N90-29029

LEARNED NAVIGATION IN UNKNOWN TERRAINS: A RETRACTION METHOD

Nageswara S.V. Rao
Dept. of Computer Science
Old Dominion University
Norfolk, VA 23529-0162

N. Stoltzfus
Dept. of Mathematics
Louisiana State University
Baton Rouge, LA 70803

S.S. Iyengar
Dept. of Computer Science
Louisiana State University
Baton Rouge, LA 70803

ABSTRACT

We consider the problem of learned navigation of a circular robot R, of radius δ (≥ 0), through a terrain whose model is not *a priori* known. We consider two-dimensional finite-sized terrains populated by an unknown (but, finite) number of simple polygonal obstacles. The number and locations of the vertices of each obstacle are unknown to R. R is equipped with a sensor system that detects all vertices and edges that are visible from its present location. In this context we deal with two problems. In the *visit problem*, the robot is required to visit a sequence of destination points, and in the *terrain model acquisition problem*, the robot is required to acquire the complete model of the terrain. We present an algorithmic framework for solving these two problems using a retraction of the free-space onto the Voronoi diagram of the terrain. We then present algorithms to solve the visit problem and the terrain model acquisition problem.

1. INTRODUCTION

We consider the problem of the collision-free navigation of a circular robot through an *unknown* terrain, i.e., a terrain whose model is not *a priori* known. Several variants of this problem have been investigated. An algorithm for a point robot to escape out of a maze using touch sensing ability is given in [1]. In [6], algorithms for a point robot to move from a source point to a destination point using touch sensing are presented. The algorithms that enable a point robot to navigate to a destination point, and at the same time "learn" about the parts of terrain that are encountered on the way to the destination are presented in [2,7]. This process of learning is termed as *incidental learning*. Here the robot uses a combination of touch sensing and distance probing. The same problem is also solved in the case where the point robot is equipped with a sensor that obtains all the visible obstacle boundaries [8]. The above problems can be grouped under a generic name of the *visit problem*, wherein a robot is required to visit a sequence of destination points through an unknown terrain. Another problem, called the *terrain model acquisition problem*, wherein a point robot is required to acquire the complete model of the terrain is also studied [10]. The solutions of [7,8,10] are based on an incremental construction of the visibility graph of the terrain. The above problems have to be distinguished from those that deal with the navigation in *known* terrains, i.e. the terrains whose models are available. A comprehensive treatment of these problems can be found in [11].

These formulations of the navigational problem are motivated by a practical application involving the development of an autonomous rescue robot. This robot is intended for carrying out rescue operations in nuclear power plants in the events of radiation leakages, and other incidents that prevent human operation. A solution to the visit problem helps in developing a robot that can carry out a set of operations in different locations in unfamiliar environments. Since the motion planning here is essentially sensor-based, the navigation involves expensive sensor operations. Further more, the robot could temporarily navigate into local detours because of the partial nature of the information returned by the sensors. By incorporating the incidental learning feature, we reduce the expected number of sensor operations, and the expected number of detours, as the robot visits newer locations. Instead, if the complete terrain model is available, the robot can avoid the local detours, and also avoid the expensive sensor operations. Thus a solution to the terrain model acquisition problem helps the robot in acquiring the terrain model during the period in between the rescue operations. A dedicated rescue robot typically idles in between two successive rescue operations, and the rescue operations could be fairly infrequent. In such cases, the resources are better utilized if the robot is employed in the terrain model acquisition process during this period. The proposed methodology in solving these navigational problems provides a basic algorithmic framework that aids the design of a navigational system for the abovementioned rescue robot. The same methodology can also aid the development of navigational systems for other autonomous mobile machines in applications such as space navigation, underwater explorations, maintenance of space laboratories etc. However, a practical implementation of the proposed system calls for advances in more general theoretical aspects as well as several other issues such as sensing and movement errors, etc., which are not discussed in the above works as well as in this paper.

The visit problem and the terrain model acquisition problem have been solved separately [7,8,10]. In this paper, we present a unified framework for solving both the problems using a method based on a retraction of free-space onto the Voronoi diagram of the terrain. In this framework, we use the single approach of implementing a graph search algorithm on a graph, called the *navigational course*. We deal with a circular robot as opposed to the point robot of earlier works. Moreover, this method has an advantage of keeping the robot as far away from the obstacles as possible. This aspect seems very important in practical implementations as the earlier methods, based on the visibility graph methods, may require that the robot navigate along the obstacle boundaries. Additionally, the proposed method results in a storage complexity of $O(N)$ as opposed to $O(N^2)$ of visibility graph based methods [7,10]. Also, this method results in a path-planning complexity of $O(N^2\sqrt{\log N})$, whereas the visibility graph method has a complexity of $O(N^3)$ for the same. In this paper, we present briefly present our results and the details can be found in our report [9].

The organization of the paper is as follows: The basic framework of our solution is outlined in Section 2. In Section 3, we present the definition and properties of the navigation course to be used for navigational purposes. In Section 4, we first present solutions for the visit problem, and the terrain model acquisition problem.

2. BASIC ALGORITHM

We first describe the problem scenario and then present the basic algorithm used in the solution of the visit problem and the terrain model acquisition problem.

Terrain: We consider a finite-sized two-dimensional *terrain* populated by a finite set $O=\{O_1,O_2, \cdots O_n\}$ (n is finite) of simple disjoint polygons, called the *obstacles*. Each obstacle O_i has a finite number of vertices. The terrain in completely *unknown* to R, i.e., the number of obstacles, and also the number and locations of vertices of each obstacle are unknown to R. The free-space is given by $\Omega=\bigcap_{i=1}^{n} O_i^C$, where O_i^C is the complement of O_i in the plane. The closure of the free-space is denoted by $\overline{\Omega}$. Let N denote the total number of vertices of all obstacles. Let $VER(O_i)$ denote the set of vertices of O_i.

Robot: We consider a circular body R of radius δ, $(\delta \geq 0)$. The location of the center of R is called the *position* of R. We treat R as an open disc of radius δ centered at the position of R. R houses a computational device with storage capability. Also, R is capable of moving along a straight-line path or a curved path of second degree (in each case the path is specified). R takes a finite amount of time to move through a finite amount of distance. Further, R is equipped with an algorithm B that plans a collision-free path (for R) through a known terrain. In particular, we can use a suitable algorithm from [11] for this purpose.

Sensor System: Let x be a position of R. A point $y \in \overline{\Omega}$ is said to be *visible* to R if the straight line joining x and y is entirely contained in $\overline{\Omega}$. R is equipped with a sensor that detects the maximal set of points on the obstacle boundaries that are visible from its present location. Such an operation is termed as the *scan* operation. We assume that the scan operation is precise and error-free.

Two Navigational Problems

Initially, R is located at a point d_0 without intersecting any obstacle polygons and at a finite distance from an obstacle. In the *terrain model acquisition problem,* R is required to acquire the model of the terrain to a degree such that it can navigate to any reachable destination location by planning a path using the known terrain algorithm B alone. If a destination position is not reachable then R should report this fact without performing sensor operations. Note that after the terrain model is completely acquired, no sensor operations are needed for navigational purposes. Second, in the *visit problem,* R is required to visit the points $d_1, d_2, \cdots, d_N$ in the specified order if there exists a path through these points. If no such path exists, then R must report this fact in a finite amount of time.

Navigation strategy

We now present the algorithm *NAV* which is the basic underlying strategy used by R to solve the visit problem and the terrain model acquisition problem. Here, R performs a "graph exploration type" of navigation using a combinatorial graph called the *navigation course,* $\xi(O)$, of the terrain O. $\xi(O)$ is a 1-skeleton embedded in Ω. The nodes (edges) of $\xi(O)$ are called ξ-nodes (ξ-edges). Each ξ-node specifies a collision-free position for R, i.e. a position for R such that it is entirely contained in Ω. An edge that joins two x-nodes v_1 and v_2 specifies a collision-free path, of finite length, from v_1 to v_2 for R. The $\xi(O)$ is initially unknown and it is incrementally constructed using the data obtained through the sensor operations. The algorithm *NAV* is given below:

Consider the execution of *NAV* by R. *NAV* in initiated with a ξ-vertex $v = v_0$ and $S_2 = \{v_0\}$. The set S_1 contains all the ξ-vertices that are visited by R. The set S_2 contains all the ξ-vertices that not visited by R, but each $v \in S_2$ is adjacent to some ξ-vertex in S_1. R keeps visiting new ξ-vertices until the set S_2 becomes empty. When R visits v for the first time the adjacency list of v is computed. In this way, the $\xi(O)$ is incrementally constructed. The scan operation of line 1 and the computational operations in lines 2-7 and 11-12 can be directly executed by R. The path planning of line 8 involves finding a graph path from v to v^*. R actually moves along the edges of the computed path in line 9.

S_1 forms a connected (graph) component with the edge set being the set of all edges that are traversed by R. Further $S_1 \cup S_2$ forms a connected (graph) component with the edge set being the union of the set of all edges traversed by R and the set of all edges computed by R. Thus there exists a path along the edges (of the component) from any vertex of S_1 to any vertex of S_2. In each step, R, located at v, selects a $v^* \in S_2$, and then it moves to v^*. In this aspect, *NAV* is similar to a standard graph exploration algorithm except for one difference. In a graph algorithm the cost associated with accessing the node v^* (after it is chosen) is a single memory access. In *NAV*, when R accesses v^* there are two associated costs: (a) the computational cost of planning a path from v to v^* (b) the cost of moving R along the computed path.

```
algorithm NAV(v);
  begin
1.  perform a scan operation from $v$;
2.  mark $v$ as visited and delete it from $S_2$ and append to $S_1$;
3.  compute the adjacency list of $v$;
4.  append to $S_2$ all neighbors of $v$ that are not visited;
5.  if ($S_2$ is not empty)
6.  then
7.     select $v^* \in S_2$;
8.     plan a path from $v$ to $v^*$;
9.     move to $v^*$;
10.    $NAV(v^*)$;
11.  else
12.    return to start vertex $v_0$;
    endif
  end;
```

In order to yield correct solutions to the visit problem and the terrain model acquisition problem, the navigational course $\xi(O)$ has to satisfy a set of properties. These properties for the proposed $\xi(O)$ are discussed in detail in the next section. Suppose that $\xi(O)$ satisfies the property of *local-constructibility*, i.e., the adjacency list of a ξ-vertex v can be computed from the information obtained by a scan operation performed from v. Further, suppose that $\xi(O)$ satisfies *finiteness* property, i.e., $\xi(O)$ has finite number of vertices. Also let $\xi(O)$ satisfy graph *connectivity* property, i.e., any two ξ-vertices are connected by a path of ξ-edges. Then we have the following observation.

Observation 1: *If $\xi(O)$ satisfies the properties of finiteness, connectivity and local-constructibility, then, R, executing the algorithm NAV, visits all vertices of $\xi(O)$ in a finite amount of time.* □

3. THE NAVIGATIONAL COURSE

We first present a structure that yields a navigational course to be used by a point robot. We then extend our discussion to a circular robot.

3.1. Point Robot

For $x \in \Omega$, we define $Near(x)$ as the set of points that belong to the boundaries of obstacles O_i, $i=1,2,\cdots,n$ and are closest to x. The *Voronoi diagram*, $Vor(O)$, of the terrain populated by O is the set of points:

$$\{x \in \Omega | Near(x) \text{ contains more than one point }\}$$

In this case, $Vor(O)$ is a union of $O(N)$ straight lines and parabolic arcs (see [4,5] for more details). Each of this line or parabolic arc is referred to as *V-edge*. The points at which the edges meet are called *V-vertices*. Furthermore, $Vor(O)$ can be specified as a combinatorial graph in which each edge is labeled with two end V-vertices, and an equation defining it as a curve in the plane. Each V-vertex is labeled with its coordinates.

Consider the convex hull $C(O)$ of union of vertices of all obstacles (i.e. convex hull of $\bigcup_{i=1}^{n} VER(O_i)$). Let $E(O)$ denote the polygonal region obtained by pushing the edges of $C(O)$ out-

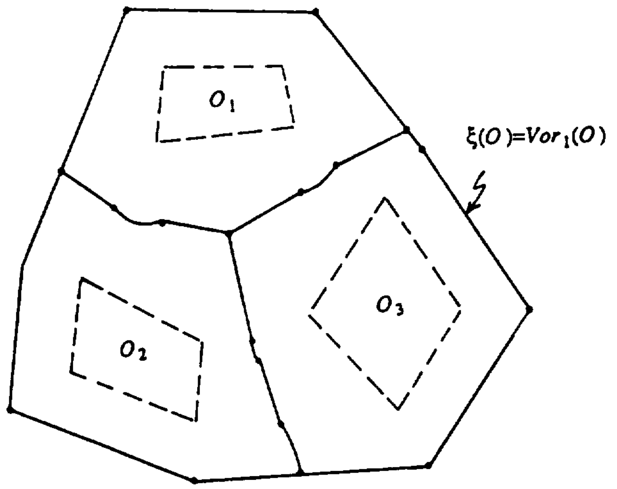


Figure 1. The $Vor_1(O)$ for the terrain $O=\{O_1,O_2,O_3\}$.

wards by a distance of s and taking the interior of 'grown' region. Let us define $Vor_1(O)=(Vor(\Omega)\cap E(O))\cup\partial E(O)$, where $\partial E(O)$ is the boundary of $E(O)$. We note that $Vor_1(O)$ precisely contains the Voronoi diagram of O that lies inside $E(O)$ and the boundary of $E(O)$. The edges (vertices) of $Vor_1(O)$ are called V_1-edges (V_1-vertices). See Fig. 1 for an example. The set of vertices of $Vor_1(O)$ is the union of V-vertices, vertices of the envelop $E(O)$ and intersection points of edges of $\partial E(O)$ with V-edges. Similarly the set of edges of $Vor_1(O)$ is the union of edges of $Vor(O)$ that are contained in $E(O)$ and the edges of $\partial E(O)$. It is easy to see $Vor_1(O)$ as a planar graph formed by V_1-vertices and V_1-edges. The set of all V_1-vertices that are adjacent to a V_1-vertex v constitute the set of *neighbors* of v. The following four basic properties of $Vor_1(O)$ are shown in [9]:

(i)*Combinatorial properties*: The number of V_1-vertices is at most $4N-n-2$, and the number of V_1-edges is at most $6N-3n-3$.

(ii)*Connectivity*: $Vor_1(O)$ is topologically connected, and consequently $Vor_1(O)$ is graph connected when viewed as a combinatorial graph i.e., there exist a path along V_1-edges between any two V_1-vertices.

(iii)*Terrain-visibility*: Every point in the closure of free-space Ω is visible from some V_1-vertex, i.e., for $x\in\overline{\Omega}$, there exist a V_1-vertex v such that the line joining x and v is entirely contained in $\overline{\Omega}$.

(iv) *Local-constructibility*: All the neighbors of a V_1-vertex v can be correctly computed from the terrain boundary information obtained by performing a scan operation at v.

3.2. Circular Robot

For $x\in\Omega$, let $Clearance(x)$ denote the distance of x from a member of $Near(x)$ (in terms of the Euclidean distance). Consider $Vor_1(O)$ such that the distance s used in obtaining $E(O)$ is at least δ. Let us consider a subset of $Vor_1(O)$ given by $\{x\in Vor_1(O) \mid Clearance(x)>\delta\}$ which is the set of points of $Vor_1(O)$ with clearance greater than δ. This set consists of a set of connected components. Initially, let R be located at $d_0\in\Omega$. Let $Vor^*{}_1(O)$ be the connected component that contains $\text{Im}(d_0)$, i.e., $Im(d_0)\in Vor^*{}_1(O)$. $Vor^*{}_1(O)$ contains either all or none of the edges of $E(O)$. Further an edge of $Vor^*{}_1(O)$ could be a truncated version of an edge of $Vor(O)$, in which case we attach a vertex at the truncated end. These vertices are called *truncated vertices*. The edge formed as a result is called the *truncated edge*. We now summarize the properties of $Vor^*{}_1(O)$.

Properties 2: *$Vor^*{}_1(O)$ satisfies the properties of finiteness, connectedness, terrain-visibility and local-constructibility.*

4. NAVIGATION ALGORITHMS

We first discuss the navigational course and the navigation strategy used by R. Then we present the algorithm *ACQUIRE* that solves the terrain model acquisition problem. We then present the algorithm *LNAV* that navigates R from d_i to d_{i+1} if a path exists from d_i to d_{i+1}. Then we obtain the algorithm *GNAV* that solves the visit problem. *GNAV* uses *LNAV* as a component and also incorporates the feature of incidental learning.

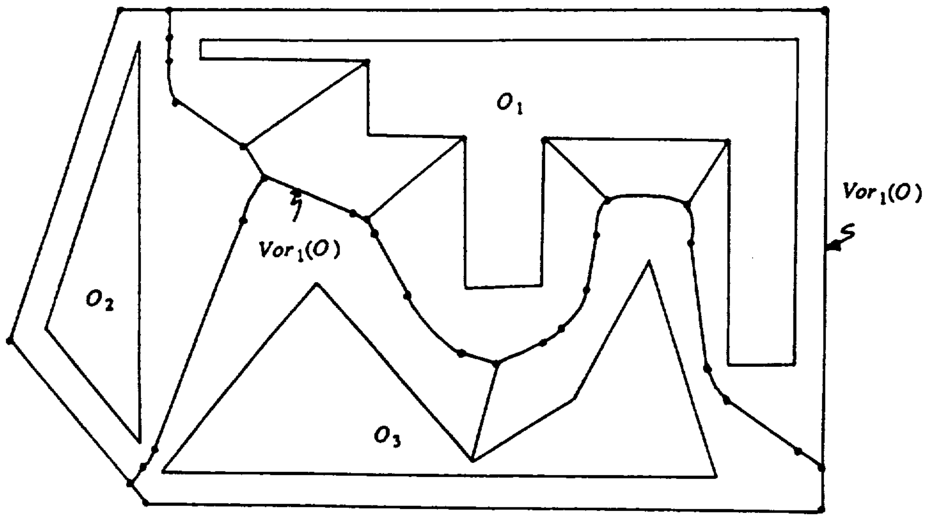


Figure 2. The terrain $O=\{O_1,O_2,O_3\}$ and $Vor_1(O)$.

4.1. Preliminaries

For a point robot, $\xi(O)$ is obtained by deleting from $Vor_1(O)$ all the V_1-edges that terminate on a concave corner. Such edges are formed by two obstacle edges that meet at a concave corner. For a circular robot $\xi(O)$ is obtained by deleting from $Vor^*{}_1(O)$ all the truncated edges. In Fig. 2, we show the terrain $O=\{O_1,O_2,O_3\}$, and the corresponding $Vor_1(O)$. In Fig. 3, we show $\xi(O)$ for a circular robot. Note that every V-edge that terminates at a concave corner generates a truncated edge. We assume that the process of deletion of an edge retains the vertex that connects the edge to the rest of $Vor_1(O)$ or $Vor^*{}_1(O)$. Now, view $\xi(O)$ as 1-skeleton embedded in the plane. It is clear from the definition that any point on $\xi(O)$ - in particular a ξ-vertex - specifies a collision-free position for R. Consequently, a ξ-edge specifies a collision-free path for R. It is direct to see that $\xi(O)$ satisfies the properties of finiteness and local-constructibility. The connectivity of $\xi(O)$ can be shown by observing that each edge that is removed from $Vor_1(O)$ and $Vor^*{}_1(O)$ is pendant and can not disconnect resultant set. Let C denote the number of concave corners of the terrain O. Note that we delete (at least) C V-edges and V-vertices from $Vor_1(O)$ ($Vor^*{}_1(O)$) for a point (circular) robot. Then we have the following properties.

Properties 3: $\xi(O)$ *for a point or a circular robot satisfies the properties of finiteness, connectivity, terrain-visibility and local-constructibility. Further more*

(i) #ξ-vertices $\leq 4N-n-C-2$

(ii) #ξ-edges $\leq 6N-3n-C-6$. □

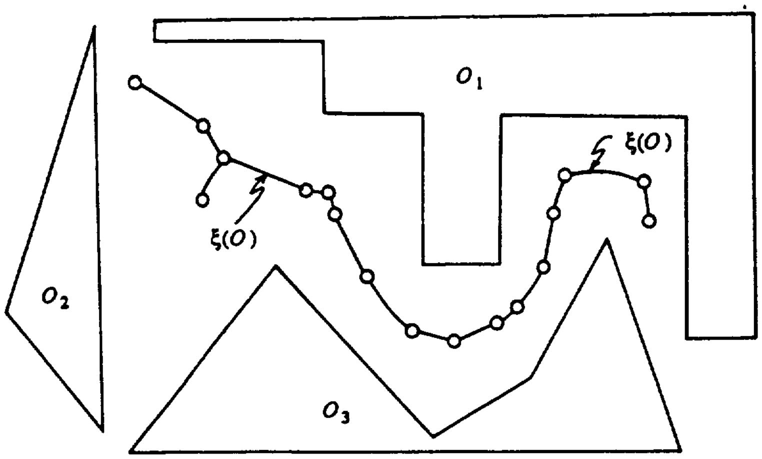


Figure 3. ξ(O) for a circular robot for O of Fig. 2.

Now consider the execution of the algorithm *NAV*. For ease of presentation, we discuss a depth-first implementation of *NAV* which specifies a particular way to select v^* the ξ-vertex to be visited next (line 7). R is presently located at the ξ-vertex v. If v has neighbors that are not visited then R chooses one of the unvisited neighbors as v^*. If all neighbors of v_* are visited then R chooses the vertex of S_2 that is reachable by a minimal distance path. This path is obtained by invoking one-to-all shortest path algorithm of [2] on the presently available $\xi(O)$ and picking the vertex that is reachable from v by a path of minimal length. Note that $\xi(O)$ is a planar graph. Cost of this computation is $O(N\sqrt{\log N})$.

4.2. Terrain Model Acquisition

The algorithm *ACQUIRE* is a direct implementation of the algorithm *NAV*. Once the terrain model is available one can use the algorithm B to plan a path to reach any destination point. This algorithm has a time complexity of $O(N\log N)$ [11]. Thus we have the following theorem.

Theorem 1: *The algorithm ACQUIRE solves the terrain model acquisition problem in a finite amount of time such that*

(i) *The number of scan operations performed is at most* $4N-n-C-2$.

(ii) *The total distance traversed by R while executing ACQUIRE is at most twice the total length of the depth-first tree of* $\xi(O)$ *rooted at* v_0.

After the execution of ACQUIRE, R can navigate to any reachable destination with a time complexity of $O(N\log N)$ *and with no sensor operations.* □

In our implementation we use the adjacency list representation of $\xi(O)$. We store the coordinates of each ξ-vertex in the adjacency lists. We maintain a table called MAP-TABLE as an AVL tree. The cost of this operation is $O(\log N)$ using the table.

Theorem 2: *The complexities of various tasks carried out by ACQUIRE are as follows:*

(i) *the storage complexity is* $O(N)$,

(ii) *cost of construction of* $\xi(O)$ *is* $O(N^2 \log N)$

(iii) *total cost of path planning is* $O(N^2\sqrt{\log N})$,

(iv) *cost of construction of MAP-TABLE is* $O(N\log N)$, *and total cost of accesses to MAP-TABLE is* $O(N\log N)$. □

4.3. Visit Problem

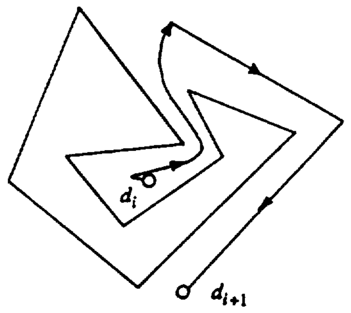
We now discuss the algorithm *LNAV* that navigates R from its present location at d_i to a destination point d_{i+1} if such path exists. If there is no path from d_i to d_{i+1}, then R will declare the same in a finite amount of time. The algorithm *LNAV* is obtained by modifying *NAV*. Initially a scan is performed from d_i and if d_{i+1} is found reachable, then R moves to d_{i+1}. If d_{i+1} is not found reachable then R computes a ξ-vertex v_0 and moves to v_0. From v_0, the algorithm *NAV* is invoked. Let R be located at v. After a scan is performed from v, R checks if d_{i+1} is reachable. If d_{i+1} is reachable, then R moves to d_{i+1} and terminates *NAV*. If not, R continues to execute *NAV* until completion. If d_{i+1} is not found after S_2 becomes empty then d_i is declared as not reachable.

Theorem 3: *Algorithm LNAV navigates R from d_i to d_{i+1} in a finite amount of time if the latter is reachable. If d_{i+1} is not reachable then R declares so in a finite amount of time. In executing the algorithm LNAV by R,*

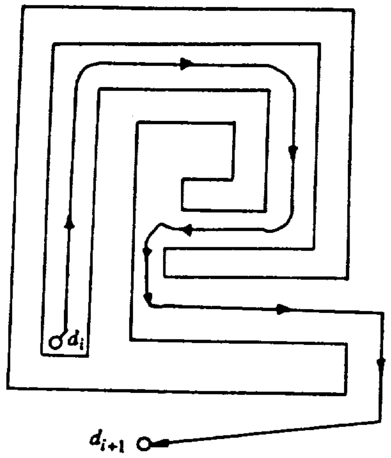
(i) *the number of scan operations is at most* $4N-n-C-2$,

(ii) *the total distance traversed is at most equal to twice the length of the depth first tree of* $\xi(O)$ *rooted at* v_0. □

The computational complexity of executing the algorithm *LNAV* follows along the lines of previous section. Thus, in executing the algorithm *LNAV*, (i) the storage required is $O(N)$, (ii) cost of construction of $\xi(O)$ is $O(N^2\log N)$, (iii) complexity of path planning is $O(N^2\sqrt{\log N})$ (iv) the cost of construction of MAP-TABLE is $O(N\log N)$, and the total cost of accesses to MAP-TABLE is $O(N\log N)$.



(a) R escaping out of a concavity



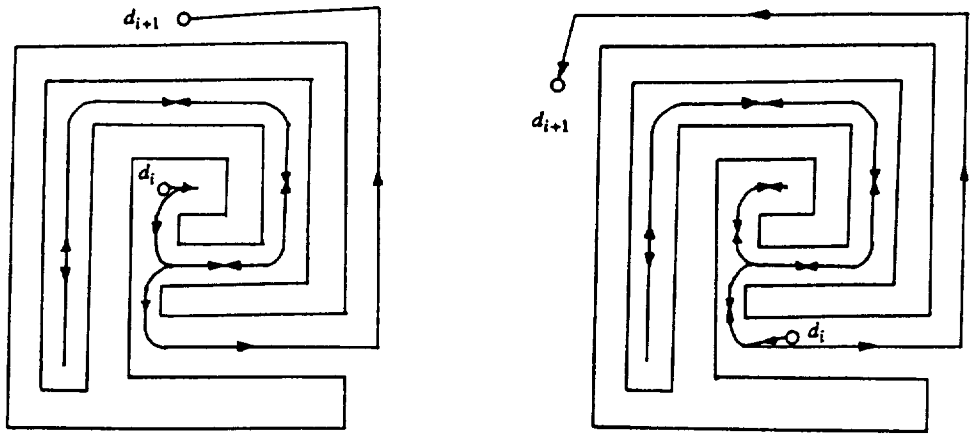
(b) R moving out of a maze

Figure 4. Execution of LNAV by R

In Fig. 4 we show a point robot moving out of a concavity, and moving out of a maze. In Fig. 5 we show a point robot moving out of a maze with backtracking. We can solve the visit problem by a repeated invocation of *LNAV*. *LNAV* completely relies on the sensor information for navigation.

Since the sensor obtains only a partial information about the terrain, as a result R might navigate into local concavities as in Fig. 5. If R is required to navigate in the regions that it navigated in previous traversals, then it can use the previous information to plan its present course of navigation. Note that the partial model of the terrain depends on the paths traversed by R in earlier traversals. We now obtain the algorithm *GNAV* as follows: We store the adjacency lists computed by R over the traversals in a global $\xi(O)$. Further the set S_2 is also stored over the traversals. Consider the navigation from d_i to d_{i+1}. Then *GNAV* computes a ξ-vertex that is reachable from d_i and moves to this vertex. Then R computes a ξ-vertex d^* that is closest to d_{i+1} according to some criterion such as distance. Then R moves along a path on $\xi(O)$ to d^*. From d^*, R uses *LNAV* to navigate to d_{i+1}. It is direct to see that *GNAV* correctly solves the visit problem. Moreover, R checks the set S_2 after every scan operation. After S_2 becomes empty, R switches-off its sensor and navigates the further traversals using the algorithm B alone. At this stage R has acquired the terrain model that is sufficient to navigate to any reachable point. Thus after this stage R does not perform scan operations for the purpose of navigation, and also R would avoid local concavities. Using the arguments of previous section it is clear that such stage will be reached after at most $4N+M-n-C-1$ scan operations. Thus we have the following theorem.

Theorem 4: *The terrain model will be completely built by R in at most* $4N+M-n-C-1$ *scans, then the execution of each traversal involves no scan operations with a time complexity of* $O(N\log N)$ □



(a) R backtracks once **(b) R backtracks twice**

Figure 5. Execution of LNAV by R.

Since the process by which R acquires the terrain is incidental, i.e., depends on the previous traversals it executed, it is possible to make probabilistic statements about the performance of *GNAV*. Let $\xi(O)=(V,E)$. Let p_v (>0) be the probability that R visits a ξ-vertex during any traversal. Probability that a scan is performed from v in ith traversal is $(1-p_v)^{i-1}p_v$. Then the probability that the terrain model will be complete during the mission of M traversals if $\prod_{v\in V}[1-(1-p_v)^M]$ which is non-zero. Moreover this probability approaches to 1 as M approaches infinity. Thus in a limiting case R obtains the complete terrain model with a probability of one.

5. CONCLUSIONS

We presented an algorithmic framework based on a retraction of free-space to solve two navigational problems for a circular robot moving in an unknown terrain. We consider the visit problem in which the robot is required to visit a sequence of destination points. We present an algorithm that enables the robot to visit the destination points using an ideal sensor, and also build the terrain model in the regions it navigates. Further the robot can detect the completion of the terrain model, and at this stage it switches to a known terrains navigation algorithm. After this stage, the future navigation is carried out without using the sensor. We also consider the terrain model acquisition problem wherein the robot is required to autonomously explore the terrain and build a model of the terrain such that the future navigation to any reachable destination can be carried out using the algorithms of known terrains alone.

References

[1] Abelson, H., and A. diSessa, (1980), "Turtle Geometry", MIT Press, 1980, pp. 179-199.

[2] G.N. Frederickson, Shortest path problems in planar graphs, Proc. 24th Ann. Symp. on Found. of Comput. Sci., 1983, pp. 242-247.

[3] Iyengar, S.S., C.C. Jorgensen, S.V.N. Rao and C.R. Weisbin, (1986), Robot navigation algorithms using learned spatial graphs, *Robotica*, vol. 4, January 1986, pp. 93-100.

[4] Kirkpatrick, D.G., (1979), Efficient computation of continuous skeletons, Proc. 20th Ann. Symp. on Found. of Comput. Sci., 1979, pp. 18-27.

[5] Lee, D.T. and S. Drysdale, (1981), Generalization of Voronoi diagrams in the plane, *SIAM J. Comput.*, vol.10, no.1, 1981, pp. 73-87.

[6] Lumelsky, V.J. and A.A. Stepanov, (1987) Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape, *Algorithmica*, vol.2, 1987, pp. 403-430.

[7] Oommen, J.B., S.S. Iyengar, N.S.V. Rao and R.L. Kashyap, (1987) Robot navigation in unknown terrains using visibility graphs. Part I:The disjoint convex obstacle case, *IEEE J. Robotics and Automation*, vol. RA-12, 1987, pp. 672-681.

[8] Rao, N.S.V., S.S. Iyengar and G. deSaussure, (1988a), The visit problem: Visibility graph-based solution, Proc. 1988 IEEE Int. Conf. Robotics and Automation, pp. 1650-1655.

[9] Rao, N.S.V., N. Stoltzfus and S.S. Iyengar, (1988b), A 'retraction' method for learned navigation in unknown terrains for a circular robot, Tech. Rep. #88-018, Dept. of Computer Science, Old Dominion University, 1988.

[10] Rao, N.S.V., S.S. Iyengar, J.B. Oommen and R.L. Kashyap, (1988c), Terrain acquisition by a point robot amidst polyhedral obstacles, *IEEE J. Robotics and Automation*, vol. 4, No. 4, 1988, pp. 450-455.

[11] Yap, C.K. (1987), Algorithmic motion planning, *in* "Advances in Robotics: Vol. 1: Algorithmic and Geometric Aspects of Robotics", Ed. J.T. Schwartz and C.K. Yap, Lawrence Erlbaum Associated Pub., Hillsdale, New Jersey, pp. 95-144.

NEURAL NETWORKS