CONCURRENT ALGORITHMS FOR AUTONOMOUS ROBOT NAVIGATION IN AN UNEXPLORED TERRAIN

S.V. Nageswara Rao *, S.S. Iyengar*, C.C. Jorgensen** and C.R. Weisbin**

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

**Center for Engineering Systems Advanced Research
Engineering Physics and Mathematics division
Oak Ridge National Laboratory
Oak Ridge, Tennessee 37831, USA

ABSTRACT

Navigation planning is one of the most vital aspects of an autonomous mobile robot. The problem of navigation in a completely known obstacle terrain is solved in many cases. Comparatively less number of research results are reported in literature about robot navigation in a completely unknown obstacle terrain. In recent times, this problem is solved by imparting the *learning* capability to the robot. The robot explores the obstacles terrain using sensors and incrementally builds the terrain model. As the robot keeps navigating, the terrain model becomes more learned and the usage of sensors is reduced. The navigation paths are computed by making use of the existing terrain model. The navigation paths gradually approach global optimality as the learning proceeds. In this paper, we present concurrent algorithms for an autonomous robot navigation in an unexplored terrain. These concurrent algorithms are proven to be free from deadlocks and starvation. The performance of the concurrent algorithms is analyzed in terms of the planning time, travel time, scanning time, and update time. The analysis reveals the need for an efficient data structure for the obstacle terrain in order to reduce the navigation time of the robot, and also to incorporate learning. The modified adjacency list is proposed as a data structure for the spatial graph that represents the obstacle terrain. The time complexities of various algorithms that access, maintain, and update the spatial graph are estimated, and the effectiveness of the the implementation is illustrated.

1. INTRODUCTION

Robotics is one of the most actively researched areas of computer science. It is replete with issues ranging from abstract mathematical problems to highly pragmatic ones. In many applications that involve monotonous and tedious tasks, (e.g. normal maintenance or inspection) it would be desirable to employ robots. In addition, hazardous environments such as the ocean, nuclear reactors, battlefields, etc. require operations that might be safely and efficiently carried out by autonomous mobile robots. Tasks requiring rapid responses in emergency situations are also appropriate for intelligent machines. An autonomous mobile robot may be characterized as a machine capable of motion planning, execution and learning. There have been numerous efforts to design automated mobile robots. Examples are SHAKEY of Nilsson [18], the JPL robot of Thompson [19], HILARE of Giralt et al [8], the CMU Rover of Moravec [17], and HERMIES of Weisbin et al [22], etc. Some of the most important research areas in robotics are knowledge representation, task planning, sensor interpretation, dynamics and control, architectures for robot computer systems, algorithms for concurrent computation, coordinated manipulation and navigation, etc.

One of the key problems in the design of an autonomous mobile robot is the navigation planning. The problem of navigation planning in a 'known' terrain involves finding collision-free (possibly optimal) paths through a terrain that is arbitrarily populated with obstacles. This problem has been the focus of much research in recent times, and has been solved in many cases. For a broader treatment on this see [2,3,6,9,14,15,21]. The techniques for navigation described in these papers generally assume that a complete global model of the obstacle laden environment is known. Most of these techniques model the obstacles and the free space as precise mathematical and geometric entities. For a robot navigating in a new or unexplored terrain, these techniques are not directly applicable or extendible. There has not been as much work reported in the literature with respect to robot navigation in an unexplored terrain. This can be attributed at least in part to the lack of global information about the obstacles and their locations. This makes the global optimality of the collision-free paths difficult to achieve. Many of the existing solutions to this problem are based on sensor information [5,8,17,19], and in general do not achieve global optimality for the navigation paths. Recently, Iyengar et al [10,11] have developed a technique for navigation planning. This technique is based on learning and requires no initial terrain model. The terrain model is gradually built by consolidating the information about the obstacles as newer paths are traversed. The global optimality of the paths is gradually achieved as the learning proceeds.

In this paper, we shall discuss concurrent navigation algorithms well-suited for implementation in the forthcoming generation of intelligent mobile robots. These concurrent algorithms implement the various activities of an autonomous mobile robot in a well coordinated manner. An efficient implementation of these algorithms calls for a data structure to store the obstacle terrain. This data structure should guarantee efficient access in implementing the path planning and learning activities of the robot. We propose and analyze a 'modified adjacency list' data structure for the terrain model.

The organization of the paper is as follows: Section 2 reviews navigation by learned spatial graph techniques. Section 3 develops the concurrent algorithms for robot navigation corresponding to methods proposed in section 2. In section 4, the performance of the concurrent algorithms is analyzed. Section 5 describes an abstract data structure for the terrain model. Included in this section are the implementation and analysis of the proposed data structure.

2. THE NAVIGATION TECHNIQUE

The robot navigation problem considered in this paper can be defined as follows: Initially, the robot is placed in an unexplored terrain that is arbitrarily cluttered with obstacles. The robot is required to autonomously perform a number of goal directed traversals. Only the gross platform motion in two dimensions is considered. Without loss of generality the robot is assumed to be a point in the plane formed by the obstacle terrain. This is not a severe restriction for the method, as path planning for a finite sized robot involves 'enlarging' the obstacle-size to account for the actual robot dimension. as described by Lozano-Perez and Wesley [15]. Navigation in an unex-

plored terrain is significantly different from the problem addressed by Brooks [2], Brooks and Lozano-Perez [3], Crowley [6], Gouzenes [9], Lozano-Perez[14], Lozano-Perez and Wesley [15], Moravec [17], and Udupa [21], as no initial terrain model is available in our case. In the technique developed by Iyengar et al [10,11], the terrain model is gradually built by the robot as it traverses newer paths. At any intermediate point of time, the partially built terrain model is used in planning the required path of navigation. The terrain model is updated by integrating the sensor information obtained during the execution of current traversal. As a result of this incremental learning, the global optimality of navigation paths is gradually approached. In this respect, the approach of [10,11] is also different from the sensor based methods of Moravec [17], Thompson [19], Giralt, Sobek, and Chatila [8], and Chattergy [5] which are not explicitly directed towards global optimality in navigation planning.

The capability to learn about the obstacle terrain is vital to an autonomous mobile robot navigating in an unexplored or partially explored terrain. Crowley [6], Laumond [12], and Turchen and Wong [20] use different forms of learning in the design of robot systems. In this section, we summarize the robot navigation method of Iyengar et al [10,11] which is based on a different implementation of learning. In [10,11], learing is incidental meaning that the robot explores only the regions that lie on the paths of navigation. The process of navigation operates in two basic modes - *local navigation* and *global navigation*. The obstacles are avoided in a localized manner using the sensor information in the local navigation mode. The global navigation mode consists of two components: (a) path planning using the partially built terrain model, (b) learning by integrating the information extracted from sensor readings. Initially, the paths are planned and traversed in local navigation mode based on the sensor readings only. These paths of navigation partition the obstacle terrain into a set of polygons. In the global navigation mode these polygons are accessed and manipulated in path planning and learning. The learning incorporated in this method enables the paths to approach global optimality as as the robot makes succesive journeys. This is a very significant factor in applications wherein the terrain model is completely unknown or only partially known. Generally, in such applications the sensor based approahes are followed for path planning [8,17,19]. But, the approach of [10,11] is more efficient than the pure sensor based approaches in a general case, because as the navigation continues (a) sensor is used to a lesser extent, (b) the paths approach global optimality. In the remainder of this section we briefly discuss the navigation technique of [10,11], and more detailed treatment can be found in those papers.

In the local navigation mode, the robot scans the obstacle terrain around the line $\vec{SD}$ joining the source point S to the destination point D. Then we compute two points of inflection (on either side of the line $\vec{SD}$) such that the scanner view is blocked by an obstacle within these two points. The robot travels to one of the two points in a such way that the distance traversed in a direction perpendicular to and opposite to $\vec{SD}$ is minimized. The same strategy is applied recursively from this intermediate point. The paths traversed in this mode are optimal only in a localized manner in terms of the distance traversed by the robot. In general this technique is not guaranteed to yeild a globally optimal path.

The initial paths are traversed in local navigation mode, and these paths partition the obstacle terrain into a set of polygons. Thus the two-dimensional plane of the obstacle terrain is represented as set of non-intersecting polygons that cover the entire navigation area. The edges of the polygons correspond to the paths previously traversed by the robot. A *free-polygon* represents an obstacle-free region. An *unexplored-polygon* represents a region whose interior is not explored by the sensor. A polygon p is an *obstacle-polygon* with respect to v, a vertex or a point on the edge, if the entire scan range of the sensor (inside p) is obstructed by the obstacle(s) contained in p when the sensor is located at vertex v. A traversal from the source point S to the destination point D consists of a series of stop-points; in between two adjacent stop-points the robot travels in straight lines. For a given source point S and destination point D, we find S^* and D^* to be the vertices of polygons nearest to S and D respectively. The navigation from S to S^* and from D^* to D is carried out in the local navigation mode. The navigation from S^* to D^* is carried out in the global navigation mode.

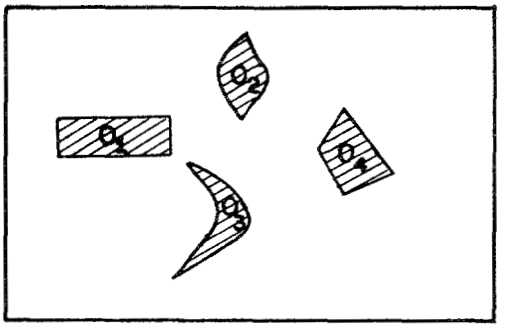


Fig. 1 Unexplored obstacle terrain

The global navigation mode can be described as follows: Let p be a polygon with S^* as a vertex and containing the end portion of the line $S^*\vec{D}^*$ towards S^*. We call this polygon to be the *near-polygon* of $S^*\vec{D}^*$. If p is a free-polygon then the robot directly traverses to the intersection point X of p with the line $S^*\vec{D}^*$. If p is an obstacle-polygon then the edges of p are accessed in the clockwise direction, starting from S^*, to obtain a point X such that the near-polygon of $X\vec{D}^*$ is different from the near-polygon of $S^*\vec{D}^*$. Such point is computed treversing in the anti-clockwise directions from S^* along the edges of p. Among the two points computed, the point nearest to S^* is chosen as Y. Then navigation from Y to D^* is recursively carried out in global navigation mode. If p is an unexplored-polygon then its interior is scanned from the vertex S^*. Based on the sensor information p is decomposed into obstacle-polygons and obstacle free regions. The obstacle regions are decomposed into free-polygons. Then adjacent free-polygons are merged to form bigger free-polygons. After this decomposition process the new near-polygon of $S^*\vec{D}^*$ is either an obstacle-polygon or a free-polygon, and the cases described above are applicable.

A summary of the results of this navigation technique is as follows (see [10,11] for details) : As learning proceeds,

a) Capability for efficient navigation planning evolves from local optimality to global optimality.

b) The polygons that bound obstacles shrink in area and as a result enclose the obstacles more tightly.

c) The free-polygons are generated to be convex and they grow in size.

d) The frequency of taking sensor readings decreases.

e) The problem solution becomes computational instead of sensor based.

We now illustrate the navigation technique using an example of rectangular terrain. Fig. 1 shows an unexplored terrain containing four obstacles O_1,O_2,O_3,and O_4. Consider the navigation of the robot from S_1 to D_4 in local navigation mode. Seven traversals are undertaken from the source points S_1,D_1,S_2,D_2,S_3,D_3,and S_4 to the

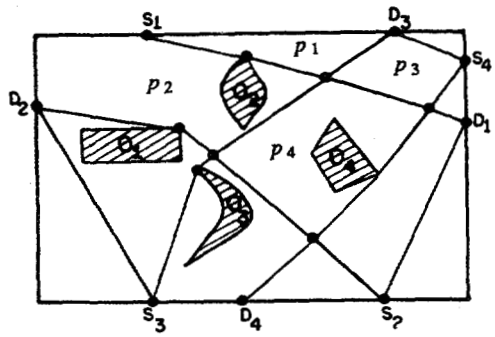


Fig. 2 Obstacle terrain explored in local navigation mode

destination points D_1,S_2,D_2,S_3,D_3,S_4, and D_4 respectively. As a result, the obstacle terrain is partitioned into a set of polygons as shown in Fig. 2. The polygons formed until this stage are designated as unexplored-polygons. Consider the navigation from S to D in the global navigation mode. The robot travels from S to S^* in local navigation mode. The polygon p_2 of Fig.2 is is scanned from S^* and decomposed into free-polygons p_{21},p_{23} and obstacle-polygon p_{22}. Then robot traverses to S_5 and explores the region p_4 of Fig.2. As a result, p_4 is decomposed into the free-polygons p_{41},p_{43} and obstacle-polygon p_{42} as shown in fig.3. Then the robot traverses directly across the free-polygon p_{41} to D^*. Then D is reached in the local navigation mode. The exact path of navigation is denoted by dotted lines in Fig.3. Observe that the obstacle O_2 and O_4 are more

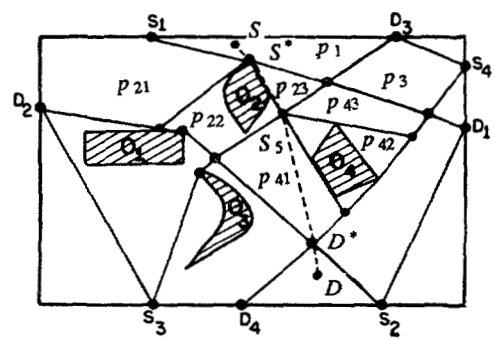


Fig. 3 Navigation from S to D in global navigation mode

tightly bounded by the polygons after this traversal. Any traversal across the polygons p_3 and p_{43} will combine them to form a single large free-polygon. For more details on this method see [10,11]. In the following section, we propose a model of concurrent computation for this robot navigation system.

3. CONCURRENT PROCESS MODEL FOR ROBOT NAVIGATION

The navigation of an autonomous robot is determined by various mechanical and control operations such as moving, sensing, stopping, starting, etc. The computer system for the robot should coordinate all these operations, apart from carrying out the computations. A close inspection of various activities involved in the navigation of a robot reveals that certain constituent operations can be carried out concurrently. Exploitation of concurrency in these operations decreases the over-all journey time of any traversal. In this section, we examine the navigation process with a view to find out the exact operations that can be carried out concurrently.

The robot is assumed to have two systems, a *control* computing system and a *planning* computing system. This abstract model is analogous to robots which have an on-board computer for controlling the motion and sensor operations, and another on-board computer for carrying out planning and world modelling [10,11,22]. Though the treatment here is based on the robot HERMIES-II [10,11,22], it is equally applicable for many other robot systems. The control system moves the robot from a source point to a destination point. It operates the sensors to scan the specified regions and returns the information to the planning system. The planning system accesses the terrain model to plan the next stop points and returns them to the control system. It also incorporates learning into the obstacle terrain by integrating the information about the explored polygons. A queue is utilized by the planning system to return the stop points to the control system. A buffer is utilized by the control system to return the information about the explored polygons to the planning system. The configuration of the system is shown in Fig. 4.

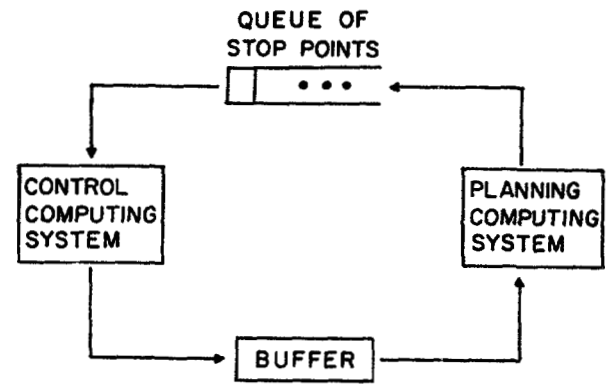


Fig. 4 Configuration of the computing system of the robot

The operation of the computer system for the robot is characterized as the concurrent processes PLANNING and CONTROL. The main function of the algorithmic processes *PLANNING* and *CONTROL* is to implement the navigation technique as a coordinated activity between the control and the planning computing systems. The planning computer computes the intermediate stop points and enters them into the queue for the control computer to pick up. If the next polygon is unexplored then process *PLANNING* process

```
process CONTROL;
// This process controls various mechanical actions //
begin
1.   while (destination is being reached) do
     begin
2.       while (queue is empty) do stop and wait;
3.       get the next entry from the queue;
4.       if (entry is * )
5.       then stop, explore and return the information to the buffer;
6.       else
         begin
7.           goto the next stop point;
8.           if (stop point has been overwritten
                    to allow continued straight line motion)
9.           then goto to the new stop point without stopping
                        and changing the direction;
         end;
     end;
end;
```

enters * into the queue as in line 16. The process *CONTROL* process picks * from the queue, scans the polygon and returns the information via the buffer as in the lines 4 and 5 of process *CONTROL*. The process *PLANNING* waits at this point as per line 17 of process *PLANNING*. The obstacle terrain model is updated in line 18 of process *PLANNING*.

```
process PLANNING;
// This process carries out the computations //
begin
1.  last-stop ← source point;
2.  while (destination is not reached) do
    begin
3.    find the near-polygon p, of the line from the
      last-stop point to the destination point;
4.    if (p is a free-polygon)
5.    then
      begin
6.      compute the next stop point, s;
7.      if (previous polygon is free-polygon )
8.      then overwrite the latest stop point with s;
9.      else enter the stop point into the queue;
      end
10.   else if ( p is an obstacle-polygon with respect to
                  the latest stop point in the queue)
11.       then
          begin
12.         compute the next stop point;
13.         compute the shorter path along edges of p;
14.         enter into the queue, in order, all the vertices
            of p on the shorter path;
          end;
15.   else
      begin
16.     enter into the queue * and the range of scan;
17.     if (buffer is empty) then wait;
18.     update the model;
19.     last-stop ← last stop point;
      end;
    end;
end;
```

The robot stops to take the corresponding sensor readings, if required, as indicated in line 5 of process *CONTROL*. As indicated in line 2 of process *CONTROL* the robot stops and waits if the next stop point is not already computed and entered into the queue (by process *PLANNING*). Since stopping and starting at the next stop-point involves a considerable amount of time, faster computation of the next stop-point would eliminate considerable time delay. However, the execution of step 7 of process *CONTROL* often involves a change in the direction of motion.

If the polygon which the robot is currently traversing and the next polygon in sequence are both free-polygons, then the robot can continue to travel straight without stopping and without a change in direction. This is possible only if the planning computer computes the next stop-point before the robot reaches the last stop-point. In the cases where it is possible, the process *PLANNING* overwrites the latest stop-point as in line 8 of process *PLANNING*. This information is utilized by process *CONTROL* to continue travel in the same direction (as given in lines 8 and 9 in process *CONTROL*). As in the earlier case, fast computation of the next stop-points is warranted, because it eliminates the significant time delays involved in the stopping, starting, and changing the direction of motion. Hence, we conclude that it is highly desirable to expedite the computation involved in finding the next stop-points. As indicated in the above algorithms, the robot stops until the model is updated, and thus the updating time directly adds to the total travel time. Hence, there is also a need for fast updating algorithms. The sensor scanning time, and the times for travel, stopping, starting and change of direction are approximately fixed for a given robot configuration. However, the steps involved in process *PLANNING* can be expedited by efficient design of the data structures and algorithms for implementation of the terrain model.

We assume that there are no closed corners in the obstacle terrain into which the robot can navigate. Stated formally, if the source and destination points lie on the opposite sides of an obstacle, the destination point can be reached by traversing around the obstacle in either direction. The obstacle terrain is finite and the robot has been navigating in the terrain for a finite amount of time. In such a situation the navigation technique described in earlier section will always terminate because the robot can always get around each of the obstacles lying on the way to the destination point. Since each path involves a finite number of polygons - and each polygon bounded by a finite number of finite sized edges, - the queue and the buffer of the computing system will also be bounded in size. Once sensor scanning is initiated by entering * into the queue, the process *PLANNING* enters a wait state. At this time the process *CONTROL* cannot be in the wait state indefinitely, as it will eventually read * and come out of wait state. After the process *CONTROL* returns the sensor data, the process *PLANNING* comes out of waiting state. Similarly, when process *CONTROL* is waiting, the process *PLANNING* will not be waiting indefinitely as it will eventually read the contents of the buffer and come out of waiting state. Hence the system should be free of deadlocks and starvation.

4. PERFORMANCE ANALYSIS

The performance of concurrent processes described in the previous section can be analyzed using four time measures, namely, the *travel-time*, *sensor-time*, *update-time*, and *plan-time* for each path from the source point S to the destination point D. Each path from S to D is characterized by the 'ordered' sequence of polygons, $S^*=<p_1,p_2,...,p_k>$, that is encountered as the robot traverses from S to D. Let s_i and d_i represent the source and destination points, respectively, corresponding to the polygon p_i. Then the path from S to D is also represented as the ordered sequence given by $<s_1,s_2,...,s_k,d_k>$, and also $s_{i+1}=d_i$, for $i=1,2,...,(k-1)$.

The *travel-time*, T_T is given by $\sum_{i=1}^{k} T_t(p_i)$, where $T_t(p_i)$ is the time taken by the robot to travel from s_i to d_i. If p_i is a free-polygon, then $T_t(p_i)$ is the time taken by the robot to travel from s_i straight to d_i. If p_i is an obstacle-polygon, then $T_t(p_i)$ is the time taken by the robot to travel from s_i to d_i via the smaller path among the two paths from s_i to d_i along the edges of the polygon p_i. The main factor that decides T_T is the formation of various polygons, which in turn depends on the paths traversed so far.

The *sensor-time*, T_S, is given by $\sum_{i=1}^{k} T_s(p_i)$, where $T_s(p_i)$ is the time needed to scan the unexplored polygon p_i. If p_i is either a free-polygon or an obstacle-polygon with respect to s_i, then $T_s(p_i)$ is zero. The value of T_S depends on the profile and location of the various explored and unexplored polygons.

The *update-time*, T_U is given by $\sum_{i=1}^{k} T_u(p_i)$, where $T_u(p_i)$ is the time needed to update the information about the polygon p_i, based on the sensor data. As in the earlier case, $T_u(p_i)$ is zero if p_i is either a free-polygon or an obstacle-polygon with respect to s_i. If p_i is an

unexplored-polygon, then $T_u(p_i)$ includes the time needed to divide p_i into visible and invisible regions, and again to divide the visible region into obstacle-polygons and free-polygons. It also includes the time needed to merge the free-polygons to form bigger free-polygons. This factor not only depends on the profile and the location of various polygons, but also on the data structures and algorithms used for implementing the terrain model.

The *plan-time*, T_P, is given by $\sum_{i=1}^{k} T_p(p_i)$, where $T_p(p_i)$ is the time required to plan a path from s_i to d_i, given that p_i is either a free-polygon or an obstacle-polygon. If p_i is a free-polygon, then $T_s(p_i)$ is the time required to find the intersection point, d_i, of p_i with the line joining s_i to D. If p_i is an obstacle-polygon with respect to s_i, then $T_p(p_i)$ includes the cost of planning the shorter path along the edges of p_i. Like the update-time, this parameter depends on the algorithms and data structures used to implement the terrain model, as well as on the profiles and locations of various polygons.

The time taken for the robot to travel from the souce point S to the destination point D is a function of all four times described above.

THEOREM 1:

The time required by the robot to traverse from S to D is $T_p(p_1)+T_T$ in the best-case.

PROOF: In the best-case, every p_i, of the sequence $S^*=<p_1,p_2,...,p_k>$ of the path from S to D, is a free-polygon. Also, the computation of d_i takes less time than $T_t(p_{i-1})$, for $i=2,3,...,k$. Thus, the stop point d_i is computed while the robot is on it's way to s_i and is kept ready in the queue before the robot reaches s_i. Hence, the total time of travel from S to D is given by:

$$T_p(p_1)+\sum_{i=2}^{k}\max(T_p(p_i),T_t(p_{i-1}))+T_t(p_k)$$

$$=T_p(p_1)+T_T \square$$

This best-case time is almost equal to the travel time needed to merely travel from S to D.

THEOREM 2:

The time required by the robot to traverse from S to D is $T_T+T_P+T_U+T_S$ in the worst-case.

PROOF: In the worst-case, every polygon $p_i \in S^*$, is to be explored. In such a case no overlap of operations is possible, and the algorithms *CONTROL* and *PLANNING* operate in strictly mutually exclusive manner. Hence, the time total time of traversal from S to D is given by:

$$\sum_{i=1}^{k}(T_p(p_i)+T_s(p_i)+T_t(p_i)+T_u(p_i))$$

$$=T_T+T_P+T_U+T_S \square$$

In general, the actual travel time lies in between these two limits. In the initial stages of learning, the travel time is close to that in the worst-case. As learning proceeds, more and more polygons are explored and the total travel time approaches the time of the best case.

5. THE DATA STRUCTURE FOR THE TERRAIN MODEL

The performance of the concurrent processes, *CONTROL* and *PLANNING*, depends on the plan-time, travel-time, sensor-time and update-time for the various polygons encountered by the robot during its traversal. Of these factors, the travel-time and sensor-time are mainly determined by the mechanical speeds of various components of the robot system, and, in general, are not solely controlled by the model used for the obstacle terrain. However, the plan-time and update-time can be controlled by suitably designing the terrain model and the methods to manipulate the terrain information. As shown in the best-case analysis of the system, even for fixed values of the travel-time and sensor-time, the optimal performance can be obtained by utilizing algorithms such that the condition $\max(T_p(p_i),T_t(p_{i-1}))=T_t(p_{i-1})$ for $i=2,...,k$ is satisfied. In precise terms, we first need to obtain a good data structure for representing the various polygons. We also need to design efficient algorithms for performing the basic operations such as finding the intersection points, partitioning the polygons, etc. In this section, we propose a spatial graph model for representing the polygons that characterize the partially explored obstacle terrain at any point of time. We use a data structure based on a modified version of the adjacency list of a graph. This data structure is specially suited for the operations to be performed on the polygons. The complexities of various algorithms are estimated and compared, in appropriate cases, to the ones based on the conventional adjacency list for the spatial graph.

5.1. THE SPATIAL GRAPH MODEL

The set of polygons that span the obstacle terrain are collectively exhaustive and mutually non-intersecting. Fig. 3 shows a partially explored terrain. A *spatial graph*, $G=(V,E)$, for an obstacle terrain is constructed by representing each vertex of a polygon by a vertex of the graph. An edge of a polygon is represented by the corresponding edge of the spatial graph. To each vertex $v \in V$, we associate a pair of coordinates representing the corresponding vertex point in the space. We note that the spatial graph is a planar graph. Fig. 5 illustrates the spatial graph for the terrain of Fig. 3. Let d denote the maximum number of edges meeting at a vertex of any polygon or equivqlently the maximum degree of a node in the spatial graph. Each polygon is represented by a plane of the spatial graph formed by the corresponding fundamental circuit (refer to [7] for preliminaries on graph theory).

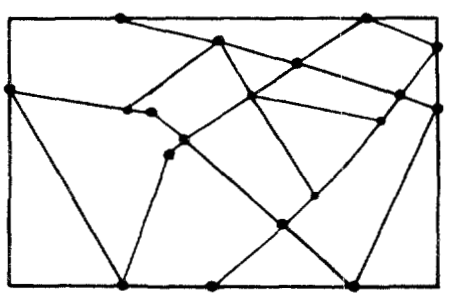
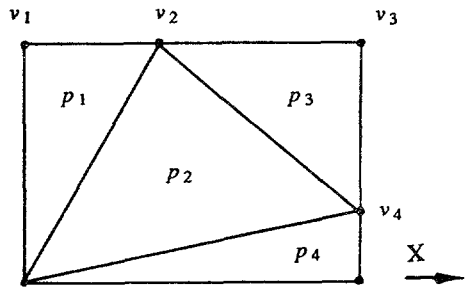


Fig. 5 Spatial graph model for the obstacle terrain shown in Fig. 3

5.2. THE MODIFIED ADJACENCY LIST

The spatial graph is implemented using a *modified adjacency list* data structure. The adjacency list corresponding to each node is represented as an array of all the adjacent nodes sorted according to the increasing values of slopes of the corresponding edges. The slope of any edge (v_1,v_2) at the vertex v_1, denoted by $S(v_1,v_2)$, is measured in terms of the angle subtended by the line $v_1\vec{v}_2$ to a reference direction $\vec{X}$. Let $A[v]=<v_1,v_2,\ldots,v_r>$, $v \in V$, $r \le d$, for $(v,v_i) \in E$, $i=1,2,...r$, represent the modified adjacency list corresponding to the vertex v. Then, we define v_{i+1} to be the *next-neighbor* of v_i, for $i=1,2,...,(r-1)$, with respect to the vertex v. The vertex v_1 is taken to be the next-neighbour of v_r. Fig. 6(b) shows the modified adjacency list structure for the spatial graph of Fig.

6(a). For example, $A[v_4]$, the adjacency list of the vertex v_4 is obtained by traversing in the anti-clockwise direction starting with the direction of X. It is easy to see that the slopes of the edges $(v_4,v_3),(v_4,v_2),(v_4,v_5),(v_4,v_6)$ are in the increasing order. The next-neighbors of v_5 are v_2 and v_4 with respect to v_1 and v_2 respectively. It is important to note that for every vertex $v_i \in V$, there exist unique pair of vertices v and v_j and a polygon p, such that v_j is the next-neighbor of v_i and (v_i,v) and (v,v_j) are the edges of p. Refering to Fig. 6, corresponding to the vertex v_5, the unique pair of vertices is v_2 and v_4, and the polygon is p_2. If the edges (v_i,v) and (v,v_j) belong to the same polygon p, then v_i and v_j are adjacent in $A[v]$. In the modified adjacency list we also store, for each vertex $v \in V$, a sorted list $M[v]=<m_1,m_2,...,m_r>$, such that $m_i = S(v,v_i)$, for $i=1,2,...,r$. Let a polygon p be represented by the ordered set of vertices $<v_1,v_2,...,v_l>$ obtained by starting with the vertex v_1 and traversing along the edges of the p in the clockwise direction. This set can be obtained by starting with v_1 and repeatedly finding the next-neighbors. For example, consider the polygon p_2, given by $<v_2,v_4,v_5>$, in Fig. 6(a). The node v_5 can be reached from v_2 by finding the next neighbor of v_2 with respect to v_4.



(a) Spatial graph

index node	adjacency list $A[v]$
v_1	v_2,v_5
v_2	v_3,v_1,v_5,v_4
v_3	v_2,v_4
v_4	v_3,v_2,v_5,v_6
v_5	v_6,v_4,v_2,v_1
v_6	v_4,v_5

(b) Modified adjacency list

Fig. 6 Modified adjacency list for a spatial graph

Each of the subproblems involved in implementing the spatial-graph may have efficient solutions or analogs with efficient solutions in an environment specifically suited to that subproblem. The field of computational geometry is replete with problems that are similar to the subproblems discussed in this paper. The details can be found in Aho et al [1], Lee and Preparta [13], and Mehlhorn [16]. Here, we are interested in a data structure that solves the subproblems with reasonable over-all efficiency.

5.3. THE COMPLEXITY ANALYSIS

In this section, we list the various computational subtasks involved in the execution of process *PLANNING* and estimate the complexity of each of these subtasks.

(A) Path Planning Algorithms

Path planning involves accessing various polygons and consists of the following operations:

(1) Finding the next-neighbor: One of the basic operations needed for the various subtasks is to find the vertex v_{i+1} that is next to the given vertex v_i, in the polygon $p = <v_1,...,v_i,v_{i+1},\cdots,v_l>$. The next edge v_{i+1} of the polygon can be obtained by carrying out a binary search for the slope $S(v_{i-1},v_i)$ on the array $M[v_i]$, and then retrieving the vertex v_{i+1} is jth entry in $A[v_i]$ such that, $m_{j-1}=S(v_{i-1},v_i)$. The cost of this operation is $O(\log d)$. It is to be noted that this cost is $O(d)$ if the conventional adjacency list representation is used.

(2) Finding near-polygon: The near-polygon p of $s\vec{D}$ is found as follows: Carry out a binary search on the array $M(s)$ to obtain the vertex v_j such that $m_{j-1} \le S(s,D) \le m_j$. This has a time complexity of $O(\log d)$. Let v_{j-1} and v_j be the jth and $j+1$th entries in $A[s]$. The vertices v_{j-1},s,v_j uniquely determine the required near-polygon p, and finding of which suffices for the line 3 of algorithm *PLANNING*. However, the complete vertex set of p can be obtained in $O(n \log d)$ time by repeatedly finding the next-neighbors ,where n is the number of vertices of the polygon p.

(3) Finding the next stop point: The computational subproblem corresponding to lines 6 and 12 of algorithm *PLANNING* involves finding the next stop point, Y, corresponding to the source point s and the final destination point D. The point s is a vertex of p and p is either a free polygon or an obstacle polygon. If p is an obstacle polygon, then the next stop point is computed as decribed in section 2. In such a case we find the a vertex Y_1 (of p) such that the near-polygon of $Y\vec{D}$ is different from near-polygon of $s\vec{D}$ by traversing along the edges of p in clockwise direction. This can be achieved by finding the next-neighbour v and the near-polygon of $v\vec{D}$ at every step. The status of the polygons can be stored by augmenting the adjacency lists. Corresponding to the array $A[v]=<v_1,\ldots,v_i,v_{i+1},...,v_r>$, an auxiliary array $B[v]=<b_1,\ldots,b_r>$ is stored, where b_i gives the status of the polygon given by v_i,v,v_{i+1}. Let n denote the number of edges of the polygon p. Thus, the computation of Y_1 has $O(n)$ such steps and each step costs $O(\log d)$. Thus the complexity of finding the point Y_1 is $O(n \log d)$. The other point Y_2 obtained by traversing along anti-clockwise has the same complexity. If p is a free polygon the next stop point is the intersection point Y of the line joining the source point s to the final destination point D with the polygon p. The point Y can be computed by starting with the vertex s, and repeatedly finding the next-neighbors until an edge that intersect the line $s\vec{D}$ is found. Thus the complexity of this is $O(n \log d)$, because the next-neighbors are found $O(n)$ times.

(4) Finding shorter path: If the currently accessed polygon p is an obstacle-polygon, then the next stop point Y is to be computed as in line 13 of process *PLANNING*. The distance corresponding to the path to the potential stop point Y_1 - via the vertices encountered while traversing along the edges of p in the clockwise direction - is computed during the computation of Y_1 as explained above. This has the complexity of

$O(n\log d)$. The distance corresponding to the other path to Y_2 - via the vertices encountered whlie travelling along the anti-clockwise direction - is computed in the same fashion. This also has the complexity of $O(n\log d)$. Then the path to Y corresponding to the minimum length is chosen. Hence, the complexity of finding the minimum path is $O(n\log d)$. In some cases the criterion for choosing the path could be the number of vertices, total number of changes in the direction, total amount of change in the direction, etc. These values can also be computed in $O(n\log n)$ time, by storing the relevent information in the corresponding adjacency lists.

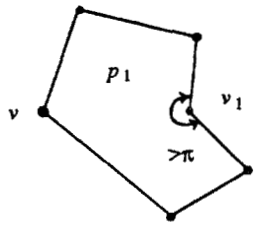
(B) Learning Algorithms

Line 18 of process *PLANNING* describes very important subtask namely learning. Learning is important in two ways. First, it implements the most important feature of the proposed method. Second, the time spent in this task directly adds to the total travel time for the robot. In this phase of navigation, the currently accessed polygon p is modifed based on the sensor data obtained from process *CONTROL*. As explained earlier the process of update has two constituents: a) partitioning the unexplored polygons, b) combining the free-polygons. The step a) involves partitioning the polygon p into visible and invisible parts with respect to s, and also partitioning the visible parts into free-polygons and obstacle-polygons. In either case, the basic operations would be an addition of a new vertex, say v, on an edge, and an addition of a new edge, say (v_1,v_2). The process of merging the free-polygons to form bigger free-polygons involves deletion of vertices and edges.

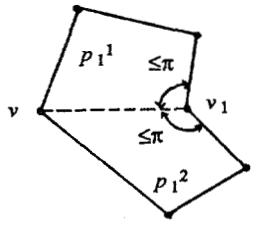
(1) **Insertion and deletion of vertices and edges:** Let the vertex v be created on the edge (v_1,v_2). This can be carried out by overwriting v_2 by v in $A[v_1]$, and overwriting v_1 by v in $A[v_1]$, and then creating $A[v]$ and $M[v]$ with two entries corresponding to v_1 and v_2. Complexity of this process is $O(logd)$. Addition of a new edge (v_1,v_2) can be carried out by entering by entering v_2 into $A[v_1]$ and $M[v_1]$, and also v_1 in $A[v_2]$ and $M[v_2]$. Searching for the respective value in the arrays A and M has the complexity of $O(logd)$. However, entering a new value might involve shifting all the elements of the corresponding arrays. Hence, the complexity of addition of a new edge is $O(d)$. Similarly, the complexity of deleting an edge is $O(d)$. The deletion of a vertex v involves deletion of all the edges incident on v, and thus has the complexity of $O(d^2)$.

(2) **Decomposition of polygons:** Let the sensor data introduce k new edges for partitioning unexplored polygon into visible and obstacle polygons. A polygon p_1 whose interior is visible froma vertex may not be convex because the angle included by vertices may not be less than π (as shown in Fig.7(a)). The vertex v from which the polygon is visible is joined to each vertex v_1 (of p_1) such that the included angle at v_1 is not less than π as shown in Fig.7(b). If v is the only vertex with includes angle not less than π, then v is joined to one of the other vertices. Such a decomposition of the visible polygons always results in convex polygons because all the included angles are made less than π. This decomposition process can introduce $O(n)$ edges, where n is the number of edges of the unexplored polygon. Hence the complexity of partitioning the polygon has the complexity of $O((k+n)d)$.

(3) **Merging of free-polygons:** When the new free polygons are created, we examine all the neighbor free polygons to find out if they can be merged with the new ones to form bigger free polygons. In general there could be overlapping subsets of free polygons that could be merged. In such case the merging could



(a) The polygon p_1 is visible from v but not convex



(b) The polygon is decomposed into convex polygons p_1^1 and p_1^2 by joining v to v_1

Fig. 7 Decomposition of visible polygon

be carried out according to a stategy that the resultant number of free polygons is minimum. There have been efforts to design a free-polygon merging algorithm that has a worst-case time complexity better than the brute-force of exhausting all the combinations of polygons. This problem is currently being pursued by the authors. The resultant merged free-polygon has to have the largest area among all the combinations of the given free polygons. The complexity of the exhaustive enumeration method is $O(2^pK+pnlogd)$, where K is the cost of computing the area for the given choice of free-polygons, p is the number of polygons to be merged, and n is the number of edges of the biggest polygon among the polygons to be merged. The first factor corresponds to the cost of computing the areas for all the combinations of polygons, and second factor corresponds to merging the selected polygon combination. Though the complexity of this phase is high, in situations where there are only a few polygons to be merged this algorithm gives reasonable results. However, the authors feel that there could be efficient solutions to this problem.

5. CONCLUSIONS

The capability to learn is essential to an intelligent autonomous mobile robot navigating in an unexplored terrain. An efficient navigation technique has been developed by the authors [10,11] to incorporate learning in a robot navigating in an unexplored terrain. As a part of learning, the robot gradually builds the terrain model as it undertakes a number of goal-directed traversals. Consequently, the paths of navigation undertaken by the robot become closer to globally optimal paths, as learning proceeds.

In this paper we propose a technique for autonomous robot navigation in terms of a system of concurrent processes running on the computing system of the robot. This system is shown to be free from deadlocks and starvation. The performance analysis of the concurrent processes reveals that the various path planning and learning operations have to be expedited in order to reduce the time of navigation for any path. This neccessitates an obstacle terrain model that efficiently supports all the operations involved in path planning and

learning. A modified adjacency list data structure is proposed for the obstacle terrain model. This data structure is proven to be efficient in supporting the operations to be performed on the terrain model. The complexities of various algorithms for manipulating the terrain model are estimated. These algorithms are currently being implemented on a hypercube computing machine mounted on HERMIES-II at Oak Ridge National Laboratory.

REFERENCES

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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