Communications

Robot Navigation in Unknown Terrains Using Learned Visibility Graphs. Part I: The Disjoint Convex Obstacle Case

B. JOHN OOMMEN, MEMBER, IEEE, S. SITHARAMA IYENGAR, NAGESWARA S. V. RAO, AND R. L. KASHYAP, FELLOW, IEEE

Abstract—**The problem of navigating an autonomous mobile robot through unexplored terrain of obstacles is discussed. The case when the obstacles are "known" has been extensively studied in literature. *Completely* unexplored obstacle terrain is considered. In this case, the process of navigation involves both learning the information about the obstacle terrain and path planning. An algorithm is presented 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 suboptimal. 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 learned 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 obviates the further usage of sensors.**

I. 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 maintenance, inspection, etc., in industries. In hazardous environments, such as nuclear power plants, underwater, etc., robots are employed to carry out humanlike 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 extensively studied and solved by many researchers—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. Whitesides [26] is an excellent reference for various strategies used to solve the find-path problem. 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 rigorous 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—Brooks [2], Chatila [4], Chattergy [5], Crowley [6], Giralt *et al.* [8], Iyengar *et al.* [10], [11], Laumond [12], Lumelsky and Stepanov [16], 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 the find-path problem.

In this correspondence 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 learned 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 important 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 benefited by the available computational geometry and other related algorithms found in [1], [7], [13], [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.

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 here is proven for convergence in terms of planning paths and also in acquiring the entire terrain model through learning. As an endeavor 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, we shall not crystallize our terrain in terms of a Voronoi diagram. Rather, we shall compute and maintain 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 a slightly restricted workspace. In the process of local navigation, the robot manipulates the LVG. It is shown that the LVG

Manuscript received March 19, 1986. This work was supported in part by the National Sciences and Engineering Council of Canada, and by the National Science Foundation under Grant CDR-85-00022. This correspondence was presented in part at the 1986 National Conference on Artificial Intelligence, Philadelphia, PA.

B. J. Oommen is with the School of Computer Science, Carleton University, Ottawa, ON K1S 5B6, Canada.

S. S. Iyengar and N. S. V. Rao are with the Department of Computer Science, Louisiana State University, Baton Rouge, LA 70803.

R. L. Kashyap is with the Department of Electrical Engineering, Purdue University, West Lafayette, IN 47907.

IEEE Log Number 8716968.

ultimately converges to the actual visibility graph (VG) of the obstacle terrain with probability one. The use of the LVG in global navigation and its acquisition during the local navigation phase is the essential difference between our technique and the techniques used by other researchers [2], [4]–[6], [12], [16].

The organization of this correspondence is as follows: Section II introduces the definitions and notations used subsequently. The local navigation technique that incorporates learning and path planning is presented in Section III. The convergence of the proposed algorithm is proven. In Section IV, 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 V, 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 VI. The execution of the navigation algorithms on a sample obstacle terrain is presented in Section VII.

II. 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 $W = \{w_1, w_2, \cdots, w_k\}$ be the set of obstacles in the terrain R, where w_i is a convex polygonal obstacle. Furthermore, the obstacles' polygons are mutually nonintersecting and nontouching. 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*. The VG is a pair (V, E) where the following hold.

1) V is the set of vertices of the obstacles.

2) E is the set of edges of the graph. A line joining the vertices v_i and 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, if $L(v_i, v_j)$ is the set of points on the line joining v_i and v_j, then $(v_i, v_j) \in E$ iff a) $L(v_i, v_j) \in Z$ or b) $L(v_i, v_j) \cap Z = \phi$.

Visibility graphs have been extensively studied in the computational 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 learned during the initial stages of the navigation process. The partially learned 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* 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 correspondence 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. These sensors are assumed to be error-free.

The interior of any polygon ξ is denoted by INT (ξ). The straight line from the point P to the point Q is denoted by $\overrightarrow{PQ}$. Further, η_{PQ} denotes the unit vector along the straight line $\overrightarrow{PQ}$. We assume throughout that the robot is operating in the plane. Thus when we use the word "terrain," we use it in a more restricted sense than it is customarily used in the literature. Undoubtedly, navigating in a three-dimensional (3-D) terrain is a far more difficult problem, and we do not claim that the technique we propose is applicable to it. Indeed, even the concept of visibility graphs is not all too meaningful in the latter problem because whereas paths along the edges of polyhedra may not exist, paths along faces of the polyhedra may [26].

III. 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 arbitrary 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 is the direction of $\overrightarrow{SA}$. 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 until 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.

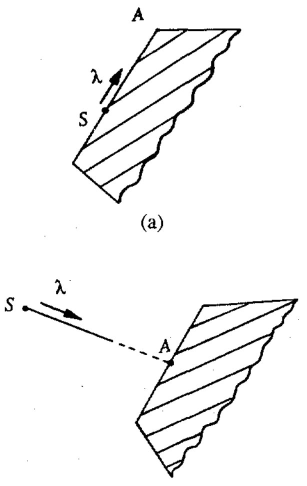
In the remainder of this section we describe the local navigation algorithm. For the treatment in this section we assume that the obstacles do not touch or intersect the boundaries of the terrain R. In other words, the obstacles are properly contained in the terrain R. This is formally represented as

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

As a consequence of this assumption, a path always exists 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 $\overrightarrow{SD}$ until 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 $\overrightarrow{AA_1}$ or $\overrightarrow{AA_2}$ as shown in Fig. 2. We define a local optimization



(b)

Fig. 1. Value returned by operation MOVE(S,A,λ), when A is not specified. (a) Motion along edge of obstacle. (b) Motion till obstacle is encountered.

criterion function J as follows:

$$J = \eta_{SD} \cdot \lambda \tag{2}$$

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

Let λ_1 and λ_2 be the unit vectors along $\overrightarrow{AA_1}$ and $\overrightarrow{AA_2}$, respectively. Let $\lambda^* \in \{\lambda_1, \lambda_2\}$ maximize the function J given in (2). The robot then undertakes an exploratory traversal along the direction $-\lambda^*$ until 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^* until 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 $(\phi, \phi)$.
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 along 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 as follows:

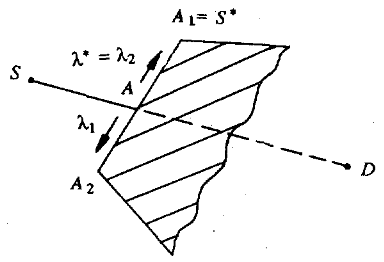


Fig. 2. Robot reached point on obstacle.

```
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($S$,$D$,$\eta_{SD}$)
3.  else
4.    if ($S$ is on an obstacle and the obstacle obstructs its view) then
5.      compute $\{\lambda_1,\lambda_2\}$, the two possible directions of motion;
6.      $\lambda^*$ = direction maximizing $\lambda_i \cdot \eta_{SD}$;
7.      if ($S$ is a vertex) then
8.        if ($S \neq V^*$) then UPDATE-VGRAPH($S$);
9.        MOVE($S$,$S^*$,$\lambda^*$);
10.     else
11.       MOVE($S$,$S_1$,$-\lambda^*$); {make exploratory trip to $S_1$}
12.       if ($S_1 \neq V^*$) then UPDATE-VGRAPH($S_1$);
13.       MOVE($S_1$,$S^*$,$\lambda^*$); {retrace steps to $S^*$}
14.       if($S^* \neq V^*$) then UPDATE-VGRAPH($S^*$);
      endif;
15.     NAVIGATE-LOCAL($S^*$,D);
16.   else {move to next obstacle}
17.     MOVE($S$,$S^*$,$\eta_{SD}$); {move to next obstacle along $\eta_{SD}$}
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: For a noninterlocking workspace, the procedure NAVIGATE-LOCAL always finds a path from S to D in finite time.[1]

Proof: There is always a path from S to D as per the assumption in (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:

$$|SD| > \overrightarrow{S^*D} \cdot \eta_{SD}. \tag{3}$$

Case IIa: 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.

[1] Please see the Appendix for the reason why the workspace should be restricted for the current version of NAVIGATE-LOCAL. The details of a workspace being noninterlocking are included in the Appendix.

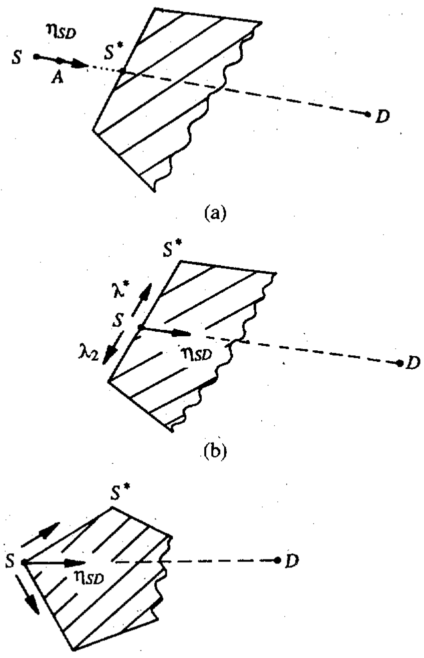


Fig. 3. Robot at S is obstructed by obstacle. (a) S is not on edge of obstacle. (b) S is on edge of obstacle but not at vertex. (c) S is at vertex of obstacle.

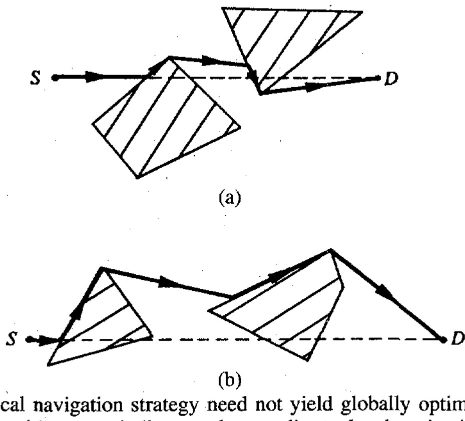


Fig. 4. Local navigation strategy need not yield globally optimal solution. Dark lines with arrows indicate path according to local navigation strategy. (a) Solution is both globally and locally optimal. (b) Solution is only locally optimal.

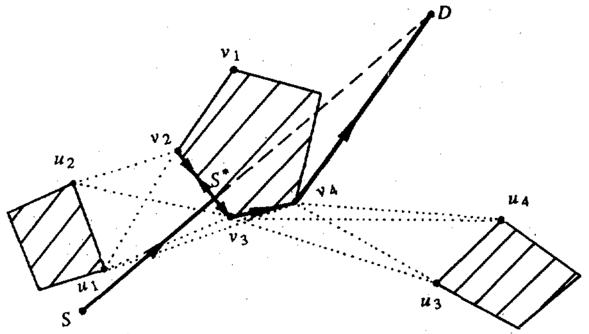


Fig. 5. Computation of intervisibility of vertices $\{v_2, v_3, v_4\}$ as result of local navigation. Dotted lines indicate visibility between vertices.

In this case, the motion is along η_{SD} and every point A along this vector satisfies the relation $|\overrightarrow{AD}| = \overrightarrow{AD}\cdot\eta_{SD}$. Thus the motion along η_{SD} to A gives the following equality: $|\overrightarrow{SD}| = |\overrightarrow{SA}| + \overrightarrow{AD}\cdot\eta_{SD}$, whence

$$|\overrightarrow{SD}| > \overrightarrow{AD}\cdot\eta_{SD}. \tag{4}$$

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

Case IIb: The point D is not visible from S, and S lies on the edge of the obstructing obstacle.

Case 1): S does not correspond to a vertex of the LVG as shown in Fig. 3(b). If $\lambda_i\cdot\eta_{SD} = 0$, then the edge is orthogonal to η_{SD}. In this case, either direction does not decrease the projected distance. Note that this situation can occur at most once for an obstacle encountered during the navigation and hence cannot persist. However, after the robot completes the MOVE corresponding to this step, the robot is located at a vertex. Since this is covered in case 2, we shall only consider the case when $\lambda_i\cdot\eta_{SD} \neq 0$. Let λ^* be the direction in which $\lambda_i\cdot\eta_{SD} > 0$. The situation is shown in Fig. 3(b). Hence the included angle S^*SD is less than $\Pi/2$, and $|\overrightarrow{SD}| - |\overrightarrow{S^*D}\cdot\eta_{SD}| = |\overrightarrow{SS^*}| \cos(S^*SD) > 0$. Hence the execution of MOVE(S, S^*, λ^*) ensures the inequality given in (3).

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

Case 2): 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 $\lambda_i\cdot\eta_{SD}$. Because the obstacle is convex, the angle between the edges at S is less than Π. Thus the angle S^*SD is less than $\Pi/2$. Using the arguments of case 1, we conclude that the execution of the MOVE operation satisfies $|\overrightarrow{SD}| - |\overrightarrow{S^*D}\cdot\eta_{SD}| > 0$, and hence the theorem.

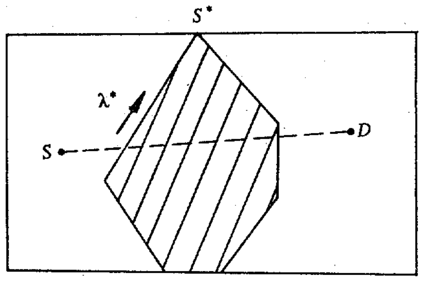
Observe that if we had only *one* polygonal obstacle, we could have gone around the obstacle in a systematic way, i.e., either in a clockwise or an anticlockwise direction, until we reached a point from which D is visible. However, the problem becomes more difficult when more than one obstacle exists. 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 (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. The modification of NAVIGATE-LOCAL for interlocking workspaces is shown in the Appendix.

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 vertex is added to V^*. Fig. 5 shows the salient features of the approach. The vertices u_2, 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. However, we choose the path $SS^*v_2S^*v_3v_4D$—which is only suboptimal. The exploratory traversal to v_2 yields the visibility information about the vertices v_1, 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.

IV. Limitations of Local Navigation and a Solution

The procedure NAVIGATE-LOCAL introduced in the previous section always yields a path in a noninterlocking workspace 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 (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 such cases. Note that when the robot starts moving around the obstacle, its way is blocked in both possible directions.



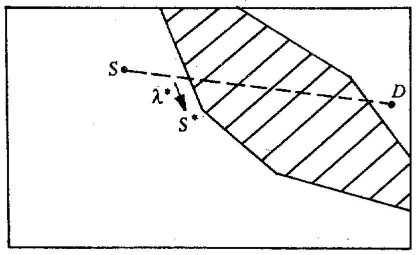


Fig. 6. No path from S to D. (a) Case 1. (b) Case 2.

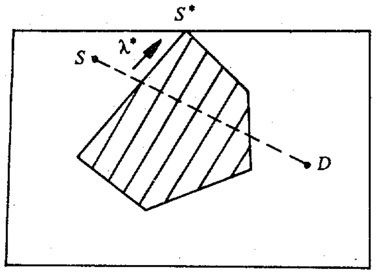


Fig. 7. Dead corner S^* formed by obstacle and terrain boundary.

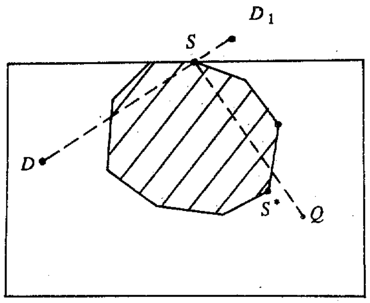


Fig. 8. Proof of convergence of procedure BACKTRACK.

b) The angle between the obstacle edge and the terrain boundary is less than $\Pi/2$. In this case we assume that a path exists 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 gets 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 performed 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. The following is the BACKTRACK algorithm:

```
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. The terrain is noninterlocking.
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.  $\lambda^*$ = only permitted direction of motion on the obstacle;
3.  while ($\overrightarrow{SD} \cdot \lambda^* < 0$) do
4.      MOVE($S_1, S^*, \lambda^*$);
5.      $S_1 = S^*$;
6.      $\lambda^*$ = 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 in a noninterlocking workspace, if one exists.

Proof: The crux of the theorem is to prove that the procedure BACKTRACK terminates in all cases. In other words, an edge exists 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 $\overrightarrow{SD}$. 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 cannot 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 used. The navigation is stopped if no path exists between S and D. The correctness of the proof of procedure NAVIGATE-LOCAL-WITH-BACKTRACK easily follows from the arguments 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.

V. Global Navigation

The procedures described in the preceding 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^* and D^* is along the optimal path computed using the LVG. Again, from D^* to D the local navigation is used. 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 as follows.

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

Criterion B: S^* is a vertex such that it is the closest to the line $\overrightarrow{SD}$. 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 a sufficient number of invocations of NAVIGATE-LOCAL.

VI. 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 navigation 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.

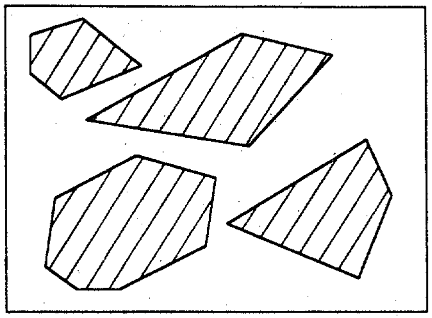


Fig. 9. Unexplored obstacle terrain.

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 nonzero 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 is proved.

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 $\{v_1, v_2, \cdots, v_{i-1}\}$ when a new vertex v_i is encountered. At this time, the robot beams its sensors in the direction of $v_j \in \{v_1, v_2, \cdots, v_{i-1}\}$ to determine if v_j is visible from v_i. Hence the number of sensor operations carried out when the ith vertex is added to the LVG is $i - 1$. Therefore, the total number of sensor operations carried out in the procedure UPDATE-VGRAPH is given by

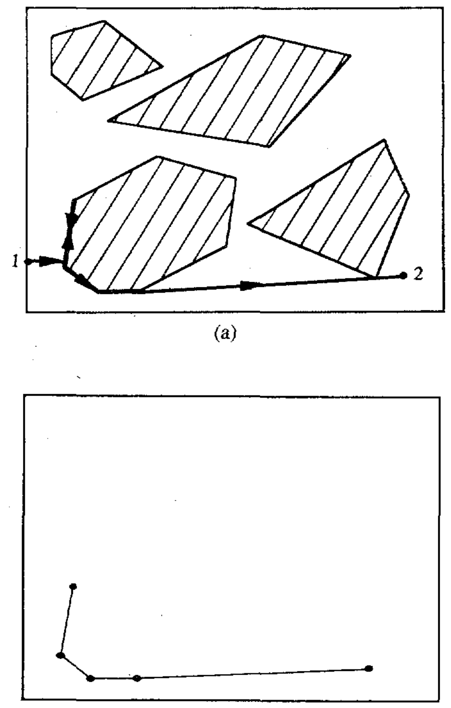
$$\sum_{i=1}^{N} i-1 = N(N-1)/2 = O(N^2),$$

hence the theorem.

The underlying premise of our work has been that we have assumed that the sensors used and the navigation technology used are error free. This, of course, is a serious limitation. The question of operating in an environment prone to errors (with these errors described either by a bound or by a probability distribution) is a problem that is far more complex. An initial (but noteworthy) move in this direction of solving the problem has been made by Brooks [2]. We are currently investigating the formalization of the convergence properties of a path-planning algorithm in the midst of uncertainties using the principle of adaptive learning. In the next section we present a practical example for the technique described in this paper.

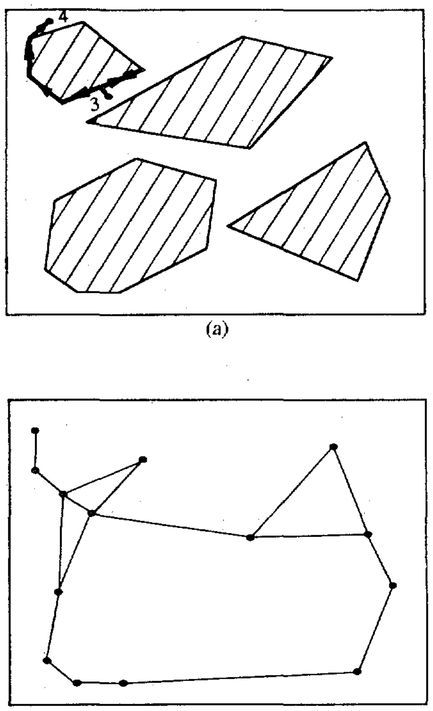
VII. An Illustrative Example

In this section we describe an illustrative example of our scheme for 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 moves first to 2 from 1, then to 3 from 2, etc., until it reaches 6. Figs.



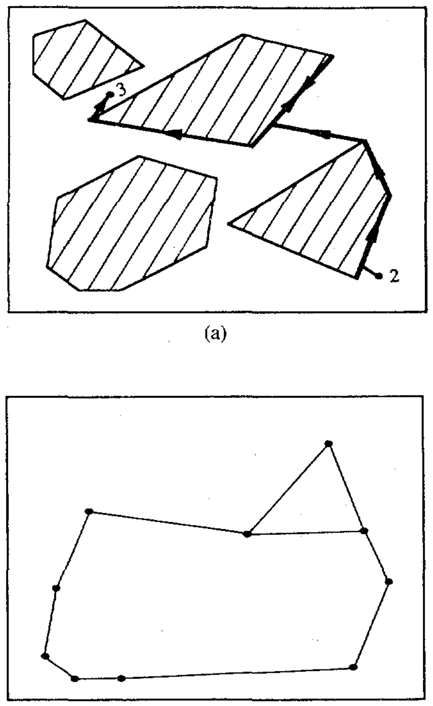
(b)

Fig. 10. NAVIGATE-LOCAL from 1 to 2. (a) Obstacle terrain. (b) Present LVG, portion of VG.



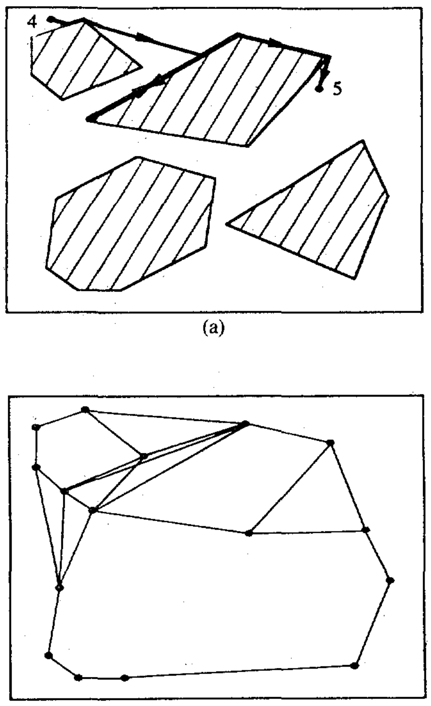
(b)

Fig. 12. NAVIGATE-LOCAL from 3 to 4. (a) Obstacle terrain. (b) Present LVG.



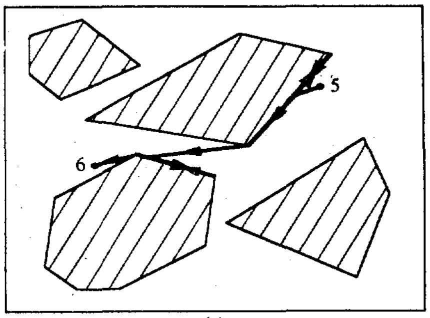
(b)

Fig. 11. NAVIGATE-LOCAL from 2 to 3. (a) Obstacle terrain. (b) Present LVG.

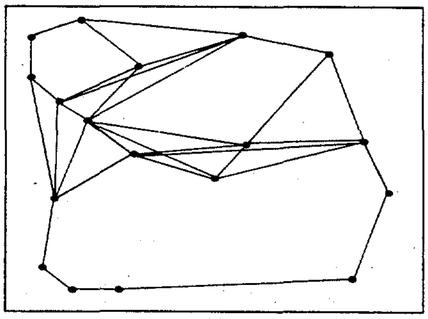


(b)

Fig. 13. NAVIGATE-LOCAL from 4 to 5. (a) Obstacle terrain. (b) Present LVG.

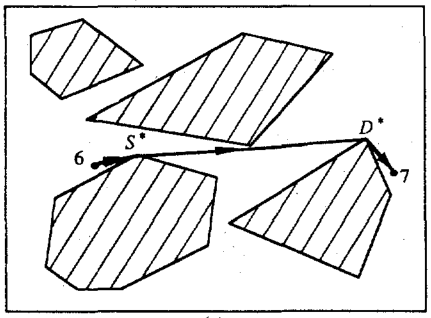


(a)

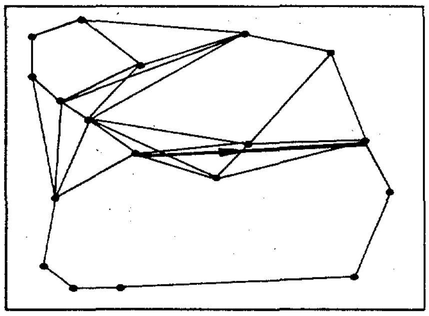


(b)

Fig. 14. NAVIGATE-LOCAL from 5 to 6. (a) Obstacle terrain. (b) Present LVG.



(a)



(b)

Fig. 15. Navigate from 6 to 7. Note that path actually computed uses visibility graph and is shortest path on LVG. (a) Obstacle terrain. (b) Present LVG.

10–14 illustrate the various paths traversed and the corresponding LVG's.

Initially, during the motion from 1 to 2, the robot learns four edges of the VG shown in Fig. 10(b). In the next traversal, seven more edges of the VG are learned. A curve showing the number of edges learned as a function of the number of traversals is given in Fig. 16. Note that as many as 31 out of a total of 39 edges of VG are learned 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 V 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. Actual simulation results obtained using random paths are presented elsewhere [27].

VIII. Conclusion

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 acquisition 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 learned* 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.

Appendix

After the paper was accepted for publication, just prior to the publication of the final manuscript, one of the reviewers noted that there was an error in the convergence proof of Theorem I. He did this by presenting a counter example which we will now present.

The robot is to navigate locally from S to D. Observe that in this case the robot can "cycle" indefinitely as shown in Fig. 17. We refer to a terrain which possesses such a cycling configuration of obstacles as a interlocking terrain.

We propose a rather minor modification of the algorithm NAVIGATE_LOCAL which considers this. Rather than the robot leave an obstacle at any arbitrary vertex (or edge), it is constrained to leave an obstacle on an edge intersected by the line SD and that *only* at the point where the edge intersects the line SD. Clearly, since the obstacles are nonintersecting and convex, there will be exactly two such eligible edges. Since the projected distance along the line SD is always minimized, the robot will leave the obstacle under consideration at the edge which is "closer" to D. For a simple example, Fig. 18 shows the edges traversed using the modification.

Observe that NAVIGATE_LOCAL is now slightly less optimal in terms of the number of traversals it requires. However, it must be noted that every extra traversal yields more information about the VG, and thus the size of the LVG will increase in this case. Thus the learning process will be catalyzed.

Acknowledgment

We would like to thank Prof. T. Lazano-Perez of the Massachusetts Institute of Technology for carefully reading the manuscript and for his helpful and critical comments. He also suggested various

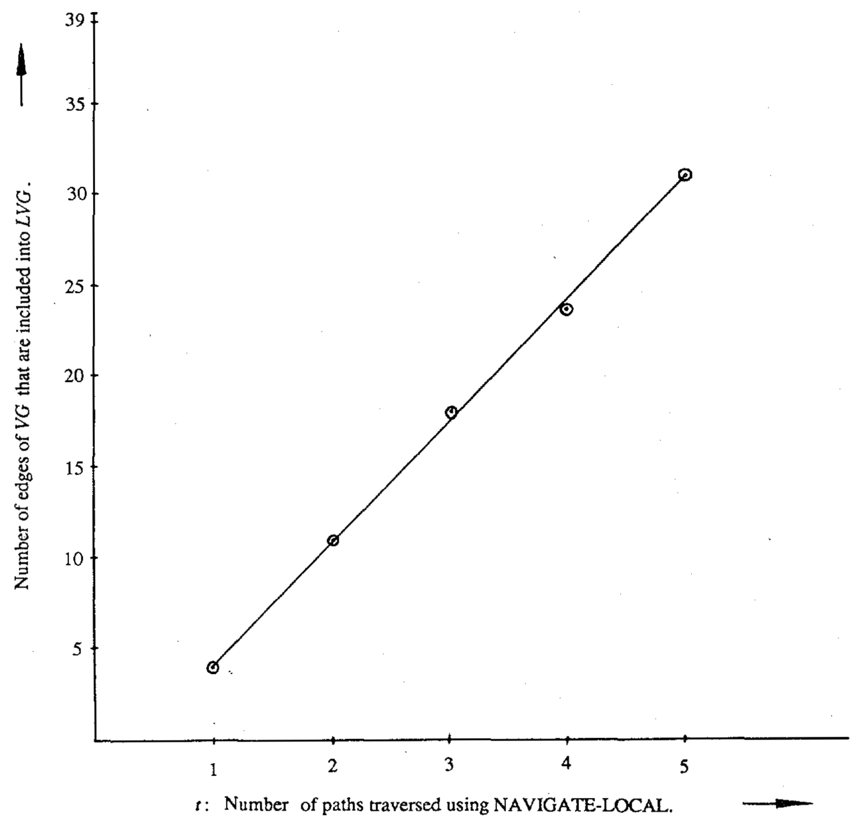


Fig. 16. Graph showing number of edges in LVG as function of number of paths traversed using NAVIGATE-LOCAL. Out of total of 39, robot learns 31 edges in five traversals.

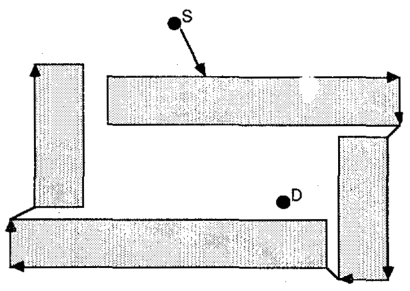


Fig. 17.

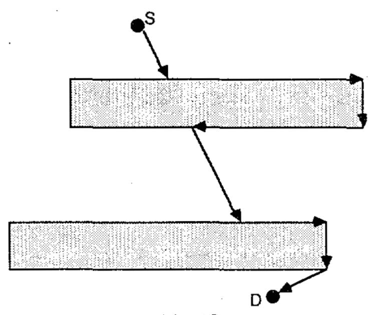


Fig. 18.

avenues for further work. We are also grateful to the anonymous referees who gave us both critical and extremely encouraging feedback, which later led to the Appendix.

References

[1] A. Aho, J. Hopcroft, and J. Ullman, *The Design and Analysis of Computer Algorithms*. Reading, MA: Addison-Wesley, 1974.

[2] R. A. Brooks, "Visual map making for a mobile robot," in *Proc. 1985 IEEE Int. Conf. Robotics and Automation*, 1985, pp. 824–829.

[3] R. A. Books and T. Lozano-Perez, "A subdivision algorithm in configuration space for path with rotation," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-15, no. 2, pp. 224–233, Mar./Apr. 1985.

[4] R. Chatila, "Path planning and environment learning in a mobile robot system," in *Proc. European Conf. Artificial Intelligence*, Torsey, France, 1982.

[5] R. Chattergy, "Some heuristics for the navigation of a robot," *Int. J. Robotics Res.*, vol. 4, no. 1, pp. 59–66, Spring 1985.

[6] J. L. Crowley, "Navigation of an intelligent mobile robot," *IEEE J. Robotics Automat.*, vol. RA-1, no. 2, pp. 31–41, Mar. 1985.

[7] N. Deo, *Graph Theory with Applications to Engineering and Computer Science*. Englewood Cliffs, NJ: Prentice-Hall, 1974.

[8] G. Giralt, R. Sobek, and R. Chatila, "A multilevel planning and navigation system for a mobile robot," in *Proc. 6th Int. Joint Conf. Artificial Intelligence*, Aug. 1979, Tokyo, Japan, pp. 335–338.

[9] L. Gouzenes, "Strategies for solving collison-free trajectories problems for mobile and manipulator robots," *Int. J. Robotics Res.*, vol. 3, no. 4, pp. 51–65, Winter 1984.

[10] S. S. Iyengar, C. C. Jorgensen, S. V. N. Rao, and C. R. Weisbin, "Robot navigation algorithms using learned spatial graphs," *Robotica*, vol. 4, pp. 93–100, Jan. 1986.

[11] ——, "Learned navigation paths for a robot in unexplored terrain," in *Proc. 2nd Conf. Artificial Intelligence Applications and Engineering of Knowledge Based Systems*, Miami Beach, FL, Dec. 1985.

[12] J. Laumond, "Model structuring and concept recognition: Two aspects of learning for a mobile robot," in *Proc. 8th Conf. Artificial Intelligence*, Aug. 1983, Karlsruhe, W. Germany, p. 839.

[13] D. T. Lee and F. P. Preparata, "Computational geometry—A survey," *IEEE Trans. Comput.*, vol. C-33, pp. 1072–1101, Dec. 1984.

[14] T. Lozano-Perez, "Spatial planning: A configuration space approach," *IEEE Trans. Comput.*, vol. C-32, pp. 108–120, Feb. 1983.

[15] T. Lozano-Perez and M. A. Wesley, "An algorithm for planning collision-free paths among polyhedral obstacles," *Commun. ACM*, vol. 22, pp. 560–570, Oct. 1979.

[16] V. J. Lumelsky and A. A. Stepanov, "Effect of uncertainty on continuous path planning for an autonomous vehicle," in *Proc. 23rd IEEE Conf. Decision and Control*, 1984, pp. 1616–1621.

[17] H. P. Moravec, "The CMU rover," in *Proc. Nat. Conf. Artificial Intelligence*, Aug. 1982, pp. 377–380.
[18] N. J. Nilsson, "Mobile automation: An application of artificial intelligence techniques," in *Proc. 1st Int. Joint Conf. Artificial Intelligence*, May 1969, pp. 509–520.
[19] J. B. Oommen and I. Reichstein, "On the problem of translating an elliptic object through a workspace of elliptic obstacles," *Robotica*, vol. 5, pp. 187–196, 1987.
[20] N. S. V. Rao, S. S. Iyengar, C. C. Jorgensen, and C. R. Weisbin, "On the robot navigation in an unexplored terrain," *J. Robotic Syst.*, vol. 3, pp. 389–407, 1986.
[21] A. M. Thompson, "The navigation system of the JPL robot," in *Proc. 5th Int. Joint Conf. Artificial Intelligence*, Aug. 1977, Cambridge, MA, pp. 749–757.
[22] M. P. Turchen and A. K. C. Wong, "Low level learning for a mobile robot: Environmental model acquisition," in *Proc. 2nd Int. Conf. Artificial Intelligence and Its Applications*, Dec. 1985, pp. 156–161.
[23] S. M. Udupa, "Collision detection and avoidance in computer controlled manipulators," in *Proc. 5th Int. Conf. Artificial Intelligence*, Mass. Inst. Technol., Cambridge, Aug. 1977, pp. 737–748.
[24] R. Wallace *et al.*, "First results in robot road following," in *Proc. 9th Int. Conf. Artificial Intelligence*, Aug. 1985, Los Angeles, CA, pp. 1089–1095.
[25] C. R. Weisbin *et al.*, "Machine intelligence for robotics applications," in *Proc. 1985 Conf. Intelligent Systems and Machines*, Apr. 1985.
[26] S. Whitesides, "Computational geometry and motion planning," in *Computational Geometry*, G. Toussaint, Ed. New York: North Holland, 1985.
[27] N. Andrade, M.C.S. thesis, School of Computer Science, Carlton Univ., Ottawa, ON, Canada, in preparation.

Adaptive Friction Compensation in DC-Motor Drives

C. CANUDAS, K. J. ÅSTRÖM, FELLOW, IEEE, AND K. BRAUN

Abstract—**A control scheme is proposed where the nonlinear effects of friction are compensated adaptively. When the friction is compensated, the motor drive can approximately be described by a constant coefficient linear model. Standard methods can be applied to design a regulator for such a model. This results in a control law which is a combination of a fixed linear controller and an adaptive part which compensates for nonlinear friction effects. Experiments have clearly shown that both static and dynamic friction have nonsymmetric characteristics. They depend on the direction of motion. This is considered in the design of the adaptive friction compensation. The proposes scheme has been implemented and tested on a laboratory prototype with good results. The control law is implemented on an IBM PC. The ideas, algorithm, and experimental results are described. The results are relevant for many precision drives, such as those found in industrial robots.**

I. Introduction

Adaptive control has predominantly dealt with generic models where all parameters are unknown. Such an approach has the advantage that it is general but also has the disadvantage that many parameters have to be estimated. Much of the work on adaptive control has also been confined to linear systems. In practice, many adaptive problems exist where the system can be described as partially known in the sense that part of the system dynamics is known and another part unknown. In this communication we consider a problem of this type, namely, a servo with nonlinear friction. Friction, which is always present to some degree, causes difficulties and gives rise to poor performance in precision servos in robots and other applications.

Manuscript received May 21, 1986; revised January 11, 1987. This work was supported in part by the Swedish Board of Technical Development under Grant 85-3225. This communication was presented in part at the IEEE Conference on Robotics and Automation, San Francisco, CA, April 1986.

C. Canudas is with the Laboratoire d'Automatique de Grenoble, B.P. 46, 38402, Saint-Martin-d'Hères, France.
K. J. Åström is with the Department of Automatic Control, Lund Institute of Technology, Box 118, S-22100, Lund, Sweden.
K. Braun was with the Fachgruppe fur Automatik, ETH-Zentrum, CH-8092 Zurich, Switzerland.

IEEE Log Number 8716969.

Velocity control of a servo motor with friction is considered. It is assumed that static and viscous frictions can be described as nonlinear functions of the angular velocity. The friction characteristics depend on the direction of the motion. The model can thus be split into two parts, depending on the direction of motion. The model isolates the friction torque effects and cancels them by feedback compensation.

Adaptive friction compensation has been considered before [17]. It was treated with model reference techniques in [7] and more recently in [15] and [10]. This work differs in the friction model and in the adaptive control law used.

The adaptive scheme introduced here attempts to use the *a priori* information available, i.e., the structure of the nonlinearity and the knowledge of some of the parameters. It seems natural to use adaptive schemes with explicit identification which utilizes this *a priori* information. Only those parameters which are not known *a priori* are estimated.The estimates are used to compensate for the friction–torque effects, and a linear control design is used to control the approximately linear system that is obtained when the friction effects are compensated. The final control structure can be viewed as a combination of a fixed linear controller and a feedback adaptive compensation.

The communication is organized as follows. Friction models proposed in the literature are discussed in Section II. A model where the friction torque is a piecewise-linear function of motor speed is established. This model captures static and dynamic friction effects. A strategy for friction compensation is presented in Section III. Section IV briefly describes the control laws for the linear system obtained when the friction effects are compensated. The design is a standard pole placement control. Section V proposes an adaptive version of the fixed friction compensation and proposes a possible design approach. The proposed ideas have been implemented on a laboratory prototype. The digital control laws were implemented using an IBM personal computer. The results of some experiments are shown in Section VI. Some conclusions are given in Section VII.

II. Mathematical Models

A dc motor with a permanent magnet was used in our experiments. Such motors are commonly used in robots and precision servos. The motor is provided with an electronic amplifier with current feedback. If all inertias are reflected to the motor axis, the motor can be described by the following model:

$$J\frac{d\omega}{dt}=KI(t)-T_f(t)+T_l(t). \tag{1}$$

Here J is the total moment of inertia reflected to the motor axis, K is the current constant, I is the motor current, T_f is the friction torque, and T_l is load disturbance torque. For the purpose of the investigation of the friction compensation, phenomena like compliance and torque ripple are not included in the model (1).

Friction Models

Friction models have been extensively discussed in the literature [5], [15], [7]. In spite of this, there is considerable disagreement on the proper model structure. It is well established that the friction torque is a function of the angular velocity. There is, however, disagreement concerning the character of the function. In the classical