ROBOT NAVIGATION IN UNKNOWN TERRAINS USING LEARNED VISIBILITY GRAPHS. PART I: THE DISJOINT CONVEX OBSTACLE CASE

B. John Oommen*
S.S. Iyenger and S.V. Nageswara Rao**
R.L. Kashyap***

SCS-TR-86

February 1986

*School of Computer Science, Carleton University, Ottawa K1S 5B6, Canada

**Department of Computer Science, Louisiana State University, Baton Rouge, LA 70803, U.S.A.

***Department of Electrical Engineering, Purdue University, West Lafayette, IN 47907, U.S.A.

This research was supported by the Natural Sciences and Engineering Research Council of Canada.

ROBOT NAVIGATION IN UNKNOWN TERRAINS USING LEARNED VISIBILITY GRAPHS. PART I: THE DISJOINT CONVEX OBSTACLE CASE

B. John Oommen

School of Computer Science
Carleton University
Ottawa: K1S 5B6, CANADA

S.S. Iyengar and S.V. Nageswara Rao

Department of Computer Science
Lousiana State University
Baton Rouge, LA 70803, USA

R.L. Kashyap

Department of Electrical Engineering
Purdue University
West Lafayette,IN 47907,USA

ABSTRACT

The problem of navigating an autonomous mobile robot through an unexplored terrain of obstacles is the focus of this paper. The case when the obstacles are 'known' has been extensively studied in literature. In this paper, we consider *completely* unexplored obstacle terrain. In this case, the process of navigation involves both learning the information about the obstacle terrain and path planning. We present an algorithm to navigate a robot in an unexplored terrain that is arbitrarily populated with disjoint convex polygonal obstacles in the plane. The navigation process is constituted by a number of traversals; each traversal is from an arbitrary source point to an arbitrary destination point. The proposed algorithm is proven to yield a convergent solution to each path of traversal. Initially, the terrain is explored using a rather primitive sensor and the paths of traversal made may be sub-optimal. The *visibility graph* that models the obstacle terrain is incrementally constructed by integrating the information about the paths traversed so far. At any stage of learning, the partially learnt terrain model is represented as a *learned visibility graph*, and it is updated after each traversal. It is proven that the learned visibility graph converges to the visibility graph with probability one, when the source and destination points are chosen randomly. Ultimately, the availability of the complete visibility graph enables the robot to plan globally optimal paths, and also obiviates the further usage of sensors.

Keywords and Phrases:

robot navigation, path planning, motion, vision for navigation, learning, visibility graphs, optimal paths.

Table of Contents

---

1. INTRODUCTION

Robotics is one of the most important and challenging areas of computer science. Robots have been increasingly applied in carrying out tedious and monotonous tasks such as normal maintenence, inspection, etc. in industries. In hazardous environments such as nuclear power plants, underwater, etc., robots are employed to carry out human-like operations. However, as a scientific discipline the area of robotics is fascinating from the directions of challenge, application and results. Perhaps the most interesting aspect of robotics is the gamut of underlying problems that spans from extremely abstract mathematical problems to highly pragmatic ones.

There are many existing robots capable of carrying out intelligent and autonomous operations. Examples are SHAKEY [18], the JPL robot [21], HILARE [8], the Stanford Cart [17], the CMU terrigator and Neptune robots [24], HERMIES [25], etc. There are many facets to a completely autonomous robot; among them some of the actively pursued fields are knowledge representation, task planning, sensor interpretation, terrain model acquisition, dynamics and control, specialized computer architectures, algorithms for concurrent computations, path planning and navigation, and coordinated manipulation.

Path planning and navigation is one of the most important aspects of autonomous roving vehicles. The *find-path* problem deals with navigating a robot through a completely known terrain of obstacles. This problem is entensively studied and solved by many researchers - Brooks [2], Brooks and Lozano-perez [3], Gouzenes[9], Lozano-perez [14], Lozano-perez and Wesley [15], and Oommen and Reichstein [19] are some of the most important contributors. Another problem deals with navigating a robot through an unknown or partially explored obstacle terrain. Unlike the find-path problem, this problem has not been subjected to a rigourous mathematical treatment, and this could be attributed, at least partially, to the inherent nature of this problem. However, this problem is also researched by many scientists - Chatila [4], Chattergy [5], Crowley [6], Giralt et al [8], Iyengar et al [10,11], Laumond [12], Rao et al [20], Turchen and Wong [22], and Udupa [23] present many important results. As pointed out in the literature, the navigation through unknown terrain involves activities such as model acquisition and learning, sensing, etc. which are absent in find-path problem.

In this paper, we deal with the problem of navigation through an unexplored terrain. A rather elementary method involves sensing the obstacles and avoiding them in a localized manner. In more sophisticated methods, the terrain is

explored as the robot navigates. Iyengar et al [10,11] propose a technique that 'learns' the terrain model as the robot navigates. Initially the robot uses the sensor information to avoid obstacles. And the terrain model is incrementally learnt by integrating the information extracted from the earlier traversals. In this method, the partially built model is used to the maximum extent in path planning, and the regions where no model is available, are explored using sensors. Another imortant aspect is to bound the obstacles using simple polygons. The free space is spanned by convex polygons. These constituent polygons are updated as the learning proceeds; as a result the polygons that bound the obstacle shrink in size, and the polygons that span the free space grow in size. Such a strategy provides a way to approximate arbitrary shaped obstacles by polygons and is also benifitted by the available computational geometry and other related algorithms found in [1,7,13,16,26]. However there are limitations on the technique of Iyengar et al [10,11]. The proposed algorithm does not yield a convergent solution in all cases.

In this paper, we propose a technique for navigation in an unexplored terrain when the terrain is populated with disjoint convex polygonal obstacles. In precise terms, the method proposed in this paper is proven for convergence in terms of planning paths, and also in acquiring the entire terrain model through learning. As an endeavour to include learning in the navigation process and to formalize the scheme, we now view the problem in a completely new framework. We assume that the robot begins the navigation in a completely unexplored terrain of finite dimensions. The terrain is populated with stationary obstacles. However, as opposed to the work done earlier, in this paper, we shall not cystallize our terrain in terms of a Voronoi diagram. Rather, we shall compute and mantain a graph termed as the **Learned Visibility Graph** (*LVG*). To obtain the *LVG*, the robot initially navigates through the obstacles using a local navigation technique. This technique, which is a 'hill climbing technique', is shown to converge. In the process of local navigation, the robot manipulates the *LVG*. It is shown, in this paper, that the *LVG* ultimately converges to the actual visibility graph (*VG*) of the obstacle terrain with probability one.

The organization of this paper is as follows: Section 2 introduces the definitions and notations used subsequently. The local navigation technique that incorporates learning and path planning is presented in section 3. The convergence of the proposed algorithm is proven. In section 4, the power of local navigation algorithm is enhanced by incorporating backtracking. As a result the interior restriction on the obstacle terrain is relaxed. The modified procedure is also

proven for correctness. In section 5, a global navigation strategy that makes use of the existing terrain model to the maximum extent is presented. The important result that the learning eventually becomes complete is presented in section 6. The execution of the navigation algorithms on a sample obstacle terrain is presented in section 7.

2. NOTATIONS AND DEFINITIONS

The robot is initially placed in a completely unexplored terrain, and it is required to undertake a number of traversals; each traversal is from an arbitrary source point S to an arbitrary destination point D. The robot is treated as a point in a plane that is arbitrarily populated with stationary disjoint and convex polygonal obstacles. Let $W=\{w_1,w_2,\ldots,w_k\}$, be the set of obstacles in the terrain R, where w_i is convex polygonal obstacle. Furthermore, the obstacles polygons are mutually non-intersecting and non-touching. Let V be the union of the vertices of all the obstacles, and Z be the set of all edges of the obstacle polygons.

Of paramount importance to this entire problem is a graph termed as the **Visibility Graph** (VG). The VG is a pair (V,E), where

(i) V is the set of vertices of the obstacles, and.

(ii) E is the set of edges of the graph. A line joining the vertices v_i to v_j forms an edge $(v_i,v_j) \in E$ if and only if it is an edge of an obstacle or it is not intercepted by any other obstacle. Formally, $(v_i,v_j) \in E$ iff (i) $(v_i,v_j) \in Z$ or (ii) $(v_i,v_j) \cap Z=\phi$.

Visibility Graphs (VG) have been extensively studied in the computaional geometry literature and are used in motion planning by Lozano-Perez[15] and many other researchers (see the survey paper of Whitesides[26]). However, in this context it is important to note that the VG is initially unknown to the robot inasmuch as the obstacles and their locations are unknown. Although the VG is completely unknown initially, it is learnt during the initial stages of the navigation process. The partially learnt VG is augmented after each traversal by integrating the information extracted from the local navigation.

The process of *learning* is completed when the entire VG of the obstacle terrain is completely built. Before the robot attains this state, the VG is only partially built. The robot graduates through various intermediate stages of learning during which the VG is incrementally constructed. These intermediate stages of learning are captured in terms of the **Learned Visibility Graph** (LVG), which is

defined as follows: $LVG = (V^*, E^*)$, where, $V^* \subseteq V$ and $E^* \subseteq E$. The LVG is initially empty, and is incrementally built. Ultimately, the LVG converges to the exact VG.

We assume, throughout this paper, that the robot is equipped with a sensor capable of measuring the distance to an obstacle in any specified direction. The availability of the present-day range sensors justifies this assumption. Also, we assume that the robot is equipped with sensors which enable the navigation along the edges of the obstacles. A sensor system constituted by a set of primitive proximity sensors can impart such an ability to the robot. Hence, the robot can navigate arbitrarily close to the obstacle edges.

The interior of any polygon ξ is denoted by $INT(\xi)$. The straight line from the point P to the point Q is denoted by $\vec{PQ}$. Further, η_{PQ} denotes the unit vector along the straight line $\vec{PQ}$.

3. LOCAL NAVIGATION AND LEARNING

When the robot navigates in a completely unexplored terrain, its path of navigation is completely decided by the sensor readings. The obstacles in the proximity of the source point are scanned and a suitable path of navigation is chosen. This localized nature of the local navigation makes a globally optimal path unattainable in a terrain with an arbitraty distribution of obstacles. However, local navigation is essential during the initial stages of the navigation. The information acquired during the local navigation is integrated into the partially built terrain model. No local navigation is resorted to in the regions where the existing terrain model is sufficient for planning globally optimal paths.

In this section, we propose a local navigation technique that enables the robot to detect and avoid obstacles along the path from an arbitrary source point, S to an arbitrary destination point, D. The robot is equipped with a primitive motion command MOVE(S,A,λ), where

(a) S is the source point, namely, the place where the robot is currently located.

(b) A is the destination point which may or may not be specified.

(c) λ is the direction of motion, which is always specified.

If A is specified, then the robot moves from S to A in a straight line path. In this case, the direction of motion λ is the vector η_{SA}, the unit vector in the

direction of $\vec{SA}$. If A is not specified, then the robot moves along the direction λ as follows: If the motion is alongside an edge of an obstacle, then the robot moves to the end point of the edge along the direction λ. This end point is returned to the calling procedure as point A as in Fig. 1(a). If motion is not alongside an edge of an obstacle, then the robot traverses along the direction λ till it reaches a point on the edge of an obstacle as shown in Fig. 1(b). This point is returned as the point A to the calling procedure.

In the remainder of this section, we describe the local navigation algorithm. For the treatment in this section we assume that the obstacles do not touch or intersect the boundaries of the terrain R. In other words, the obstacles are properly contained in the terrain R. This is formally represented as

$$\bigcup_{i=1}^{k} INT(w_i) \subseteq INT(R) \qquad (1)$$

As a consequence of this assumption there is always a path from a source point S to a destination point D. However, this restriction is removed in the next section.

We present the procedure NAVIGATE-LOCAL that uses a *hill-climbing* technique to plan and execute a path from an arbitrary source point S to an arbitrary destination point D. The outline of this procedure is as follows: The robot moves along $\vec{SD}$ till it gets to the nearest obstacle. It then circumnavigates this obstacle using a local navigation strategy. The technique is then recursively applied to reach D from the intermediate point. Further, apart from path planning the procedure also incorporates the learning phase of acquiring the VG.

We now concentrate on local navigation strategy. The robot moves along the direction η_{SD} till it encounters an obstacle at a point A which is on the obstacle edge joining the two vertices, say, A_1 and A_2. At this point the robot has two possible directions of motion: along $\vec{AA}_1$ or $\vec{AA}_2$ as shown in Fig. 2. We define a local optimization criterion function J as follows:

$$J = \eta_{SD} . \lambda \qquad (2)$$

where λ is a unit vector along the direction of motion.

Let λ_1 and λ_2 be the unit vectors along $\vec{AA}_1$ and $\vec{AA}_2$ respectively. Let $\lambda^* \in \{\lambda_1, \lambda_2\}$ maximize the function J given in equation (2). The robot then undertakes an exploratory traversal along the direction $-\lambda^*$ till it reaches the corresponding vertex called the **exploratory** vertex. At this exploratory point the

terrain is explored using the procedure UPDATE-VGRAPH. Then the robot retraces along the locally optimal direction λ^* till it reaches the other vertex S^*, whence it again calls UPDATE-VGRAPH. The procedure NAVIGATE-LOCAL is recursively applied to navigate from S^* to D.

The procedure UPDATE-VGRAPH implements the learning component of the robot navigation. Whenever the robot reaches a new vertex v_i this vertex is added to the LVG. From this vertex, the robot beams its sensor in the direction of all the **existing** vertices of the LVG. The edge (v_i,v) is added to the edge set E^*, corresponding to each vertex $v \in V^*$ visible from v_i. The algorithm is formally presented as follows:

---

procedure UPDATE-VGRAPH(v);
input: The vertex v which is newly encountered.
output: The updated $LVG = (V^*,E^*)$.
Initially the LVG is set to (ϕ,ϕ).
comment: $DIST(v_1,v_2)$ indicates the euclidian distance
between vertices v_1 and v_2, if they are visible
to each other.
This is the auxiliary information stored aong with the LVG.
begin
1. $V^* = V^* \cup \{v\}$;
2. **for** all $v_1 \in V^* - \{v\}$ **do**
3. **if** (v_1 is visible from v) **then**
4. DIST$(v_1,v) = |\overrightarrow{v_1 v}|$;
5. $E^* = E^* \cup \{(v_1,v)\}$;
6. **else**
7. $DIST(v_1,v) = \infty$;
endif
endfor;
end;

The procedure NAVIGATE-LOCAL uses the motion primitive motion command MOVE and the procedure UPDATE-VGRAPH during execution. This procedure is formally described follows:

---

procedure NAVIGATE-LOCAL(S,D);
Input: The source point S and the destination point D
Output: A sequence of elementary MOVE commands
begin

1. **if** (D is visible from S) **then**
2. MOVE(S,D,η_{SD})
3. **else**
4. **if** (S is on an obstacle and the obstcle obstructs its view) **then**
5. compute $\{\lambda_1,\lambda_2\}$, the two possible directions of motion;
6. λ^* = direction maximizing $\lambda_i.\eta_{SD}$;
7. **if** (S is a vertex) **then**
8. **if** ($S \notin V^*$) **then** UPDATE-VGRAPH(S);
9. MOVE(S,S^*,λ^*);
10. **else**
11. MOVE(S,S_1,$-\lambda^*$); { make exploratory trip to S_1 }
12. **if** ($S_1 \notin V^*$) **then** UPDATE-VGRAPH(S_1);
13. MOVE(S_1,S^*,λ^*); { retrace steps to S^* }
14. **if** ($S^* \notin V^*$) **then** UPDATE-VGRAPH(S^*);
    endif;
15. NAVIGATE-LOCAL(S^*,D);
16. **else** { move to next obstacle }
17. MOVE(S,S^*,η_{SD}); { move to next obstacle alnog η_{SD} }
18. NAVIGATE-LOCAL(S^*,D);
    endif;

endif;
end;

We shall now prove that the procedure NAVIGATE-LOCAL converges. In a subsequent section we shall show that the *LVG* updated using UPDATE_VGRAPH ultimately converges to the exact *VG*.

THEOREM 1:

The procedure NAVIGATE-LOCAL always finds a path from S to D in finite time.

PROOF: There is always a path from S to D as per the assumption in equation (1). Hence, it suffices to prove that the recursion is correctly applied. We shall prove that this is indeed the case, and also that the MOVE operations minimize the projected distance along η_{SD}. Then, the theorem follows from the fact that total the number of vertices of all the obstacles is finite.

CASE I - Terminating Step:

If D is visible from S the procedure terminates as per line 2 in NAVIGATE-LOCAL. In this case, the projected distance of the path traversed by the robot is reduced from $|SD|$ to zero in one step.

CASE II - Recursive Steps:

This step consists of three mutually exclusive and collectively exhaustive cases. In each case, we shall show that each execution of MOVE(S, S^*, λ^*) forces the following *strict* inequality:

$$|SD| > \vec{S^*D}.\eta_{SD} \qquad (3)$$

II(a): The point D is not visible from S, and S is not on the boundary of the obstructing obstacle. Fig. 3(a) depicts this scenario. The lines 17 and 18 of NAVIGATE-LOCAL give the corresponding actions. In this case, the motion is along η_{SD} and every point A along this vector satisfies the relation $|\vec{AD}| = \vec{AD}.\eta_{SD}$. Thus the motion along η_{SD} to A gives the following equality: $|\vec{SD}| = |\vec{SA}| + \vec{AD}.\eta_{SD}$, whence,

$$|\vec{SD}| > \vec{AD}.\eta_{SD} \qquad (4)$$

The equation is particularly true for $A = S^*$, and hence the result.

II(b) The point D is not visible from S and S lies on the edge of the obstructing obstacle.

Case (i): S does not correspond to a vertex of the *LVG* as shown in Fig. 3(b). If $\lambda_i.\eta_{SD} = 0$, then the edge is orthogonal to η_{SD}. In this case either direction does not decrease the projected distance. It is to be noted that this

sitaution can occur at most once for an obstacle encountered during the navigation, and hence can not persist. But, after the robot completes the MOVE corresponding to this step, the robot is located at a vertex. Since this is covered in case (ii), we shall only consider the case when $\lambda_i.\eta_{SD} \neq 0$. Let λ^* be the direction in which $\lambda_i.\eta_{SD} > 0$. The situation is shown in Fig. 3(b). Hence, the included angle S^*SD is less than $\Pi/2$, and $|\vec{SD}| - |\vec{S^*D}.\eta_{SD}| = |\vec{SS^*}| \cos(S^*SD) > 0$. Hence, the execution of MOVE(S,S^*,λ^*) ensures the inequality given in equation (3).

In this case, the robot temporarily diverges from the locally optimal path. This trip being purely exploratory does not contribute to the navigation. It is to be noted that this trip takes finite time.

Case (ii): The point S is located at a vertex of an obstacle. Fig. 3(c) depicts the situation. In this case, the edges of the convex polygon meet at S. Let the direction λ^* be the direction that maximizes $\lambda_i.\eta_{SD}$. Because the obstacle is convex, the angle between the edges at S is less than Π. Thus the anlge S^*SD is less than $\Pi/2$. Using the arguements of case (i), we conclude that the execution of MOVE operation satisfies $|\vec{SD}| - |\vec{S^*D}.\eta_{SD}| > 0$.

Hence the theorem. □

Observe that if we had only *one* polygonal obstacle, we could have gone around the obstacle in a systematic way, i.e., either in a clockwise or an anti-clockwise direction, till we reach a point from which D is visible. However, the problem becomes more difficult when there are more than one obstacles. In this case, the motion must be made in such a way that a criterion function is minimized. We have chosen to minimize the projected distance along SD by maximizing the function J in equation (2). This method may not give rise to a globally optimal path as shown in Fig. 4. Such counter examples exist for any localized navigation scheme for the want of global information about the obstacles.

It is easy to conceive of a scheme in which the sensor readings can give all the visible edges and vertices of the obstacles. In such a case there may be a shorter path for navigation. However, we choose to go along the path dictated by NAVIGATE-LOCAL so that the *LVG* can be updated in the process of navigation while the projected distance along η_{SD} is minimized. The procedure UPDATE-VGRAPH makes sure that edges to all the visible vertices of *LVG* are added to E^* when a new vetex is added to V^*. Fig. 5 shows the salient features of the approach. The vertices u_1,u_2,u_3,u_4,v_1 are presently existing in *LVG*, and

the edges $(u_1,u_2),(u_3,u_4)$ are also present. A globally optimal path is Sv_4D. We, however, choose the path $SS^*v_2S^*v_3v_4D$ - which is only suboptimal. The exploratory traversal to v_2 yeilds the visibility information about the vertices v_2,v_3,v_4, which is obtained from the sensor information. It is conceivable that the procedure NAVIGATE-LOCAL can be modified to avoid exploratory trips along the explored edges of the obstacles. However, we regard this issue as rather straightforword and prefer not to elaborate on it.

4. LIMITATIONS OF LOCAL NAVIGATION AND A SOLUTION

The procedure NAVIGATE-LOCAL introduced in the previous section always yields a path if one exists and if the obstacles do not touch the terrain boundaries. These preconditions are implicitly satisfied as a consequence of the assumption in equation (1). The relaxation of this assumption results in two conditions, in which the procedure NAVIGATE-LOCAL is not guaranteed to halt:

(a) There is no path existing between the source point S to the destination point D. In this case, a single obstacle blocks all the paths from S to D. Fig. 6 shows some of such cases. It is to be noted that when the robot starts moving around the obstacle, its way is blocked in both possible directions.

(b) The angle between the obstacle edge and the terrain boundary is less than $\Pi/2$. In this case we assume that there is a path existing between S and D, or stated equivalently no obstacle blocks all the paths between S and D. In such a case the robot may be forced to move to the dead corner formed by the obstacle and terrain boundary. At this point the robot has no further defined moves. The robot starting at S ges into the dead-corner at S^*. This situation is depicted in Fig. 7.

In this section, we relax the condition in (1), and enhance the capability of NAVIGATE-LOCAL by imparting to it the ability to *backtrack*. The robot backtracks (by invoking procedure BACKTRACK) whenever it reaches a point from which no further moves are possible (see Fig. 8). This procedure intelligently guides the robot in the process of retracing steps. That is, the robot backtracks along the edges of the obstructing obstacle till an edge (S,S_1), that makes an angle less than $\Pi/2$ with η_{SD} is encountered. The fact that such an edge exists is guaranteed because of the convexity of the obstacles. The search for this edge is peroformed by the while loop of lines 3-6 of procedure BACKTRACK. As a result the robot moves to a point from which the NAVIGATE-LOCAL can take over. If for the same obstacle, the robot has to backtrack twice, then there is no

path between S and D. In other words, if a path from S to D exists, then the robot needs to backtrack at most once along the edges of any obstacle. These aspects are further discussed subsequently in this section.

---

procedure BACKTRACK(S,D,S^*);

Input: The point D is the destination point.

S is a dead corner,i.e., a vertex of an obstacle and is also on the boundary of the terrain.

Output: A sequence of MOVEs from S in such a way that if a path exists, then it can be determined using NAVIGATE-LOCAL. The location S^* is returned to the calling procedure.

begin

1. $S_1 = S$;
2. λ^* = only permitted direction of motion on the obstacle;
3. **while** ($\vec{SD}.\lambda^* < 0$) **do**
4. MOVE(S_1,S^*,λ^*);
5. $S_1 = S^*$;
6. λ^* = only permitted direction of motion on the obstacle;

endwhile;

end;

The convergence of the procedure BACKTRACK is proved in the following theorem.

THEOREM 2:

The procedure BACKTRACK leads to a solution to the navigation problem, if one exists.

PROOF: The crux of the theorem is to prove that the procedure BACKTRACK terminates in all cases. In otherwords, there exists an edge that makes an angle less than $\Pi/2$ with η_{SD}. Fig. 8 shows the scenario. Consider the line SQ, a normal to SD at S. Because the obstacle is convex, the normal line SD at S must intersect the obstacle again, and at this point the corresponding edge makes angle less than $\Pi/2$ with $\vec{SD}$. Thus, the required vertex is found just after this edge because after this edge the first vertex indeed has a smaller value for the projected distance along η_{SD}. Hence, by moving along the boundary in this direction the procedure BACKTRACK

will take the robot to a place from which NAVIGATE-LOCAL can be applied. □

We note that if a path exists the further execution of NAVIGATE-LOCAL will not lead to a dead-end formed by the same obstacle. That is because, if the procedure BACKTRACK leads the robot to another "dead-end" on the same obstacle, clearly, the robot can not navigate across the obstacle. Hence, no path exists between S and D.

Let the procedure NAVIGATE-LOCAL with the enhanced capability to backtrack be called procedure NAVIGATE-LOCAL-WITH-BACKTRACK. This procedure utilizes NAVIGATE-LOCAL to navigate till the robot encounters a dead-end. At this point, the procedure BACKTRACK is invoked, after which the NAVIGATE-LOCAL is resorted to. The navigation is stopped if no path exists between S and D. The correctness of proof of procedure NAVIGATE-LOCAL-WITH-BACKTRACK easily follows from the arguements of this section. Similarly the formal statement of procedure NAVIGATE-LOCAL-WITH-BACKTRACK easily follows from those of NAVIGATE-LOCAL and BACKTRACK and is omitted for the sake of brevity.

5. GLOBAL NAVIGATION

The procedures described in the preceeding sections enable a robot to navigate in an unexplored terrain. Such a navigation involves the usage of sensor equipment and traversing the exploratory trips. The navigation paths are not necessarily globally optimal from the path planning point of view. However, the extra work carried out in the form of learning is inevitable because of the lack of information about the obstacles. Furthermore, the *LVG* is gradually built as a result of learning.

In the regions where the visibility graph is available, the optimal path can be found by computing the shortest path from the source point to the destination point on the graph. The computation can be carried out in quadratic time in the number of nodes of the graph by using the Dijkstra's algorithm [1]. Such a trip can be obtained by using only computations on the *LVG* and not involving any sensor operations.

We shall now propose a technique that utilizes the available *LVG* to the maximum extent in planning navigation paths. In the regions where no *LVG* is available, the procedure NAVIGATE-LOCAL is used for navigation. In these regions the *LVG* is updated for future navigation. The outline of the global

navigation strategy as follows:

---

procedure NAVIGATE-GLOBAL(S,D);
begin
1. Compute-Best-Vertices(S^*,D^*);
2. NAVIGATE-LOCAL-WITH-BACKTRACK(S,S^*);
3. Move-On-LVG(S^*,D^*);
4. NAVIGATE-LOCAL-WITH-BACKTRACK(D^*,D);

end

Given S and D, two nodes S^* and D^* on the existing LVG are computed. The robot navigates from S to S^* using local navigation. Then the navigation from S^* to D^* is along the optimal path computed using the LVG. Again, from D^* to D the local navigation is resorted to. Computation of S^* and D^*, corresponding to line 1 of NAVIGATE-GLOBAL, can be carried out using various criteria. We suggest three such possible criteria below:

Criterion A: S^* and D^* are the nodes of the LVG closest to S and D. The computation of these nodes involes $O(|V^*|)$ distance computations.

Criterion B: S^* is a vertex such that it is the closest to the line $\vec{SD}$. D^* is similarly computed. Again the complexity of this computation is $O(|V^*|)$.

Criterion C: S^* is a vertex which minimizes the angle S^*SD. Again the complexity of this computation is $O(|V^*|)$.

The closeness of the paths planned by NAVIGATE-GLOBAL to the globally optimal path depends on the degree to which the LVG is built. The paths tend to be globally optimal as the LVG converges to the VG. We shall now prove that the LVG indeed converges to VG after sufficient number of invocations of NAVIGATE-LOCAL.

6. COMPLETE LEARNING

Learning is an integral part of NAVIGATE-LOCAL, primarily because the robot is initially placed in a completely unexplored obstacle terrain, and the LVG is incrementally constructed as the robot navigates. The central goal of the learning is to eventually construct the VG of the entire obstacle terrain. Once the VG is completely constructed, the globally optimal path from S to D can be computed **before** the robot sets into motion as in [15]. Furthermore, the availability

of the complete VG obviates the further usage of sensors. Hence, the focus of our naviagation scheme is to continually augment the LVG with the information extracted from sensor readings, with the aim of ultimately obtaining the complete VG. In this section we prove that the learning incorporated in our technique is *complete*, i.e., the LVG ultimately converges to VG with probability one, if the source and destination points are randomly selected in the free space.

THEOREM 3:

If no point in the free space has a zero probability measure of being a source or destination point or a point on a path of traversal, then the LVG converges to the VG with a probability one.

PROOF: As per the procedure UPDATE-VGRAPH, when a new vertex is included in V^*, all the edges corresponding to the visible nodes of the present LVG are added to E^*. Hence, it suffices to prove that every vertex of the VG is eventually added to the LVG. Equivalently, it is sufficient to prove that every edge of the obstacle is eventually explored by the robot.

Let p_i be the probability that an edge e_i is explored during any traversal. Since every point in the compact free space has a non-zero probability measure of being a source point or destination point or intermediate point, we have $p_i > 0$. Then, the probability that e_i is *not* encountered after k successive independent and randomly chosen paths is $(1-p_i)^k$. Clearly, this tends to zero as k tends to infinity. Hence, the theorem. □

We conclude this section with an interesting result that for the complete convergence of the LVG to the VG, the number of sensing operations involved in the procedure UPDATE-VGRAPH is quadratic in the total number of vertices of the obstacles.

THEOREM 4:

The number of sensor operations performed within the procedure UPDATE-VGRAPH to learn the complete VG is $O(N^2)$, where N is the total number of vertices of the obstacles.

PROOF: A explained in the lines 8, 12 and 14 of procedure NAVIGATE-LOCAL, no sensing operations are carried out when the robot encounters an already visited vertex. The sensor operations are performed only when the robot encounters a new vertex. Suppose the LVG presently has $i-1$ vertices, $\{v_1, v_2, \ldots, v_{i-1}\}$, when a new vertex v_i is encountered. At this time, the robot beams its sensors in the direction of $v_j \in \{v_1, v_2, \ldots, v_{i-1}\}$, to determine if v_j

is visible from v_i. Hence, the number of sensor operations carried out when the i th vertex is added to the LVG is $i-1$. Hence, the total number of sensor operations carried out in the procedure UPDATE-VGRAPH is given by

$$\sum_{i=1}^{N} i-1 = N(N-1)/2 = O(N^2)$$

Hence, the theorem. □

In the next section, we present a practical example for the technique described in this paper.

7. EXPERIMENTAL RESULTS

In this section, we describe experimental results obtained using a rectangular obstacle terrain shown in Fig. 9. Initially the terrain is unexplored and the LVG is empty. A sequence of five paths is undertaken in succession by the robot. In other words, the robot first to 2 from 1, then to 3 from 2, etc. till it reaches 6. Fig. 10 to Fig. 14 illustrate the various paths traversed and the corresponding LVG s.

Initially, during the motion from 1 to 2, the robot learnt four edges of the VG shown in Fig. 10(b). In the next traversal, seven more edges of the VG are learnt. A curve showing the number of edges learnt as a function of the number of traversals is given in Fig. 16. It is to be noted that as many as 31 out of total 39 edges of VG are learnt in five traversals.

Suppose that at this point the global navigation strategy is invoked to navigate to 7 from 6. The S^* and D^* obtained by using criterion (A) of section 5 are shown in Fig. 15. The robot navigates locally from S to S^* then along the LVG from S^* to D^*, and finally locally from D^* to D. Note that the path from S^* to D^* does not involve any sensor operations, but, only quadratic time computation on the LVG to find the shortest path.

8. CONCLUSIONS

The terrain model acquisition and path planning problems are very important aspects of an autonomous robot navigating in an unexplored terrain. In the literature, this problem has not been subjected to a rigorous mathematical treatment as far as the model aquisition is concerned.

In this paper, we propose a technique that enables an autonomous robot to navigate in a totally unexplored terrain. The robot builds the terrain model as it navigates, and stores the processed sensor information in terms of a learned

visibility graph. The proposed technique is proven to obtain a path if one exists. Furthermore, the terrain is guaranteed to become *completely learnt*, when the complete visibility graph of the entire obstacle terrain is built. After this stage the robot traverses along the optimal paths, and no longer needs the sensor equipment. The significance of this technique is the characterization of both the path planning and learning in a precise mathematical framework. The convergence of the path planning and the learning processes is proven.

REFERENCES

[1] AHO, A., J. HOPCROFT, and J. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass., 1974.

[2] BROOKS, R. A., Solving the Find-path Problem by Good representation of Free-space. **IEEE Trans. Systems, Man and Cybernetics**, Vol. SMC-13, No. 3, March/April 1983.

[3] BROOKS, R. A., and T. LOZANO-PEREZ, A Subdivision Algorithm in Configuration Space for Path with Rotation. **IEEE Trans. Systems, Man and Cybernetics**, Vol. SMC-15, No. 2, March/April 1985, pp. 224-233.

[4] CHATILA, R., Path Planning and Environment Learning in a Mobile Robot System, *Proc. European Conf. Artificial Intelligence*, Torsey, France, 1982.

[5] CHATTERGY, R., Some Heuristics for the Navigation of a Robot. **The Int. J. Robotics Research**, Vol.4, No. 1, Spring 1985, pp. 59-66.

[6] CROWLEY, J. L., Navigation of an Intelligent Mobile Robot, **IEEE J. Robotics Reseach**, Vol. RA-1, No. 2, March 1985, pp.31-41.

[7] DEO, N., *Graph Theory with Applications to Engineering and Computer Science*, Prentice-Hall, New York.

[8] GIRALT, G., R. SOBEK and R. CHATILA, A Multilevel Planning and Navigation System for a Mobile Robot, *Proc. 6th Int. Joint Conf. Artificial Intelligence*, Aug 20-23, 1979, Tokyo, pp. 335-338.

[9] GOUZENES, L., Strategies for Solving Collision-free Trajectories Problems for Mobile and Manipulator Robots, **The Int. J. Robotics Research**, Vol. 3, No. 4, Winter 1984, pp. 51-65.

[10] IYENGAR, S.S., C. C. JORGENSEN, S. V. N. RAO, and C. R. WEISBIN, Robot Navigation Algorithms Using Learned Spatial Graphs, ORNL Technical Report TM-9782, Oak Ridge National Laboratory, Oak Ridge, August 1985, to appear in **Robotica**, Jan. 1986.

[11] IYENGAR, S.S., C. C. JORGENSEN, S. V. N. RAO and C. R. WEISBIN, Learned Navigation Paths for a Robot in Unexplored Terrain, *Proc. 2nd Conf. Artificial Intelligence Applications and Engineering of Knowledge Based Systems*, Miami Beach, Florida, December 11-13, 1985.

[12] LAUMOND, J., Model Structuring and Concept Recognition: Two Aspects of Learning for a Mobile Robot, *Proc. 8th Conf. Artificial Intelligence*, August 8-12, 1983, Karlsruhe, West Germany, p. 839.

[13] LEE D.T., and F. P. PREPARATA, Computational Geometry - A Survey, **IEEE Trans. Computers**, Vol. C-33, No. 12, December 1984, pp. 1072-1101.

[14] LOZANO-PEREZ, T., Spatial Planning: A Configuration Space Approach, **IEEE Trans. Computers**, Vol. C-32, February 1983, pp. 108-120.

[15] LOZANO-PEREZ, T., and M. A. WESLEY, An Algorithm for Planning Collision-free Paths Among Polyhedral Obstacles, **Commun. ACM**, Vol. 22, No. 10, October 1979, pp. 560-570.

[16] MEHLHORN, K., *Data Structures and Algorithms 3: Multidimensional Searching and Computational Geometry*, EATCS Monographs on Theoretical Computer Science, Springer-Verlag, Berlin, 1984.

[17] MORAVEC, H. P., The CMU Rover, *Proc. Nat. Conf. Artificial Intelligence*, August 1982, pp. 377-380.

[18] NILSSON, N.J., Mobile Automation: An Application of Artificial Intelligence Techniques, *Proc. 1st Int. Joint Conf. Artificial Intelligence*, May 1969, pp. 509-520.

[19] OOMMEN J.B. and I. REICHSTEIN, On The Problem of transalting an Elliptic Object Through a Workspace of Elliptic Obstacles, Tech. Rep. No. SCS-TR-79, School of Computer Science, Carleton University, Canada, July 1985.

[20] RAO, S.V.N., S.S. IYENGAR, C.C. JORGENSEN, and C.R. WEISBIN, Concurrent Algorithms for Autnomous Robot Navigation in an Unexplored Terrain, Tech. Rep. #85-048, Dept. Computer Science, Louisiana State University, Sept.1986 (accepted for *IEEE Robotics and Automation Conference*, April 1986).

[21] THOMPSON, A. M., The Navigation System of the JPL Robot, *Proc. 5th Int. Joint Conf. Artificial Intelligence*, August 22-25, 1977, Cambridge, Mass., pp. 749-757.

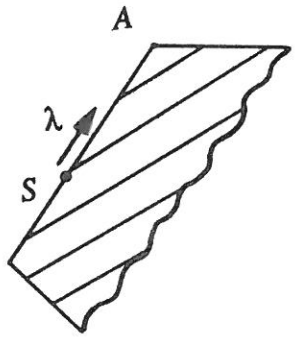
[22] TURCHEN M. P., and A. K. C. WONG, Low Level Learning for a Mobile Robot: Environmental Model Acquisition, to be published in *Proc. 2nd Int. Conf. Artificial Intelligence and Its Applications*, December 1985.

[23] UDUPA, S. M., Collision Detection and Avoidance in Computer Controlled Manipulators, *Proc. 5th Int. Conf. Artificial Intelligence*, Massachusetts Institute of Technology, Cambridge, Mass., August 1977, pp. 737-748.

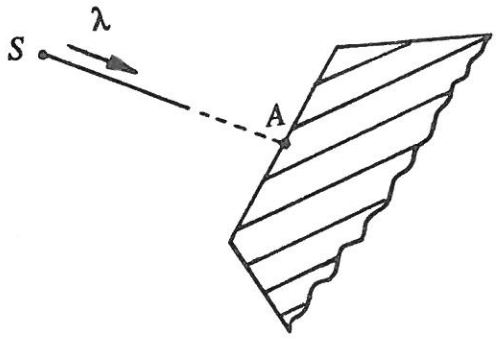
[24] WALLACE, R. et al, First Results in Robot Road Following, *Proc. 9th Int. Conf. Artificial Intelligence*, August 18-23, 1985, Los Angeles, CA, pp. 1089-1095.

[25] WEISBIN, C. R., J. BARHEN, G. DeSAUSSURE, W.R. HAMEL, C. JORGENSEN, E.M. OBLOW, and R. E. RICKS, Machine Intelligence for Robotics Applications, *Proc. 1985 Conf. Intelligent systems ans Machines*, April 22-24, 1985.

[26] WHITESIDES, S., Computational Geometry and Motion Planning, *Computational Geometry*, Ed. by G. Toussaint, North Holland, 1985.



(a) Motion along the edge of an obstacle.



(b) Motion till an obstacle is encountered.

Fig.1. Value returned by the operation MOVE(S ,A λ), when A is not specified.

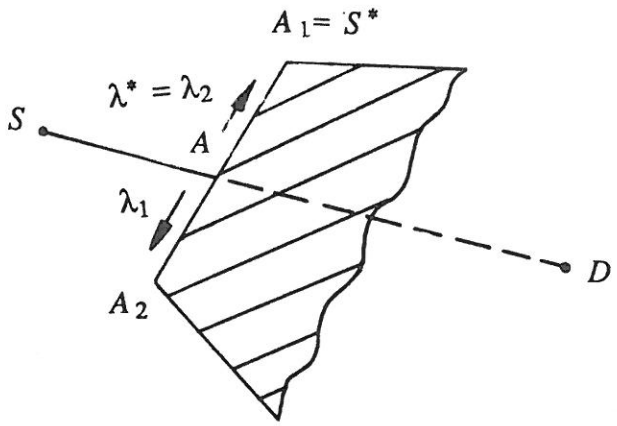
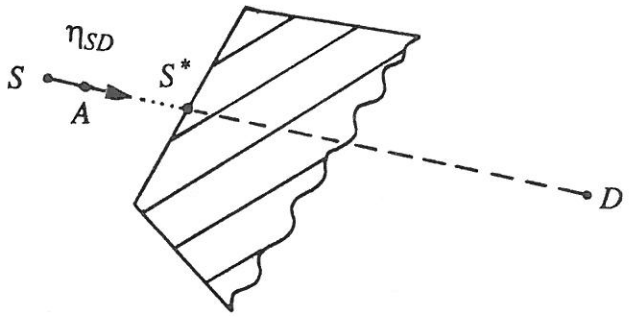
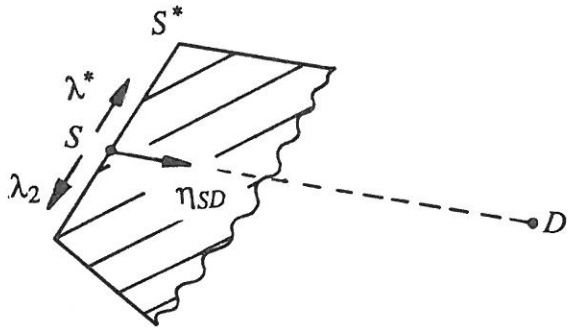


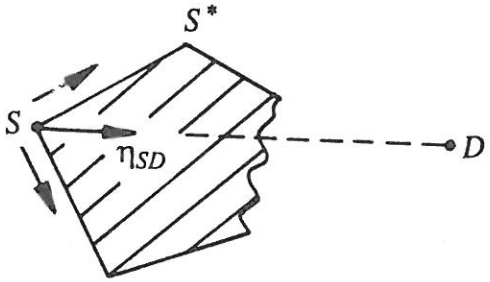
Fig.2. The robot reached a point on the obstacle.



(a) S is not on the edge of an obstacle

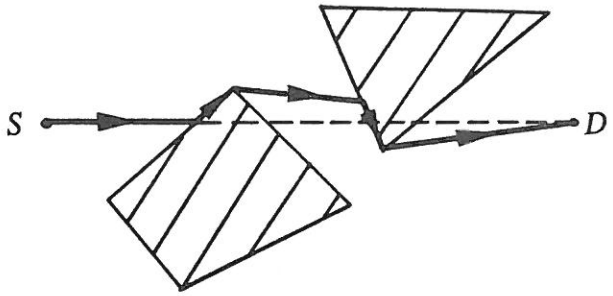


(b) S is on the edge of an obstacle but not at a vertex.

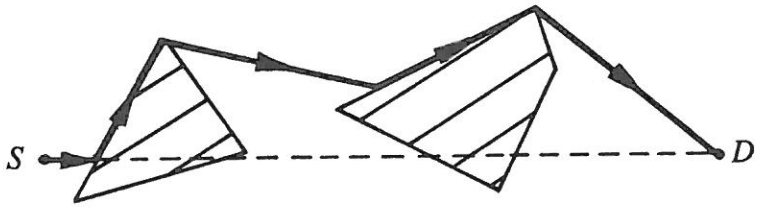


(c) S is at the vertex of an obstacle.

Fig.3. The robot at S is obstructed by an obstacle.



(a) Solution is both globally and locally optimal.



(b) Soultion is only loally optimal.

Fig.4. Local navigation strategy need not yeild a globally optimal solution. Dark lines with arrows indicate the path according to the local navigation strategy.

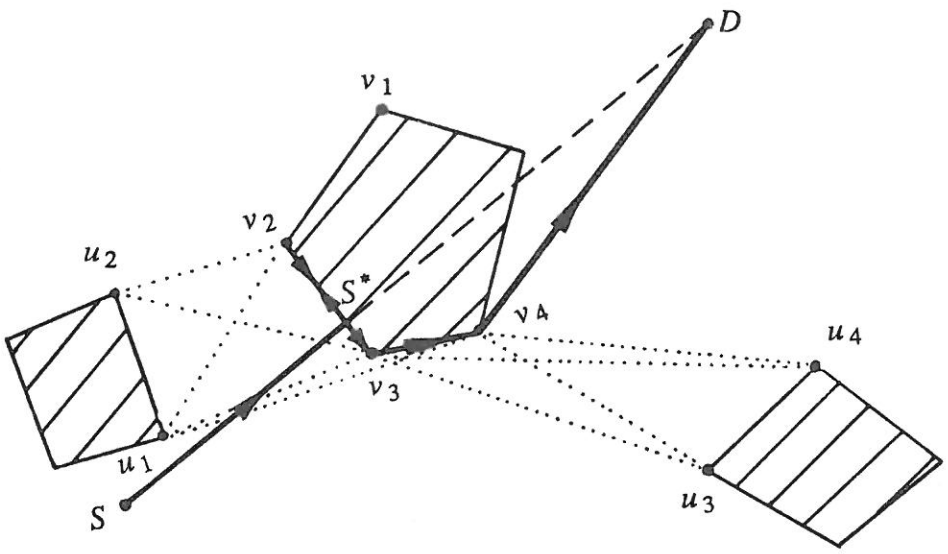
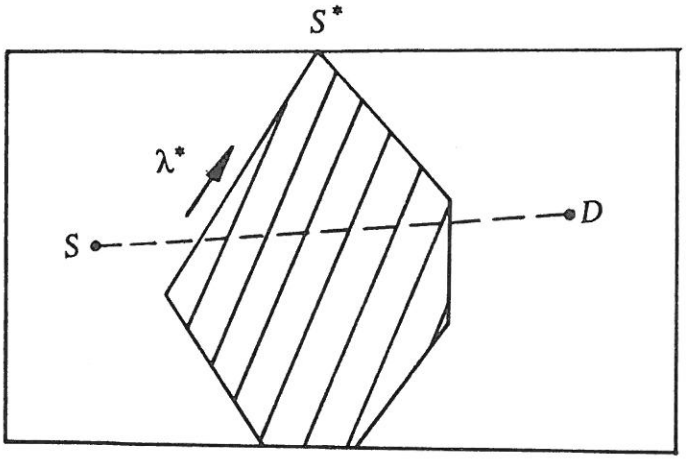
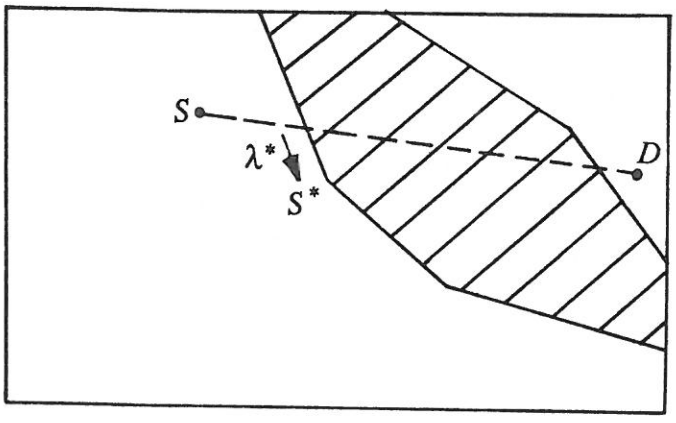


Fig.5. Computation of intervisibility of the vertices $\{v_2, v_3, v_4\}$ as a result of local navigation. Dotted lines indicate the visibility between the vertices.



(a) Case 1.



(b) Case 2.

Fig.6. No path from S to D.

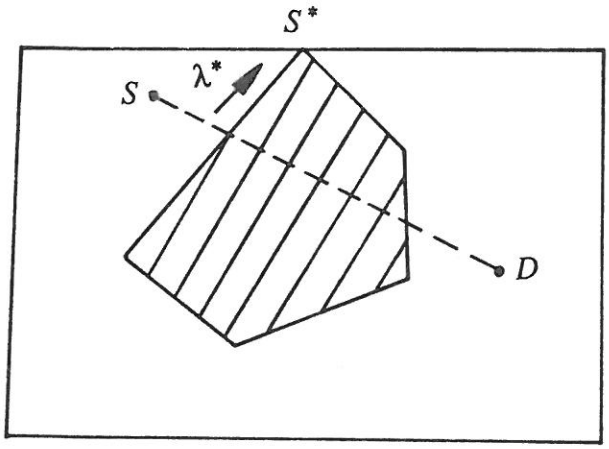


Fig.7. Dead-corner S^*, formed by the obstacle and the terrain boundary.

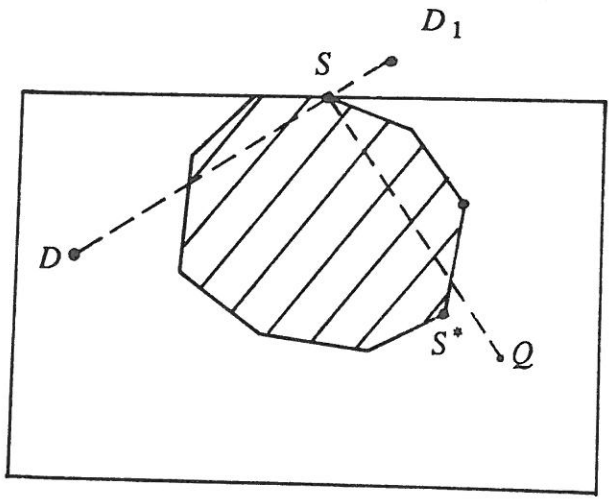


Fig.8. Proof of convergence of the procedure BACKTRACK.

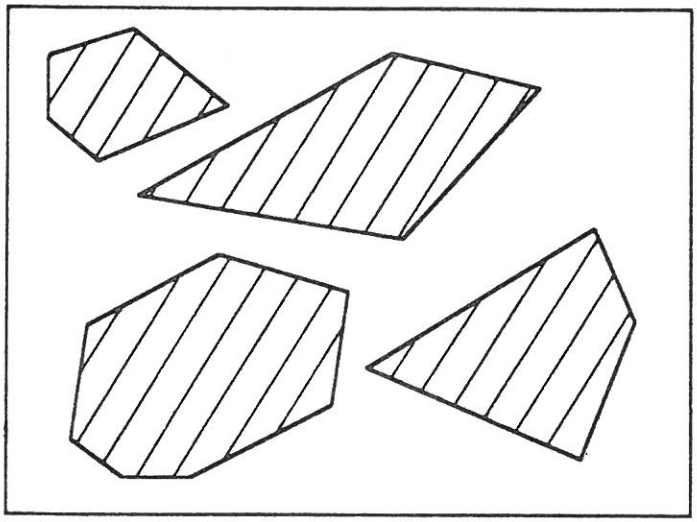
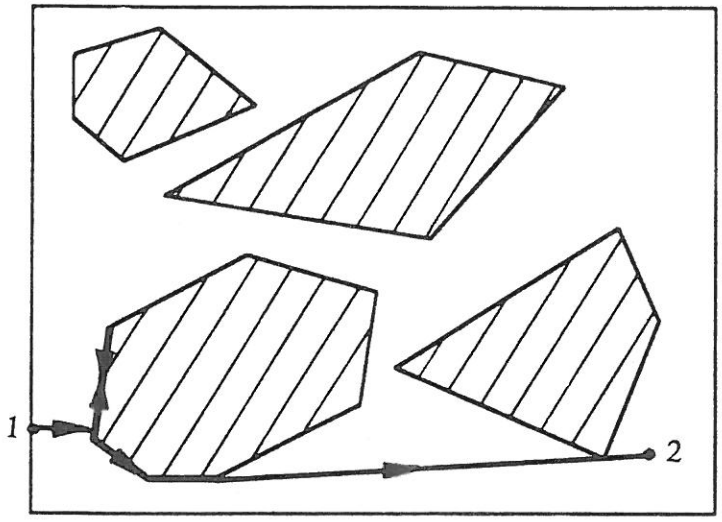
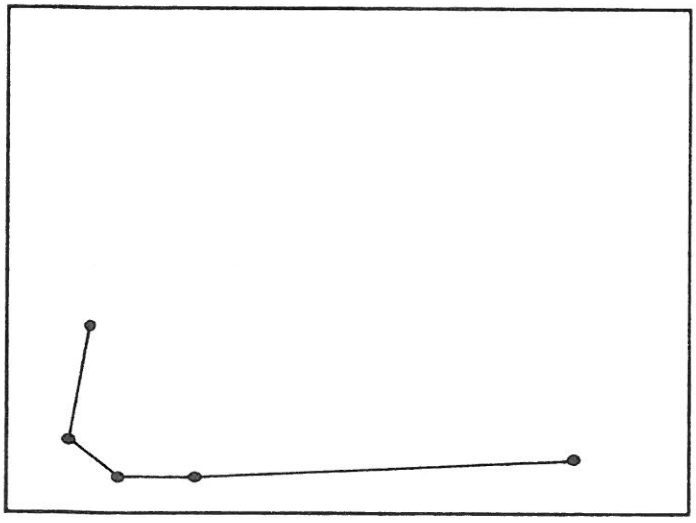


Fig. 9 Unexplored obstacle terrain.

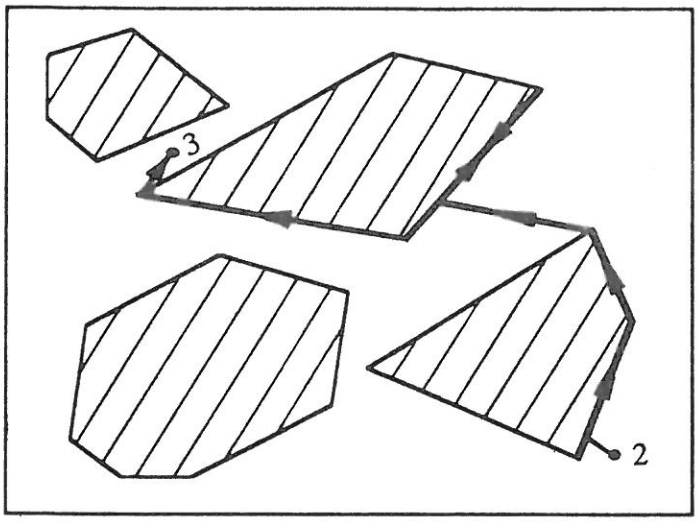


(a) obstacle terrain.

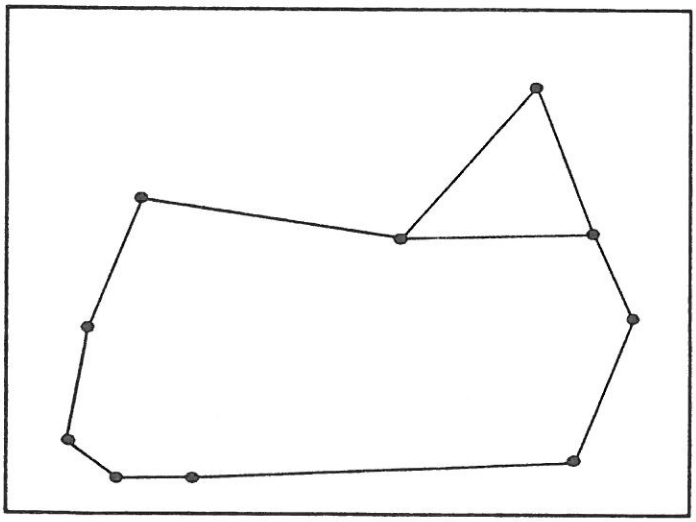


(b) Present *LVG*, a portion of *VG*.

Fig. 10. NAVIGATE-LOCAL from 1 to 2.

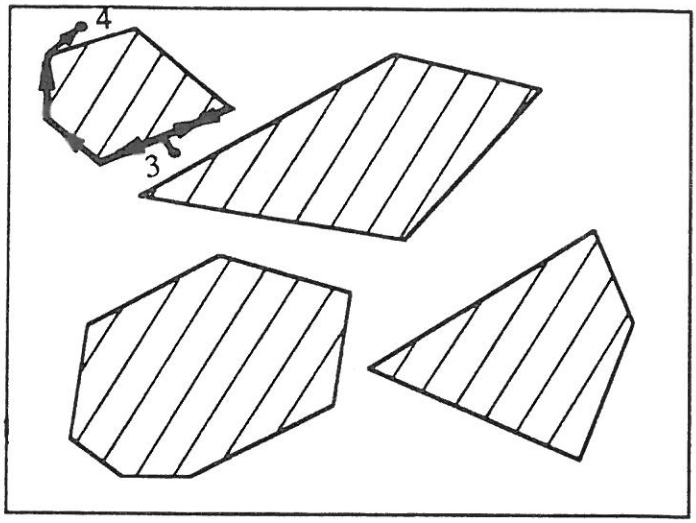


(a) Obstacle terrain.

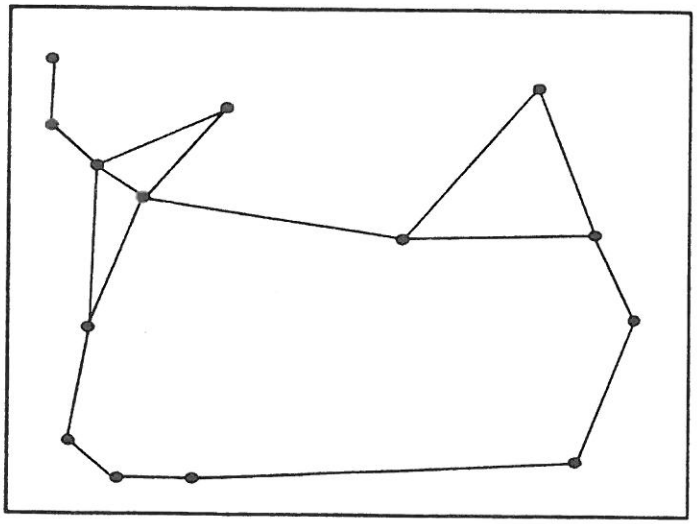


(b) Present *LVG*.

Fig. 11. NAVIGATE-LOCAL from 2 to 3.

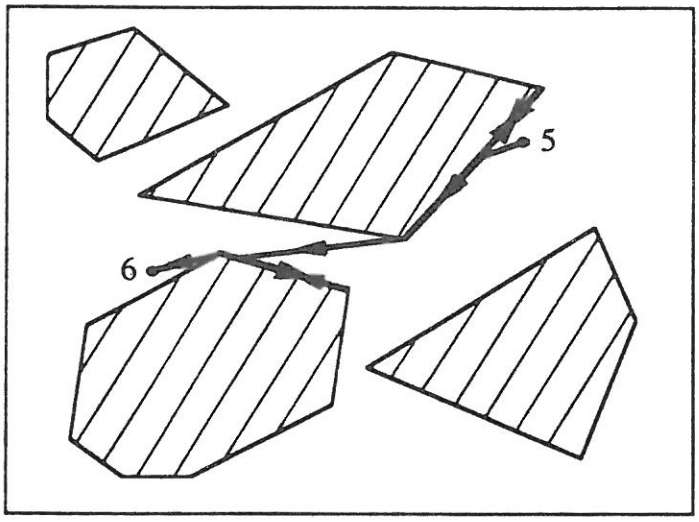


(a) Obstacle terrain.

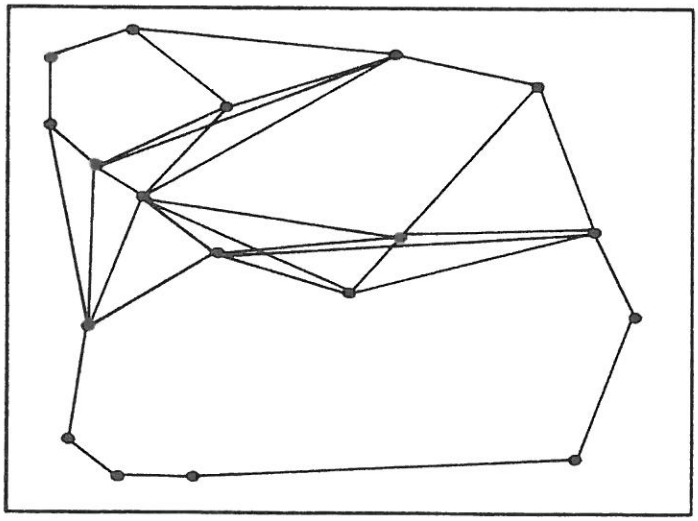


(b) Present *LVG*.

Fig.12. NAIGATE-LOCAL from 3 to 4.

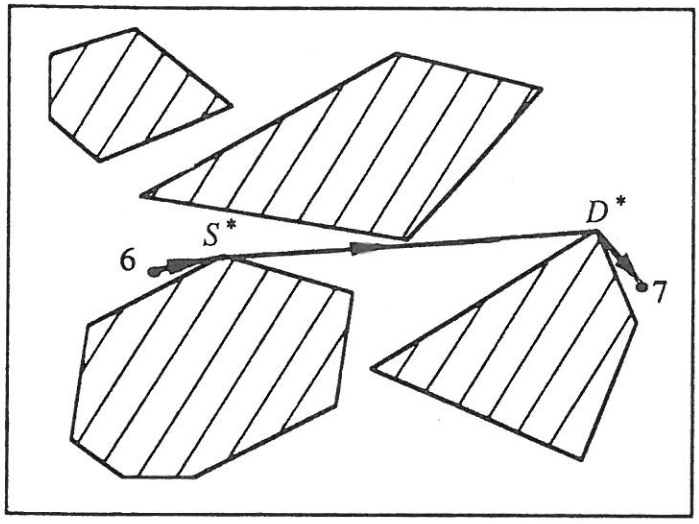


(a) Obstacle terrain.

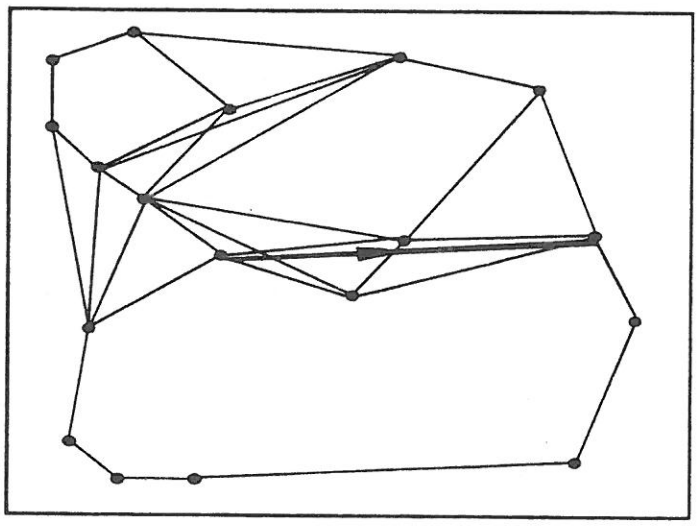


(b) Present *LVG*.

Fig.14. NAIGATE-LOCAL from 5 to 6.



(a) Obstacle terrain



(b) Present *LVG*.

Fig. 15. Navigate from 6 to 7. Note that the path actually computed uses th visibility graph, and is the shortest path on *LVG*.

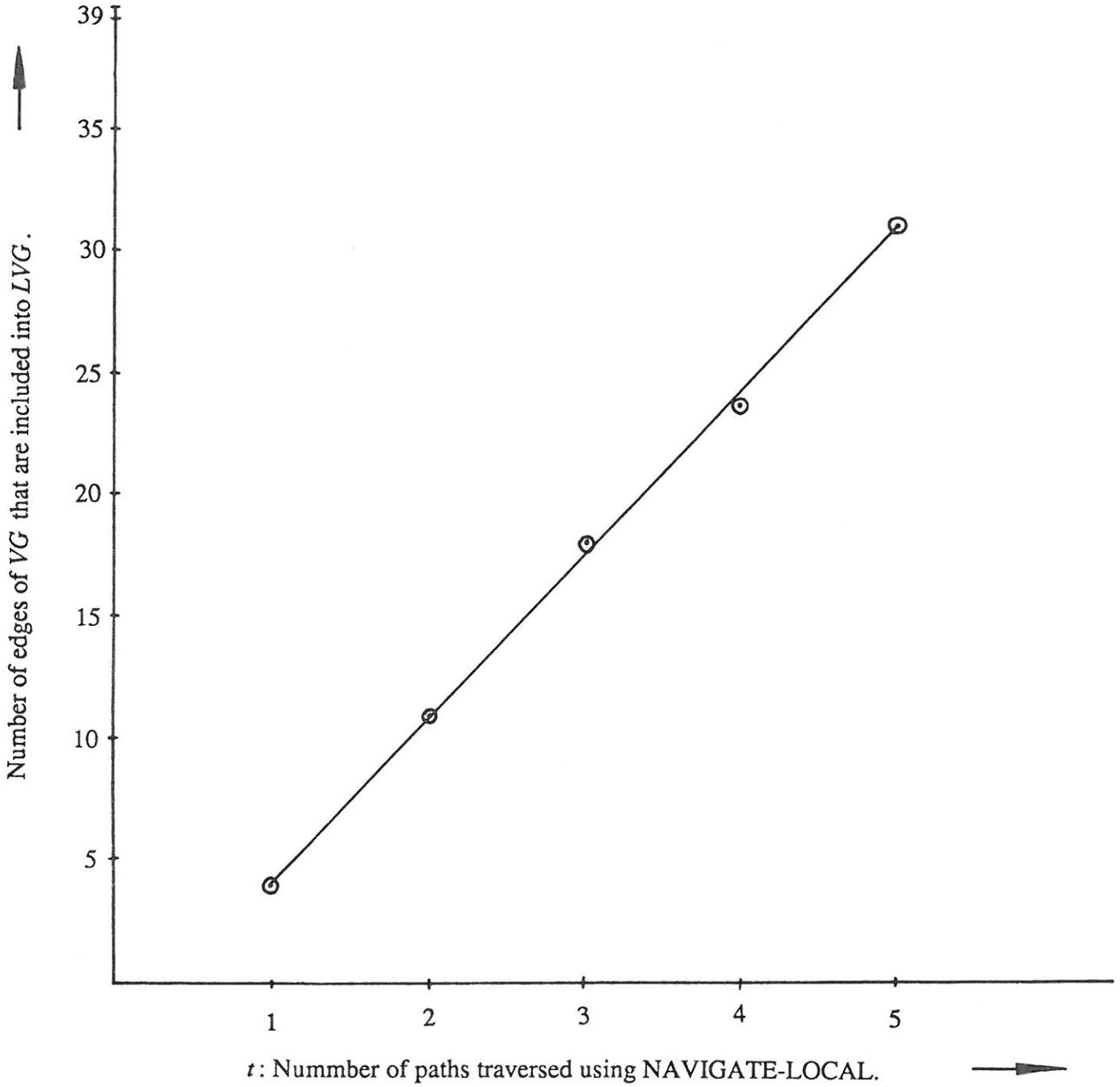


Fig. 16. Graph showing the number of edges in the *LVG* as a function of the number of paths traversed using NAVIGATE-LOCAL. Out of a total of 39, the robot learns 31 edges in five traversals.