a more efficient use of the robot's vision; by "cutting corners," the robot can often skip some obstacles that interest the M-line and that it would otherwise have to pass. But, from the convergence standpoint, it is important to assure that in such cases the considered candidate points on the M-line do indeed lie on the Bug2 path. Note that in Step 4 of the procedure Compute T_i-21 a noncontiguous point Q on the M-line is considered a possible candidate for an intermediate target only if the current location C of the robot is in the main semiplane. We want to show that this arrangement always produces intermediate targets that lie on the Bug2 path.

Consider a current location C of the robot along the VisBug-21 path, a current intermediate target T_i, and some visible point Q on the M-line that is being considered as a candidate for the next intermediate target, T_{i+1}. Apparently, Q can be accepted as the intermediate target T_{i+1} only if it lies further along the Bug2 path than T_i.

To assure convergence, algorithm Bug2 organizes the set of hit and leave points, H_j and L_j, along the M-line so as to form a sequence of segments

$$|ST| > |H_1T| > |L_1T| > |H_2T| > |L_2T| > \cdots \tag{11}$$

that shrinks to T [8]. This inequality dictates two conditions that the candidate points Q must satisfy in order for algorithm VisBug-21 to converge: i) when the current intermediate target T_i lies on the M-line, then only those points Q should be considered for which $|QT| < |T_iT|$; ii) when T_i is not on the M-line, it lies on the obstacle boundary, in which case there must be the latest crossing point X between the M-line and the boundary, such that the boundary segment (XT_i) is a part of the Bug2 path; in this case, only those points Q should be considered for which $|QT| < |XT|$. Since points Q, T_i, and X are already known, both of these conditions can be easily checked. We assume that these conditions are satisfied. Note that the crossing point X does not necessarily correspond to a hit point for both the Bug2 and the VisBug-21 algorithms. The following statement holds.

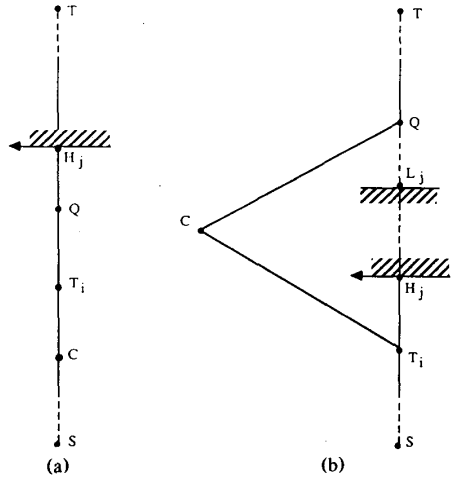

Fig. 5. Illustration for Lemma 2. (a) Case 1. (b) Case 2.

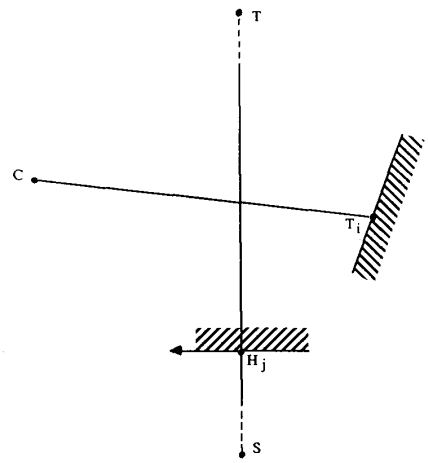

Fig. 6. Illustration for Lemma 2: Observation 2.

Lemma 2: For the point Q to be further along the Bug2 path than the intermediate target T_i, it is sufficient that the current robot position C lies in the main semiplane.

Proof: Assume that C lies in the main semiplane; this includes a special case when C lies on the M-line. Then all possible situations can be classified into three cases:

1) Both T_i and C lie on the M-line.
2) T_i lies on the M-line, whereas C does not.
3) T_i does not lie on the M-line.

Now each of these cases will be considered separately.

1) Here the robot is moving along the M-line towards T; thus T_i is between C and T (Fig. 5(a)). Since T_i is by definition on the Bug2 path and both T_i and Q are visible from C, then Q must be on the Bug2 path; and because of the condition 1) already given, Q must be further along the Bug2 path than T_i.
2) This case is shown in Fig. 5(b). If there are no obstacles crossing the M-line between points T_i and Q, then the lemma obviously holds. If, however, there is at least one such obstacle, then a hit point, H_j, would appear. By design of the Bug2 algorithm, the line segment T_iH_j is a segment of the Bug2 path. At H_j the Bug2 path would turn left and proceed along the obstacle boundary as shown. For each hit point, there must be a matching leave point. Where does the corresponding leave point, L_j, lie?

   Consider the triangle T_iCQ. Because of the visibility condition, the obstacle cannot cross the line segments CT_i or CQ. Also, the obstacle cannot cross the line segment T_iH_j, because otherwise some other hit point would have been defined between T_i and H_j. Therefore the obstacle boundary, and the corresponding segment of the Bug2 path, must cross the M-line somewhere between H_j and Q. This produces the leave point L_j. Thereafter, because of the condition 1) already given, the Bug2 path either goes directly to Q or meets another obstacle, in which case the same argument applies. Thus Q is on the Bug2 path, and it is further along this path than T_i.
3) Before considering this case in detail, we make two observations.

Observation 1: Within the assumptions of the lemma, if T_i is not on the M-line, then the current position C of the robot is not on the M-line either. Indeed, if T_i is not on the M-line, then there must exist an obstacle that produced the latest hit point, H_j, and then the intermediate target T_i; this obstacle prevents the robot from seeing any point Q on the M-line that would satisfy the requirement 2) already given.

Observation 2: If C is not on the M-line, then the segment $|CT_i|$ will never cross the open-line segment $|H_jT|$ ("open" here means that the endpoints of the segment are not included). Here H_j is the lastly defined hit point. Indeed, for such a crossing to take place, T_i must lie in the secondary semiplane (Fig. 6). For this to happen, the Bug2 path would have to proceed from H_j first into the main semiplane and then enter the secondary semiplane somewhere outside of the line segment $|H_jT|$ (otherwise, the leave point, L_j, would be established and the Bug2 path would stay in the main semiplane at least until the next hit point, H_{j+1}, is defined). Note, however, that any such way of entering the secondary semiplane would produce segments of the Bug2 path that are not contiguous (because of the visibility condition) to the rest of the Bug2 path. By the algorithm, no points on such segments can be chosen as intermediate targets T_i—which means that if the point C is in the main semiplane, then the line segments $|CT_i|$ and $|H_jT|$ never intersect.

Situations that fall into the case in question can in turn be divided into three groups.

3a) Point C is located on the obstacle boundary and $C = T_i$. This happens when the robot walks along a

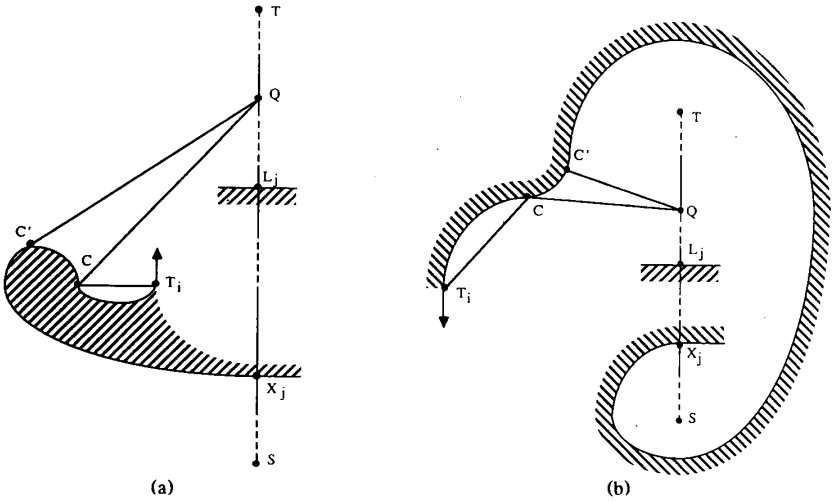

Fig. 7. Illustration for Lemma 2. (a) Case 3a. (b) Case 3b.

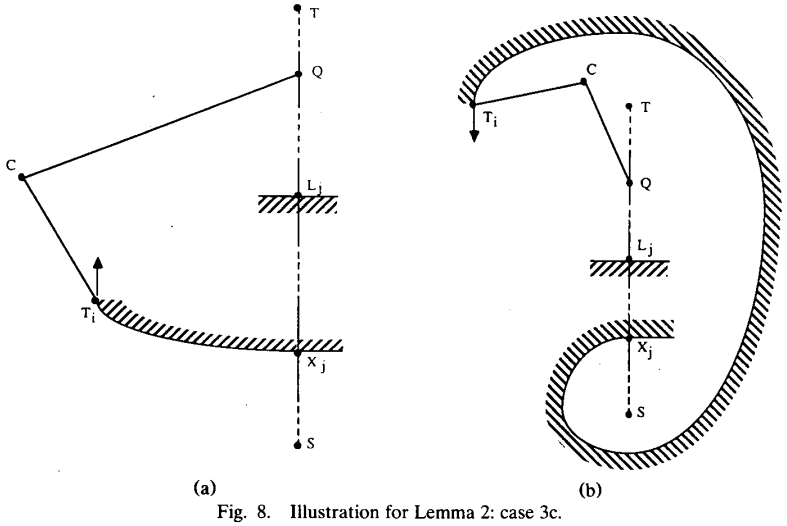

Fig. 8. Illustration for Lemma 2: case 3c.

locally convex obstacle boundary (point C', Fig. 7). Consider the curvilinear triangle $X_jC'Q$. Continuing the boundary segment (X_jC') after the point C', the obstacle (and the corresponding segment of the Bug2 path) will either curve inside the triangle, with $|QT|$ lying outside the triangle (Fig. 7(a)), or it will curve outside the triangle, leaving $|QT|$ inside (Fig. 7(b)). Since the obstacle can cross neither the line $|C'Q|$ nor the boundary segment (X_jC'), it (and the corresponding segment of the Bug2 path) must eventually intersect the M-line somewhere between X_j and Q before intersecting $|QT|$. The rest of the argument is identical to case 2.

3b) Point C is on the obstacle boundary and $C \neq T_i$ (Fig. 7). Consider the curvilinear triangle X_jCQ. Again, the obstacle can cross neither the line of visibility $|CQ|$ nor the boundary segment (X_jC), and so the obstacle (and the corresponding segment of the Bug2 path) will either curve inside the triangle, with $|QT|$ left outside of it, or curve outside the triangle, with $|QT|$ lying inside. The rest is identical to case 2.

3c) Point C is not on the obstacle boundary. Then a curvilinear quadrangle is formed, X_jT_iCQ (Fig. 8). Again, the obstacle will either curve inside the quadrangle, with $|QT|$ outside of it, or curve outside the quadrangle, with $|QT|$ lying inside. Since neither the lines of visibility $|CT_i|$ and $|CQ|$ nor the boundary segment (X_jT_i) can be crossed, the obstacle (and the corresponding segment of the Bug2

path) will eventually cross $|X_jQ|$ before intersecting $|QT|$ and form the leave point L_j. The rest of the argument is identical to case 2.

If the robot is currently located in the secondary semiplane, then it is indeed possible that a point that lies on the M-line and seems otherwise a good candidate for the next intermediate target T_i does not lie on the Bug2 path. Thus such a point should not even be considered. Such an example is shown in Fig. 4 where, while at the location C, the robot will reject the seemingly attractive point Q (Step 2 of the algorithm) because it does not lie on the Bug2 path. At this point, the convergence of the algorithm can be established.

Theorem: The algorithm VisBug-21 converges.

Proof: To generate the next intermediate target point T_i, the robot uses either its vision or its capability to move along the obstacle boundary (such as on a convex obstacle, where vision is of no help). By the very definition of the intermediate target point T_i, for any T_i defined at the given location C of the robot, T_i is reachable from C. According to the algorithm, the next step of the robot is always in the direction of the current T_i. This means that if the locus of points T_i is converging to T, so will the locus of points C. In turn, we know that if the locus of points T_i presents the path generated by the algorithm Bug2, then it indeed converges to T [8]. The main question then is whether all the points T_i generated by VisBug-21 lie on the Bug2 path.

All the steps of the procedure Compute T_i-21, except Step 4, explicitly test each candidate for the next T_i for being contiguous to the previous T_i and belonging to the Bug2 path. The only questionable points are the intermediate targets T_i on the M-line that are not required to be contiguous to the previous T_i; these are produced in Step 4 of the procedure. In such cases, the points T_i in question are chosen only if the robot's location C lies in the main semiplane, in which case the conditions of Lemma 2 apply. Therefore all the intermediate targets T_i generated by the algorithm VisBug-21 path lie on the Bug2 path.

V. Algorithm VisBug-22

A. Algorithm

The structure of this algorithm is quite similar to VisBug-21, except the robot does not try to ensure that all the intermediate targets T_i lie on the Bug2 path. Instead, it attempts to choose those of its intermediate targets that lie on the M-line as close to the target T as possible. This results in a different mechanism of convergence and in a behavior different from algorithm VisBug-21.

Consider an environment with given points S and T, and consider a third point, S', that lies on the M-line somewhere between S and T. As before, the term Bug2 path refers to the path produced by algorithm Bug2 between S and T. A quasi-Bug2 path segment is a contiguous path segment that starts at S' and produces a part of the path that algorithm Bug2 would have generated if points S' and T were its starting and target points, respectively. As point S' needs not be on the Bug2 path, a quasi-Bug2 path segment needs not be a segment of the Bug2 path.

The algorithm VisBug-22 will keep identifying points along the Bug2 path or a quasi-Bug2 path segment, until a better point (in terms of its proximity to T), S', is identified on the M-line. Then S' becomes the starting point of another quasi-Bug2 path segment, and the process repeats. As a result, unlike algorithms Bug2 and VisBug-21, where each defined hit point has its matching leave point, in VisBug-22 no such matching necessarily occurs. To be chosen as the starting point of the next quasi-Bug2 path segment, point S' must satisfy certain requirements that assure convergence (see subsection B, to follow).

The algorithm consists of the Main Body, which is identical to the main body of algorithm VisBug-21, and a procedure called Compute T_i-22, which produces the next intermediate target T_i for a given current position of the robot C, and also performs the test for target reachability. Initially, $C = S = T_i$.

Procedure Compute T_i-22: of algorithm VisBug-22. The procedure consists of the following steps.

- Step 1: If T is visible then define $T_i = T$; procedure stops.
  Else if T_i is on an obstacle boundary go to Step 3.
  Else go to Step 2.
- Step 2: Define point Q as the endpoint of the maximum length contiguous segment of the M-line, $|T_iQ|$, extending from T_i in the direction of T.
  If an obstacle has been identified crossing the M-line at point Q then define a hit point, $H = Q$; define $T_i = Q$; go to Step 3.
  Else define $T_i = Q$; go to Step 4.
- Step 3: Define point Q as the endpoint of the maximum length contiguous segment of the obstacle boundary, (T_iQ), extending from T_i in the local direction.
  If the obstacle has been identified crossing the M-line at a point $P \in (T_iQ)$, $|P,T| < |HT|$ and the line $|PT|$ does not cross the obstacle at P then define a leave point, $L = P$, define $T_i = P$, and go to Step 2.
  If the lastly defined hit point, H, is again identified and $H \in (T_iQ)$ then the target is not reachable; procedure stops.
  Else define $T_i = Q$; go to Step 4.
- Step 4: If T_i is on the M-line define $Q = T_i$, otherwise define $Q = H$.
  If points, $\{P\}$, on the M-line are identified such that $|S'T| < |QT|$, $S' \in \{P\}$ then find the point $S' \in \{P\}$ that produces the shortest distance $|S'T|$; define $T_i = S'$; go to Step 2.
  Else procedure stops.

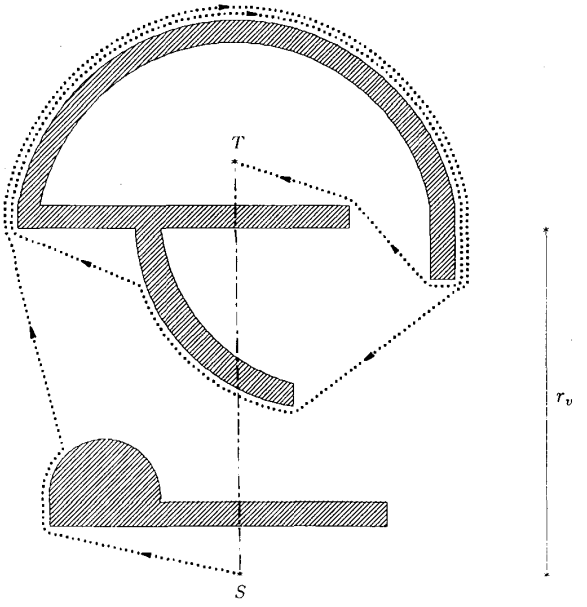

Fig. 9. Environment 1: path generated by VisBug-22; radius of vision r_v is larger than that in Fig. 3 (and equal to that in Fig. 4).

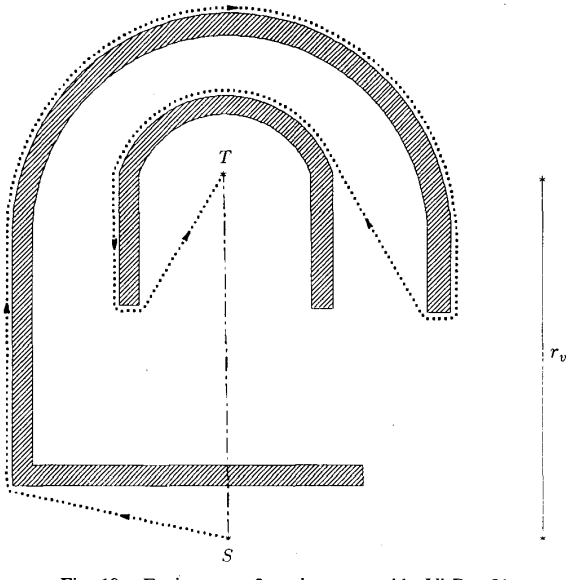

Fig. 10. Environment 2: path generated by VisBug-21.

B. Analysis

The performance of algorithm VisBug-22 is demonstrated in Figs. 3 and 9, where the same values of radius of vision r_v as for VisBug-21 in Figs. 3 and 4 are used. Compare this with the performance of algorithm Bug2 in the same environment (Fig. 2). Note that VisBug-21 and VisBug-22 can sometimes perform identically (Fig. 3). Observe that, in general, neither algorithm is superior to the other. For example, in Figs. 4 and 9, VisBug-21 does better than VisBug-22, whereas the opposite is true in the examples shown in Figs. 10 and 11.

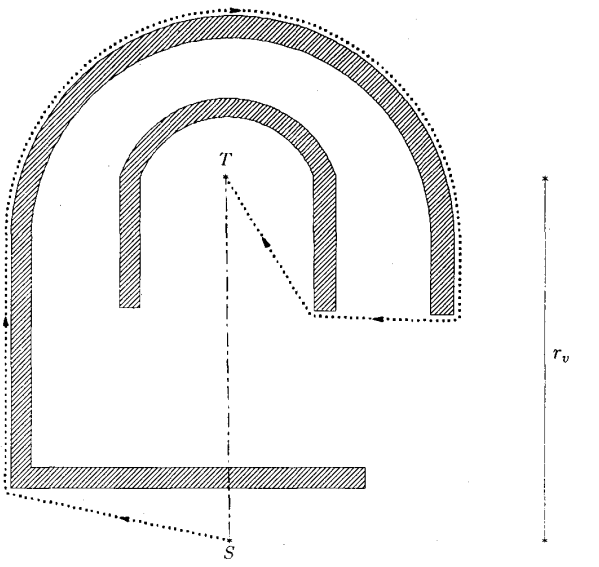

Fig. 11. Environment 2: path generated by VisBug-22.

The convergence of algorithm VisBug-22 follows from the fact that all the starting points, S', of the successive quasi-Bug2 path segments lie on the M-line, and they are organized in such a way as to produce a finite sequence of distances shrinking to T:

$$|S_1'T| > |S_2'T| > |S_3'T| > \cdots \tag{12}$$

where points S' are numbered in the order of their appearance.

VI. Conclusion

A paradigm has been presented for incorporating sensory range data into the robot motion-planning operation. The key issues that we address are, first, the design of provable motion planning algorithms based on local range data, and second, interaction between the subsystems responsible for gathering sensor data and path planning. Along these lines, two algorithms are described that not only use on-line sensory data for motion planning, but also actively prescribe directions at which range data should be gathered at each step. Our robot will usually define its intermediate target by scanning a limited sector, which includes an obstacle boundary that it tries to follow, instead of scanning the whole 360° circumference. As experimentalists know, the latter feature is quite important for real-time operation: for example, a typical laser range finder is a relatively slow sequential scanning device.

The described algorithms, *ad hoc* as they might look at first glance, do guarantee convergence and use the input information quite effectively. In fact, the rather natural assumptions of our model—that input information is only of local character and that obstacles can be of arbitrary shapes—seem to limit a number of options for designing

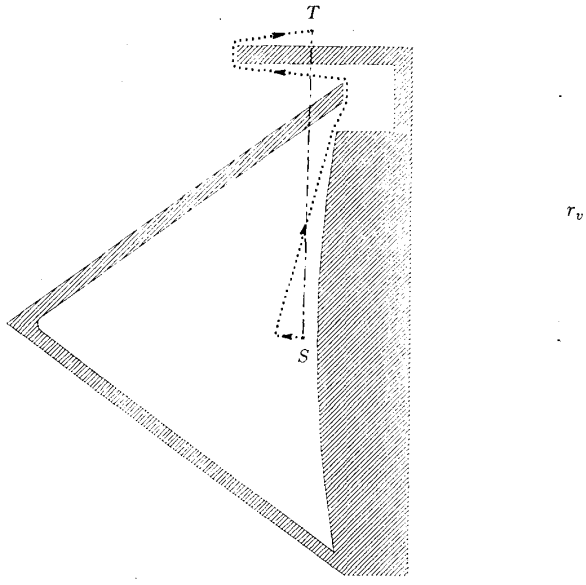

Fig. 12. Performance of algorithm VisBug-21 with smaller radius of vision.

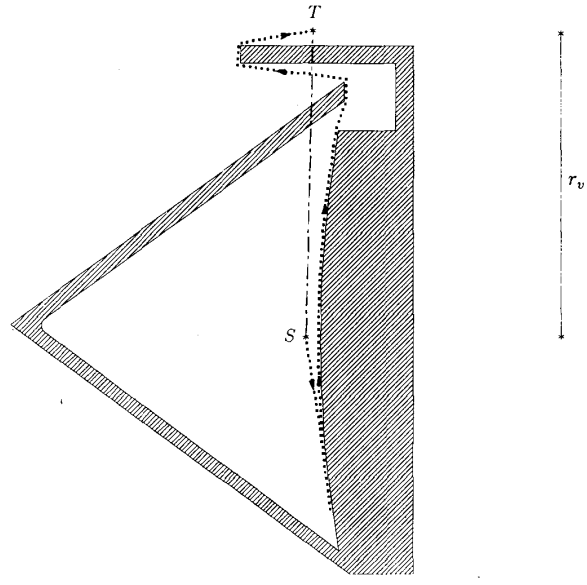

Fig. 13. Performance of algorithm VisBug-21 with larger radius of vision (compare with Fig. 12).

algorithms (an interesting discussion on the effect of the former assumption can be found in [9]). For example, a popular technique that makes use of "visibility graphs" based on common tangent lines to obstacles would not work here. On the other hand, because of the latter assumption, memorizing incremental maps would require prohibitive amounts of memory.

This work indicates that the relationship between the amount of sensor information and the efficiency of motion-planning algorithms is quite complex. Although no worst-case or average-case bounds can be given at this time, our general conclusion is that, as long as the robot sensors provide only partial information—which all existing sensors do—the quality of the generated paths cannot be guaranteed to improve via better algorithms or better sensors. This indirect relationship is common to the general problem of motion planning with uncertainty, rather than is a consequence of a restricted model, of specific algorithms, or specific sensing media. Besides theoretical interest, the realization of this fact may have a major effect on how one chooses sensing media for a real mobile robot.

We show, for example, that although algorithm VisBug-21 is never inferior to the "tactile" algorithm Bug2 (see Lemma 1), surprisingly, a larger radius of vision does not necessarily result in a shorter path (compare Figs. 12 and 13). The situation becomes even more complex in the case of algorithm VisBug-22. Although the paths it generates are typically significantly shorter than those produced by algorithm Bug2, this cannot be guaranteed (compare Figs. 2 and 9). And again, a larger radius of vision does not guarantee shorter paths (compare Figs. 3 and 9). Note also that, in general, neither algorithm is superior to the other. For example, in the scene shown in Figs. 4 and 9, algorithm VisBug-21 outperforms algorithm VisBug-22, whereas the opposite is true in the scene shown in Figs. 10 and 11.

References

[1] C. Yap, "Algorithmic motion planning," in *Advances in Robotics, Vol. 1: Algorithmic and Geometric Aspects*, J. Schwartz and C. Yap, Eds. Hillsdale, NJ: Erlbaum, 1987, pp. 95–143.

[2] M. Hebert, "Outdoor Scene Analysis Using Range Data," in *Proc. 1986 IEEE Int. Conf. on Robotics and Automation*, San Francisco, CA, Apr. 1986, pp. 1426–1432.

[3] L. Matthies and S. Shafer, "Error modeling in stereo navigation," *IEEE Trans. Robot. Autqmat.*, June 1987.

[4] A. Elfes, "Sonar based real-world mapping and navigation," *IEEE Trans. Robot. Automat.*, vol. RA-3, no. 3, pp. 249–265, June 1987.

[5] D. Kriegman, E. Triendl, and T. Binford, "A mobile robot: Sensing, planning and locomotion," in *Proc. 1987 IEEE Int. Conf. on Robotics and Automation*, Raleigh, NC, Apr. 1987, pp. 402–408.

[6] N. Rao, S. Iyengar, C. Jorgensen, and C. Weisbin, "On terrain acquisition by a finite-sized mobile robot in plane," in *Proc. 1987 IEEE Int. Conf. on Robotics and Automation*, Raleigh, NC, Apr. 1987, pp. 1314–1319.

[7] R. Cole and C. Yap, "Shape from probing," New York University, Courant Institute, Tech. Rep. 104, Dec. 1983.

[8] V. Lumelsky and A. Stepanov, "Dynamic path planning for a mobile automaton limited information on the environment," *IEEE Trans. Automat. Contr.*, vol. AC-31, no. 11, Nov. 1986, pp. 1058–1063.

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

[10] J. Borenstein and Y. Koren, "Real-time obstacle avoidance for fast mobile robots," *IEEE Trans. Syst. Man Cybern.*, vol. SMC-19, no. 5, pp. 1179–1187, Sept./Oct. 1989.

[11] T. Skewis, J. Evans, V. Lumelsky, B. Krishnamurthy, and B. Barrows, "Motion planning for a hospital transport robot," Yale University, Center for Systems Science, Tech. Rep. 9001, Mar. 1990.

Vladimir J. Lumelsky (M'80–SM'83) received the Ph.D. degree in applied mathematics from the Institute of Control Sciences (ICS), U.S.S.R. National Academy of Sciences, Moscow, in 1970.

From 1967 to 1975 he held academic positions of Junior Researcher and Senior Research Fellow in ICS, conducting research in pattern

recognition, cluster analysis, factor analysis, and control systems. Concurrently, from 1970 to 1975, he served as Adjunct Professor at the Moscow Institute of Radioelectronics and Automation. From 1976 to 1980 he was with the Ford Motor Company Scientific Laboratories, Dearborn, MI, doing research in robotics, image processing, and industrial automation. From 1980 to 1985 he was on the research staff at the General Electric Research Center, Schenectady, NY, doing research in robotics, pattern recognition, system engineering, and control theory. Since 1985 he has been on the faculty of the Department of Electrical Engineering at Yale University. His research interests are in robotics, image processing, pattern recognition, and control theory.

Dr. Lumelsky is a member of the IEEE, the ACM, and Robotics International of the SME.

Tim Skewis (S'86) received the B.S. degree in electrical engineering from Lehigh University, Bethlehem, PA, in 1983; the M.S. degree in electrical engineering from Yale University, New Haven, CT, in 1986; and is currently a candidate for the Ph.D. degree in electrical engineering at Yale University.

During 1984–1987, he was on the technical staff of Unimation Inc., Danbury, CT. He developed motion planning and trajectory software for the UniVAL robot arm controller and was awarded U.S. patent #4,773,025. During 1987–1989 he was on the technical staff of the Transitions Research Corporation, Danbury, CT, and developed navigation software for HelpMate, a mobile robot for hospital transport tasks. His research interests include sensor-based motion, mobile robots, and graphical robot simulation.

Mr. Skewis is a recipient of a CT High Technology Scholarship and is