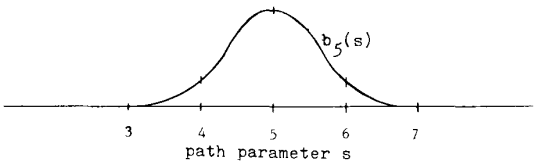


Fig. 10. The B-spline basis function $b_5(s)$.

For example, if $i = 5$, a plot of $b_5(s)$ is shown in Fig. 10. The function $b_6(s)$ is simply a copy of $b_5(s)$ shifted to the right by one interval. Note that for any s, only four of the b_i in (A1) will be nonzero. This property allows one to vary the curvature in certain portions of the path, without affecting others.

REFERENCES

[1] R. H. Bartels, J. C. Beatty and B. A. Barsky, "An introduction to the use of splines in computer graphics," U. C. Berkeley Tech. Rep. TR CSD, 83/136, Aug. 1983.

[2] J. E. Bobrow, S. Dubowsky and J. S. Gibson, "Time-optimal control of robotic manipulators along specified paths," *Int. J. Robotics Res.*, vol. 4, no. 3, Fall 1985.

[3] R. A. Brooks, "Solving the find-path problem by good representation of free space," in *Proc. AAAI 2nd Annu. Nat. Conf. on Artificial Intelligence* (Pittsburgh, PA, Aug. 18-20, 1982), pp. 381-386.

[4] S. Dubowsky and Z. Shiller, "Optimal dynamic trajectories for robotic manipulators," in *Proc. V, CISM-IFToMM Symp. on Theory and Practice of Robots and Manipulators* (Udine, Italy, 1984).

[5] S. Dubowsky, M. A. Norris and A. Shiller, "Time optimal trajectory planning for robotic manipulators," in *1986 IEEE Conf. on Robotics and Automation* (San Francisco, CA, Apr. 1986), pp. 1906-1912.

[6] R. Fletcher and M. J. D. Powell, "A rapidly convergent method for minimization," *Comput. J.*, vol. 6, no. 2, pp. 163-168, 1963.

[7] E. G. Gilbert and D. W. Johnson, "Distance functions and their application to robot path planning in the presence of obstacles," *IEEE J. Robotics Automat.*, vol. RA-1, pp. 21-30, Mar. 1985.

[8] D. W. Johnson and E. G. Gilbert, "Minimum-time robot path planning in the presence of obstacles," in *24th Conf. on Decision and Control* (Ft. Lauderdale, FL, Dec. 1985), pp. 1748-1753.

[9] M. E. Kahn and B. Roth, "The near-minimum time control of open-loop articulated kinematic chains," *ASME J. Dynamic Syst., Meas., Contr.*, vol. 93, pp. 164-172, Sept. 1971.

[10] D. E. Kirk, *Optimal Control Theory*. Englewood Cliffs, NJ: Prentice-Hall, 1970, pp. 245-246.

[11] T. Lozano-Pérez, "Automatic planning of manipulator transfer movements," *IEEE Trans. Syst. Man, Cybern.*, vol. SMC-11, pp. 681-698, Oct. 1981.

[12] J. Y. S. Luh and C. E. Campbell, "Minimum distance collision-free path planning for industrial robots with a prismatic joint," *IEEE Trans. Automat. Contr.*, vol. AC-29, pp. 675-680, Aug. 1984.

[13] E. B. Meier and A. E. Bryson, "An efficient algorithm for time-optimal control of a two-link manipulator," in *AIAA Conf. on Guidance and Control* (Monterey, CA, Aug. 1987), pp. 204-212.

[14] V. T. Rajan, "Minimum time trajectory planning," in *1985 IEEE Conf. on Robotics and Automation* (St. Louis, MO, Mar. 1985), pp. 759-764.

[15] E. G. Shin and N. D. McKay, "Minimum-time control of robotic manipulators with geometric path constraints," *IEEE Trans. Automat. Contr.*, vol. AC-30, pp. 531-541, June 1985.

[16] K. G. Shin and N. D. McKay, "Selection of near-minimum time geometric paths for robotic manipulators," *IEEE Trans. Automat. Contr.*, vol. AC-31, no. 6, pp. 501-511, June 1986.

[17] G. N. Vanderplaats, ADS, "A Fortran program for automated design synthesis, Version O.O," Naval Postgraduate School, Monterey CA, 1983.

[18] G. N. Vanderplaats, *Numerical Optimization Techniques for Engineering Design: With Applications*. New York, NY: McGraw-Hill, 1984.

[19] A. Weinreb and A. E. Bryson, "Optimal control of systems with hard control bounds," *IEEE Trans. Automat. Contr.*, vol. AC-30, no. 11, pp. 1135-1138, Nov. 1985.

On Terrain Model Acquisition by a Point Robot Amidst Polyhedral Obstacles

NAGESWARA S. V. RAO, S. S. IYENGAR, B. JOHN OOMMEN, AND R. L. KASHYAP

***Abstract*—We consider the problem of terrain model acquisition by a roving point placed in an *unknown* terrain populated by stationary polyhedral obstacles in two/three dimensions. The motivation for this problem is that after the terrain model is completely acquired, navigation from a source point to a destination point can be achieved along the collision-free paths. And this can be done without the usage of sensors by applying the existing techniques for the well-known find-path problem. In this communication, the Point Robot Autonomous Machine (PRAM) is used as a simplified abstract model for real-life roving robots. We present an algorithm that enables PRAM to autonomously acquire the model of an unexplored obstacle terrain composed of an unknown number of polyhedral obstacles in two/three dimensions. In our method, PRAM undertakes a systematic exploration of the obstacle terrain with its sensor that detects all the edges and vertices visible from the present location, and builds the complete obstacle terrain model.**

I. INTRODUCTION

In recent times there has been an enormous spurt of research activity in the algorithmic aspects of motion planning. The problem of navigating a body through a terrain populated by a set of *known* obstacles (i.e., the precise geometric characterization of the obstacles is available) is solved in many cases. Lozano-Pérez and Wesley [3], O'Dunlaing and Yap [5], Reif [7], and Schwartz and Sharir [8] present some of the most fundamental solutions to this problem. Whitesides [10] presents a comprehensive treatment on these and other solutions to the find-path and related problems. In all these methods, the precise model of the obstacle terrain is *known a priori*, and path planning is done entirely computationally. Once a path is planned, the robot moves along the planned path, and no sensors are used for navigational purposes.

Another interesting problem is the navigation of a robot in an unexplored or a partially explored terrain. In this case, the entire terrain model may not be known, and the robot relies on its sensors for navigation. Lumelsky and Stepanov [4] present sensor-based navigation algorithms for navigating a point automaton to a destination point using "touch" type of sensor. In this method localized sensor information is used to guide the point automaton, and this information is not put to any further global use. In many applications, *incidental learning* is shown to be an important enhancement in the navigation planning. Here, a composite model of the terrain is built by integrating the sensor information obtained as the robot executes sensor-based and goal-directed navigation. Iyengar *et al.* [2], Oommen *et al.* [6], Turchan and Wong [9] discuss different versions of learned navigation in unexplored terrains. Here we consider the problem of acquiring the terrain model by systematic exploration of the terrain using a sensor. Our main motivation stems from the fact that the availability of the terrain model enables us to plan the entire

Manuscript received February 26, 1987; revised December 7, 1987. A preleminary version of this paper was presented at the 3rd IEEE Conference on AI Applications, Orlando, FL, Feb. 1987. The work of B. J. Oomman was partially supported by the National Sciences and Engineering Council of Canada.

N. S. V. Rao is with the Department of Computer Science, Old Dominion University, Norfolk, VA 23529-0162.

S. S. Iyengar is with the Department of Computer Science, Louisiana State University, Baton Rouge, LA 70803.

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

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

IEEE Log Number 8820163.

navigation path using the techniques of known terrains and without using sensors. This is to be contrasted with the techniques based on incidental learning, where sensors may have to be used for navigation planning at every stage.

In this communication, we introduce the *Point Robot Autonomous Machine* (PRAM) as a simplified abstract model for a mobile robot. Now, the *terrain acquisition problem* can be stated as follows: The PRAM is initially placed in a completely unexplored terrain populated by an unknown (but, finite) number of polyhedral obstacles of unknown sizes and locations. The obstacles are known to be polyhedra (finite in number and each with a finite number of vertices). It is also known that the terrain is finite-sized, i.e., the terrain can be inscribed in a circle/sphere of finite radius in two/three dimensions. No other information about the obstacles is available to PRAM. Then, PRAM is required to scan the terrain and autonomously acquire the dimensions of all edges and the locations of all vertices for each obstacle. In other words, the complete terrain model should be obtained by PRAM with each obstacle described as a polyhedron in two/three dimensions. Our problem is similar, in "spirit," to that of Cole and Yap [1] where the shape of an n-gon is to be detected by probing with a tactile sensor. Here, the main idea behind the terrain acquisition is that after the terrain model is built, the path planning can be carried out "computationally" using the techniques for known terrains such that: a) the paths of navigation can be made optimal in possible cases, b) no further usage of the sensor equipment is necessary for navigation planning. In other words, the terrain acquisition (in a way) is a precursory problem to the find-path problem which is well-known in robotics.

One of the important requirements on any terrain acquisition algorithm is to ensure that the model of every obstacle is completely acquired. We prove the completeness of the acquisition process by establishing that the visibility graph of the terrain is connected, and making use of the fact that a depth-first search algorithm visits all nodes of a connected graph.

The communication is organized as follows: Section II discusses the problem definition and the abstract robot model. In Section III, we describe the terrain model acquisition algorithm. We analyze the performance of the algorithm in terms of the robot motion parameters and computational complexity in Section IV. In Section V we present an example to illustrate the process of terrain model acquisition in simple terrain.

II. Preliminaries

We now describe the *Point Robot Autonomous Machine* (PRAM). PRAM is point-sized (in two/three dimensions), and houses a computing device with a finite storage capability. Formally, PRAM supports two navigation instructions—SCAN(L) and MOVE(v) which are described below:

a) The execution of SCAN(L) is as follows: All obstacle vertices that are visible from the present location of PRAM are returned in the list L. Furthermore, if PRAM is located at an obstacle vertex v, all obstacle edges that are incident at v are marked and returned.

b) The instruction MOVE(v) moves PRAM in a straight line to the point v from its present location. Here v could be a point in the free space or an obstacle vertex.

The execution of a single SCAN(L) is termed as a *scan operation*. Note that the scan operation is a very-high-level abstraction of sensing operations performed by real-life robots. Such operations, in general, involve the actual process of sensing and further processing; both these activities could be time-consuming and computationally complex. We keep this entire process as a single logical entity, namely, the scan operation. By doing this we do not imply the difficulty or ease with which such an operation can be carried out. The execution of a single MOVE(v) is termed as an *elementary traversal*. We assume that these instructions are executed without errors. We characterize the performance of an algorithm for PRAM in terms of the number of scan operations and elementary traversals. We also consider the complexities of computations carried by the computing device housed on PRAM. Our PRAM is very similar to the *Point Automaton* (PA) proposed earlier by Lumelsky and Stepanov [4]. But, whereas PRAM has memory, the PA does not support this feature. Another difference is that PRAM is equipped with a "scan" sensor as opposed to the "touch" sensor of PA. Further, PRAM can navigate only along straight lines, whereas PA can navigate along any arbitrary simple curve.

Visibility graphs have been extensively studied in computational geometry and robot motion planning [3], [6]. Formally, the *Visibility Graph*, $VG(O) = (V, E)$, of an obstacle terrain O is defined as follows (we assume that O is finite-sized and consists of a finite number of polyhedral obstacles; each obstacle has a finite number of vertices):

i) V is the union of vertices of all obstacles,
ii) 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 either an edge of an obstacle or it is not intersected by any obstacle.

Our solution to this problem is based on planning the motion of PRAM in such a way that the order in which the new vertices are visited corresponds to the depth-first search (DFS) traversal of $VG(O)$. Observe that the $VG(O)$ is initially not known to PRAM. The adjacency list of any vertex v of $VG(O)$ can be obtained by placing PRAM at v and obtaining the vertices visible from v (by performing a scan operation). After this operation, the adjacency list of v is stored in the *Partial Visibility Graph*, denoted by $PVG(O)$, which is available for further usage. The $PVG(O)$ is augmented after each visit to a new vertex. Note that in order to simulate a depth-first search, at any stage, we only need to know the adjacency lists of the vertices visited so far and the vertex at which PRAM is located at present. These adjacency lists are known through scan operations performed from appropriate vertices. We shall show that $VG(O)$ is connected. We then make use of the fact that a DFS traversal on a connected graph visits all the vertices. Thus by simulating a DFS using PRAM we make sure that all the vertices of the obstacles are visited. At this stage $PVG(O)$ converges to $VG(O)$ from which the complete terrain model is obtained by grouping the marked edges that correspond to the individual obstacles.

III. Terrain Model Acquisition Algorithm

To describe the terrain model acquisition algorithm we shall specify the steps that are executed and simultaneously allude to the lines of code they represent in the algorithm ACQUIRE described below. Initially PRAM performs a scan operation and moves to a vertex v_0. Then it systematically visits the obstacle vertices. Let PRAM be presently located at a vertex v (initially $v = v_0$). PRAM performs a scan operation (from v) and stores the adjacency list of v in $PVG(O)$ (lines 1–2 of ACQUIRE). At this point the vertex v is pushed onto a stack called PATH-STACK. Here, we have two cases. In the first case, some of the adjacent vertices of v are not visited earlier by PRAM. Then v^*, an unvisited vertex in L nearest to v, is computed and PRAM moves to v^* (lines 8–10 of ACQUIRE). From v^*, ACQUIRE is recursively invoked (line 11). In the second case, all adjacent nodes of v are already visited by PRAM. Now the algorithm PLAN-PATH is used to obtain a vertex v^* to visit next, and PRAM moves to v^* (lines 4–5 of ACQUIRE). If $v^* \neq v_0$, then ACQUIRE is recursively applied from v^* (lines 6–7), and ACQUIRE terminates otherwise. At the termination of ACQUIRE, we appropriately collect the edges that belong to individual obstacles (line 12). In Lemma 1 we show that each obstacle gives rise to a connected component in $VG(O)$ entirely consisting of marked edges. These components can be obtained in linear time (in number of edges and vertices) using standard connected component algorithms. This description provides the complete obstacle terrain model.

algorithm ACQUIRE(v);
begin
1. SCAN(L);
2. update the $PVG(O)$ with information from L and store v on PATH-STACK;
3. **if** (all nodes adjacent to v are visited) **then**
4. PLAN-PATH(v^*, P);

```
5.    move along the path specified by P;
6.    if (v* ≠ v0) then
7.       ACQUIRE(v*);
      end-if;
8.  else
9.    v* ← unvisited vertex in L nearest to v;
10.   MOVE(v*);
11.   ACQUIRE(v*);
    end-if;
12. appropriately group the marked edges corresponding to
    individual obstacles;
end;
```

The algorithm PLAN-PATH essentially manipulates PATH-STACK on which the path taken by PRAM is stored. The top of the stack is repeatedly popped until a vertex v_2 with an unvisited adjacent node is found (lines 2–4 of PLAN-PATH). A shortest path, in terms of the number of edges, to an unvisited node v^* adjacent to v_2 is computed by using Dijkstra's shortest path algorithm and is returned in P (lines 9–10 of PLAN-PATH). The PRAM moves along this shortest path to v^* (line 5 of ACQUIRE). If no vertex with unvisited adjacent nodes is found on PATH-STACK, then a shortest path to v_0 is planned (as in line 7 of PLAN-PATH), and the acquisition process is terminated. This process is formally described in the algorithm PLAN-PATH.

At this stage, we wish to note a difference between the DFS algorithm and ACQUIRE. Consider a stage at which PRAM is at vertex v, and all nodes adjacent to v are visited. Let w be the vertex on the stack obtained on PATH-STACK by PLAN-PATH and let v^* be the adjacent node of w chosen to visit next. The backtrack path followed (conceptually) by the DFS algorithm is from v to w and then to v^*. Note that PRAM moves to v^* along the path obtained by using Dijkstra's algorithm on $PVG(O)$. This path may or may not coincide with the path followed by the DFS algorithm.

```
algorithm PLAN-PATH(v*, P);
 begin
1.  v2 ← top element of PATH-STACK;
2.  while (PATH_STACK in not empty) and (all nodes adjacent
    to v2 are visited) do
3.     pop out the top of PATH-STACK;
4.     v2 ← top element of PATH-STACK;
    end-while;
5.  if (all nodes of v2 are visited) then
6.     v* ← v0
7.     return a shortest path to v0 in P;
8.  else
9.     find a shortest path to an unvisited node adjacent to v2;
10.    return the planned path in P;
    end-if;
 end;
```

We now show the correctness of ACQUIRE. Our proof consists of two steps: First, we prove that $VG(O)$ is *connected*, i.e., there is a path from every vertex to every other vertex in $VG(O)$ (Lemma 1). Then, we use the property that the execution of ACQUIRE by PRAM is equivalent to performing a depth-first search on $VG(O)$.

Lemma 1: The graph VG(O) is connected.

Proof: Let $EXT(O_i)$ denote the exterior of an obstacle polyhedron $O_i \in O$. Let $VER(O_i)$ and $EDG(O_i)$ be the sets of vertices and edges, respectively, of the obstacle O_i. The graph $G_i = (VER(O_i), EDG(O_i))$ is connected because every vertex of a polyhedron O_i can be reached from every other vertex by traversing along the edges of O_i. Hence, the connectivity of $VG(O)$ can be shown by showing that there exists at least one path between each pair of graphs $(VER(O_i), EDG(O_i))$ and $(VER(O_j), EDG(O_j))$, for $i \neq j$.

First we show that $VG(O)$ is connected if each $O_i \in O$ is a convex polyhedron. Let $VISI(v)$, for $v \in VER(O_i)$ be the points visible from v, when only the O_i is present in the obstacle terrain, i.e., for $x \in VISI(v)$, the line segment joining x and v lies entirely in $EXT(O_i)$. We have

$$\bigcup_{v \in VER(O_i)} VISI(v) = EXT(O_i)$$

for a convex polyhedron O_i. Let the obstacle terrain consist of exactly two convex obstacles O_1 and O_2. It is easily seen that at least one edge exists (that coincides with line/plane of support) between one vertex of O_1 and one of the vertices of O_2. Thus O_1 and O_2 form a connected graph.

Consider placing another obstacle O_3 in the existing terrain. First consider the two-dimensional case. For each vertex v of O_3 let v_1 and v_2 denote the vertices adjacent to v such that O_3 lies to the right of the line segments $\overline{v_1 v}$ and $\overline{v v_2}$ ($\overline{pq}$ denotes the line segment joining two points p and q). Imagine a semi-infinite ray r originating from v and containing $\overline{v_1 v}$. Let us sweep r in the clockwise direction until r contains $\overline{v v_2}$. By sweeping such rays from every vertex of O_3 we cover the entire $EXT(O_3)$. Since both O_1 and O_2 are contained in $EXT(O_3)$, the ray touches one of O_1 and O_2 in one of the configurations shown in Fig. 1. The obstacle O_1 and O_2 may be encountered separately by r as in Fig. 1(a). Alternatively, one obstacle may cover the other as in Fig. 1(b). In the third case, the obstacles may be as shown as in Fig. 1(c). In all these cases at one point of the r's touches one of the vertices of O_1 or O_2. This implies that there is an edge between one of the vertices of O_3 and a vertex of O_1 or O_2. Now consider the three-dimensional case. Let v be vertex and let $f_1, f_2, \cdots, f_k$ be the clockwise listing of faces that meet at v when we look at v from outside of O_3. Let e_i be the edge (that contains v) between f_i and f_{i+1} (e_k is the edge between f_k and f_1). Now consider the half-plane with e_i as end line. Let us sweep this plane (in the exterior of O_3) with e_i as axis; initially, this plane contains f_i and after the sweep contains f_{i+1}. It is clear that by sweeping all planes corresponding to all vertices of O_3 we cover the $EXT(O_3)$. By using the earlier arguments at least one plane should touch one of the vertices of either O_1 or O_2. This proves the existence of the suitable edge. We observe that at least one vertex of O_3 lies in $VISI(v)$, for $v \in VER(O_1) \cup VER(O_2)$. Hence, $VG(O)$ for $O = \{O_1, O_2, O_3\}$ is a connected graph. This argument can be extended for any finite number of convex polyhedra. Hence, $VG(O)$ is connected if every obstacle polyhedron in O_i is a convex.

Consider the terrains with nonconvex obstacles. Consider the convex hull $CH(O_i)$ formed by joining the "outer" vertices of $O_i \in O$. If two obstacles O_i and O_j are such that $CH(O_i) \cap CH(O_j) \neq \phi$, then at least one obstacle enters a "concavity" of the other. We can apply the "sweeping" method (sweeping area restricted to the concavity) to show that an edge exists between $VG(\{O_i\})$ and $VG(\{O_j\})$. Let us "conceptually" combine these two obstacles, and note that $VG(\{O_i, O_j\})$ is connected. Let us recursively apply this technique on the resultant terrain to obtain a terrain of "combined" obstacles denoted by

$$O' = \{O_1', O_2', \cdots, O_m'\}, \qquad m \leq |O|$$

and

$$\bigcup_{i=1}^{|O|} VER(O_i) = \bigcup_{i=1}^{m} VER(O_i').$$

By our construction

$$CH(O_i') \cap CH(O_j') = \phi.$$

Consider a vertex $v \in VER(O_i)$ and $v_1 \notin VER(CH(O_i'))$. There is always a path from v to v_1 along the edges of $VG(\{O_i'\})$ and thus v and v_1 are connected. Hence $VG(\{O_i'\})$ is connected for $i = 1, 2, \cdots, m$. Now $VG(\{CH(O_1'), CH(O_2'), \cdots, CH(O_m')\})$ is connected since each $CH(O_i)$ is convex. Thus $VG(O)$ is connected. □

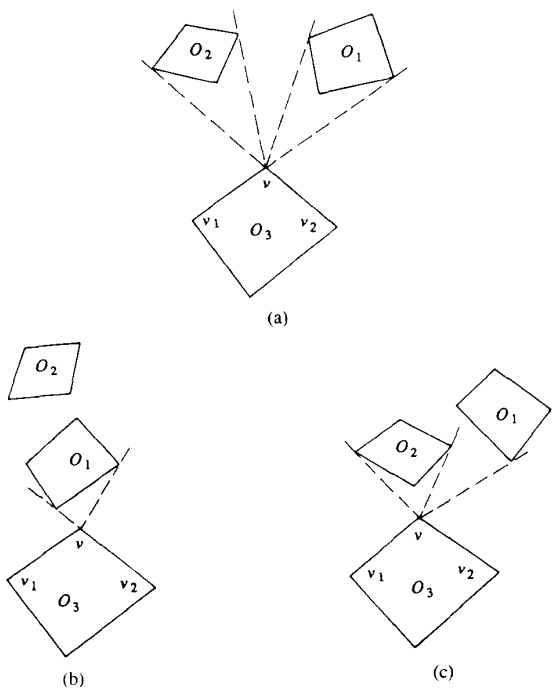


Fig. 1. Possible configurations of O_1, O_2, and O_3.

A close look at the algorithm ACQUIRE reveals the following property:

Property 1: The order in which the unexplored vertices of the obstacle terrain O are visited by PRAM while executing ACQUIRE is exactly the same as the order in which the nodes of VG(O) are visited when a depth-first-search traversal is performed on VG(O).

□

Note that the process of visiting an obstacle vertex by PRAM involves physically locating PRAM at the vertex. Whereas the process of accessing a graph node of $VG(O)$ that corresponds to an obstacle vertex (by say algorithm PLAN-PATH) involves an access to the memory. The correctness of the algorithm ACQUIRE directly follows from the Lemma 1 and Property 1.

Theorem 1: The algorithm ACQUIRE builds the complete obstacle terrain model in a finite amount of time.

Proof: From Lemma 1 the $VG(O)$ is a connected graph. Hence, any depth-first traversal on $VG(O)$ visits all the nodes in a finite amount of time (note that $VG(O)$ has finite number of nodes). Thus using Property 1, we conclude that the entire $VG(O)$ is built from the sensor readings taken from each of the vertices of the obstacles. Then, the terrain model is built from the $VG(O)$ by appropriate grouping of the obstacle edges.

□

IV. Performance Analysis

In this section, we analyze the performance of ACQUIRE in terms of the number of scan operations and elementary traversals, and also in terms of computational complexity. Let N denote the total number of obstacle vertices.

Theorem 2: To acquire the complete model of the obstacle terrain O, using ACQUIRE,

a) the total number of scan operations required is N

b) the total number of elementary traversals is at most 2(N − 1).

Proof: Part a) directly follows from Lemma 1 and Property 1. We now prove Part b). We observe that when PRAM accesses the PATH-STACK for finding the next stop point w, a path to w always exists along the DFS tree. Thus in the worst case PRAM backtracks along this path on the DFS tree. Any other path to w planned by PLAN-PATH will have no more edges than this path. Note that PRAM backtracks along a path at most once, because once the path is removed from the PATH-STACK it will not be pushed onto it again. In the worst case, all the paths planned by PLAN-PATH are along the edges of the DFS tree. Thus in the worst case, each edge of the DFS tree is traversed twice, hence the theorem.

□

The computational efforts involved in the execution of ACQUIRE are estimated in Theorem 3. We maintain a table, called MAP-TABLE, to obtain a node number in $VG(O)$ for any obstacle vertex specified by its coordinates. The MAP-TABLE is maintained as an AVL-tree: the value of each node is obtained by concatenating the coordinate values and treating it as a single value. Thus any vertex of an obstacle is uniquely represented as a node specified by a single value. Along with each node of the AVL-tree, the corresponding node number in $PVG(O)$ is stored. Additionally, the information indicating whether a vertex is visited or not is also stored in the corresponding node of the AVL-tree. Thus complexity of finding the node number in $PVG(O)$ for any vertex that is specified by its coordinates is $O(\log N)$.

Theorem 3: In acquiring the complete model of the obstacle terrain O using ACQUIRE, the computational complexities of various operations are as below:

a) the total number of node accesses is $O(N^3)$;

b) the complexity of constructing MAP-TABLE is $O(N \log N)$;

c) the number of accesses to MAP-TABLE is $O(E \log N)$, where E is the number of edges of VG(O);

d) the complexity of storage is $O(N^2)$.

Proof: a) The $PVG(O)$ is accessed by the algorithm for planning the shortest paths from the current vertex to another unvisited vertex using Dijkstra's shortest path algorithm (lines 9–10 of algorithm PLAN-PATH). The planning of each path accesses $O(N^2)$ nodes, and the number of path planning operations is given by $O(N)$. Thus total number of the node accesses in the complete execution of the algorithm ACQUIRE is $O(N^3)$.

b) A vertex is inserted into MAP-TABLE when it is detected by a SCAN operation. The cost of each insertion is $O(\log N)$, and there are N such insertions. Thus part b) is proven.

c) The MAP-TABLE is accessed while inserting new vertices detected as the result of a scan operations. The vertices are checked for membership in MAP-TABLE before insertion. The number of such operations is $O(E)$. Thus the complexity of this task is $O(E \log N)$. The MAP-TABLE is also accessed while finding whether all the nodes adjacent to a given node are visited (as in line 2 of the algorithm PLAN-PATH). For each node on the stack the number of accesses to MAP-TABLE is equal to its degree in $PVG(O)$. Hence, the total number of times the MAP-TABLE is accessed for this purpose is at most twice the sum of the degrees of all nodes in PVG. Thus the MAP-TABLE is accessed $O(E)$ times and the total number of accesses to the MAP-TABLE is $O(E \log N)$.

d) The complexity of storing the visibility graph is $O(N^2)$. The storage complexity of PATH-STACK is $O(N)$ and that of MAP-TABLE is $O(N)$. Thus the total complexity is $O(N^2)$. Hence, the theorem.

□

In the next section we present an example to illustrate the working of the terrain acquisition algorithm.

V. Example

Consider the two-dimensional obstacle terrain shown in Fig. 2. Initially, PRAM is located at vertex 1, and PRAM does not have any terrain model. Then PRAM scans the terrain from vertex 1, and detects vertices 1_1, 1_2, 1_3, 1_4, 2, 1_5, which are visible from vertex 1 (see Fig. 3(a)). At this point, the vertices 2 and 1_5 in the adjacency list of the vertex v are specially marked to indicate that (1, 1_5) and (1, 2) are edges of an obstacle. Then, PRAM moves to vertex 2, which is the nearest to vertex 1. The path taken by the robot is shown in bold

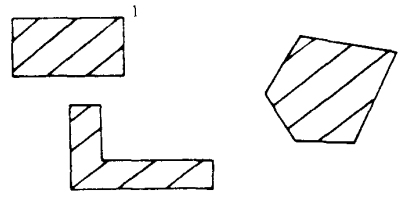
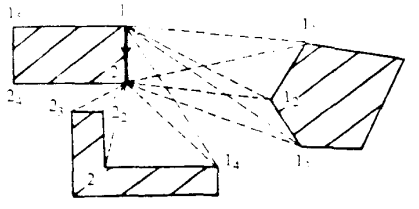
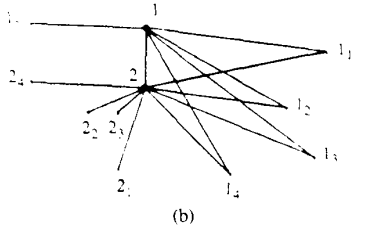


Fig. 2. The unexplored terrain.

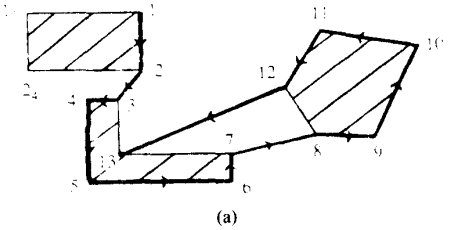


(a)

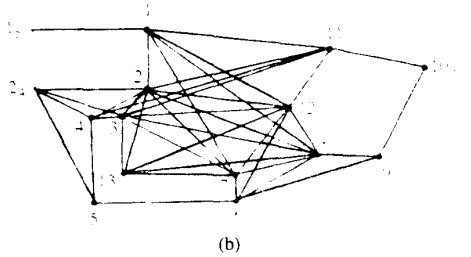


(b)

Fig. 3. Initial storage acquisition. (a) The PRAM is presently located at vertex 2. (b) *PVG*(*O*).



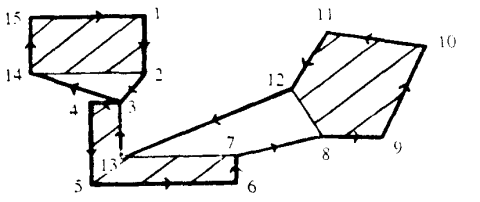
(a)



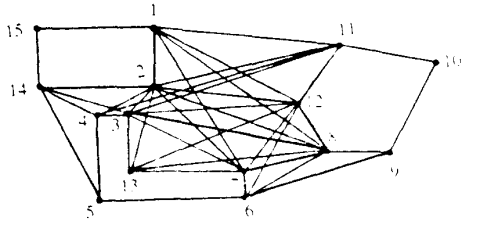
(b)

Fig. 4. Intermediate stage of acquisition. (a) The PRAM is presently located at vertex 13. (b) *PVG*(*O*).

lines with arrows. Then PRAM obtains the new vertices 2_1, 2_2, 2_3, 2_4. The *PVG* at this stage is shown in Fig. 3(b). Vertices 1 and 2, shown by bigger circles, are marked as "visited." The contents of the PATH-STACK are 1, 2 at this stage. In Fig. 4(a), we show an intermediate stage of terrain acquisition. The PRAM has moved to vertex 13. Until this stage, the procedure PLAN-PATH is not invoked. At this point the contents of PATH-STACK are 1, 2, $\cdots$, 13, and also all vertices visible from 13 are visited. Then PATH-STACK is popped, until a vertex with an unvisited neighbor vertex is found. Vertex 4 is found as a result since its neighbor vertex 2_4 is not visited. The path to 2_4 via 3 is found by Dijkstra's algorithm. Let 2_4 be called 14 for convenience. Then the contents of the PATH-STACK are changed to 1, 2, 3, 4, 14. The present *PVG* is shown in Fig. 4(b). From vertex 14 PRAM moves to vertex 15 (call 1_5 as 15 for convenience), and at this point all the neighbors of all the vertices are visited. Then PRAM moves back to vertex 1 (see Fig. 5(a)). Note that the number of MOVE($\cdot$) operations is 16. The complete visibility graph is shown in Fig. 5(b), from which the terrain model can be easily constructed.



(a)



(b)

Fig. 5. Completion of terrain model acquisition. (a) The PRAM returns to vertex 1. (b) *PVG*(*O*) converges *VG*(*O*).

VI. Conclusions

In this paper we consider the terrain model acquisition by a point robot roving in an obstacle terrain populated by an unknown number of polyhedral obstacles in two/three dimensions. We present a method that enables the point robot to acquire the complete terrain model in a finite amount of time. The implementation of the proposed technique on a real-life robot involves modifying the algorithm PATH-PLAN to account for the size and configuration of the robot. Specifically, at any stage during terrain acquisition, the paths are to be planned using the partially built terrain model. A two-dimensional version of ACQUIRE has been implemented on the HERMIES-II robot at Oak Ridge National Laboratory in Fortran 77 language running on an NCUBE control computer. The algorithm ACQUIRE is also implemented in a simulated mode in C on a VAX 11/780.

Acknowledgment

The authors wish to thank the anonymous reviewers whose careful and extensive comments have greatly improved the presentation of the material in the communication.

References

[1] R. Cole and C. K. Yap, "Shape from probing," *J. Algorithms*, vol. 8, no. 1, pp. 19-38, 1987.

[2] 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, 1986.

[3] T. Lozano-Pérez and M. A. Wesley, "An algorithm for planning collision-free paths among polyhedral obstacles," *Commun. ACM*, vol. 22, no. 10, pp. 560-570, 1979.

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

[5] C. O'Dunlaing and C. Yap, "A 'retraction' method for planning the motion of a disc," *J. Algorithms*, vol. 6, pp. 104-111, 1985.

[6] J. B. Oommen, S. S. Iyengar, N. S. V. Rao, and R. L. Kashyap, "Robot navigation in unknown terrains using learned visibility graphs. Part I: The disjoint convex obstacle case," *IEEE J. Robotics Automat.*, vol. RA-3, no. 6, pp. 672-681, Dec. 1987.

[7] J. Reif, "Complexity of the mover's problem and generalizations," in

Proc. 20th Symp. on Foundation of Computer Science, 1979, pp. 421–427.

[8] J. T. Schwartz and M. Sharir, "On the piano movers' problem I: The special case of a rigid polygonal body moving amidst polygonal barriers," *Commun. Pure Appl. Math.*, vol. 36, pp. 345–398, 1983.

[9] M. P. Turchen and A. K. C. Wong, "Low level learning for a mobile robot: Environmental model acquisition," in *Proc. 2nd Conf. on Artificial Intelligence Applications* (Miami Beach, FL, Dec. 1985), pp. 156–161.

[10] S. Whitesides, "Computational geometry and motion planning," in *Computational Geometry*, G. Toussaint, Ed. Amsterdam, The Netherlands: North-Holland, 1985.

[11] N. S. V. Rao, S. S. Iyengar, B. J. Oommen, and R. L. Kashyap, "Terrain acquisition by a point robot amidst polyhedral obstacles," in *Proc. 3rd Conf. on Artificial Intelligence Applications* (Orlando, FL, Feb. 1987), pp. 170–175.