Multiresolution Path Planning for Mobile Robots

SUBBARAO KAMBHAMPATI AND LARRY S. DAVIS, MEMBER, IEEE

Abstract—**The problem of automatic collision-free path planning is central to mobile robot applications. An approach to automatic path planning based on a quadtree representation is presented. Hierarchical path-searching methods are introduced, which make use of this multiresolution representation, to speed up the path planning process considerably. The applicability of this approach to mobile robot path planning is discussed.**

I. INTRODUCTION

THE PROBLEM of automatic collision-free path planning is central to mobile robot applications. Path planning for mobile robots is in many ways different from the more familiar case of path planning for manipulators [19]. Examples of these differences are as follows.

1) A mobile robot may have only an incomplete model of its environment, perhaps because it constructs this model using vision and thus cannot determine what is occluded by an object.

2) A mobile robot will ordinarily negotiate any given path only once (as opposed to a manipulator, which might perform the same task thousands of times). This implies that it is more important to develop a negotiable path quickly than it is to develop an "optimal" path, which is usually a costly operation.

3) A mobile robot should keep as far away from obstacles as possible. A manipulator's reason for doing this is mainly collision avoidance. For a mobile robot proximity to obstacles also gives rise to severe occlusion and reduction in the field of view.

Conventional path-planning algorithms can be divided broadly into two categories. In the first category are the methods which make trivial (if any) changes to the representation of the image map before planning a path. The regular grid search [19] and vertex graph methods [9], [18], [10] fall into this category.

Though these methods keep the representational cost to a minimum, their applicability to mobile robot navigation is limited. For example, the regular grid search is [19], [20] "too local" and its path planning cost increases with grid size rather than with the number of obstacles present. Further, both regular grid search and vertex graph methods generate paths which clip obstacle corners.

Manuscript received October 23, 1985; revised February 18, 1986. This work was supported by the Defense Advanced Research Projects Agency and the U.S. Army Night Vision and Electro-Optics Laboratory under contract DAAK70-83-K-0018.

The authors are with the Computer Vision Laboratory, Center for Automation Research, University of Maryland, College Park, MD 20742, USA.

IEEE Log Number 8608952.

The methods in the second category make elaborate representation changes to convert to a representation, which is easier to analyze before planning the path. Free space methods [1], medial axis transform methods, Voronoi methods, etc., fall into this category. A potential practical shortcoming of such methods for mobile robot navigation is that the path-planning cost is still very high because of the representation conversion process involved.

Though the above two categories by no means exhaust the existing methods (there are configuration space methods that use a vertex graph approach [7] and others that use a free space approach [8] to solve the manipulator findpath problem), they do point out that what mobile robots need may be a compromise between these two categories.

It is these considerations that motivated the multiresolution (hierarchical) representation based path-planning algorithms described in this paper [3], [6]. Similar considerations also led to the use of hierarchical representations in manipulator "findpath" problems (see Section IV for a discussion of related work). In this paper, we first develop a method of path planning for mobile robots using a hierarchical representation based on quadtrees and then describe staged search as a way of exploiting the hierarchical nature of the representation to gain substantial computational savings. Throughout this paper we restrict our attention to two-dimensional path planning without rotation and a vehicle with circular cross-section.

Section II develops a quadtree-planning algorithm based on A^* search. Section III presents a staged (hierarchical) path-planning algorithm, which has computational advantages as compared to the pure A^* search on quadtrees. The staged search involves inclusion of gray nodes in the search. Section IV discusses related work, and Section V summarizes the conclusions reached from this research and discusses future directions. In the remainder of this section we define some terms used in these discussions.

Quadtree-Related Terminology: A *quadtree* is a recursive decomposition of a two-dimensional picture into uniformly colored $2^i \times 2^i$ blocks (e.g., see Fig. 1) [16]. A node of a quadtree represents a $2^j \times 2^j$ square region of the picture. A *free node* of a quadtree is a node of the quadtree representing a region of freespace. An *obstacle node* is a node representing a region of obstacles. A *gray node* is a node representing a region having a mixture of freespace and obstacles. A *leaf node* of a quadtree is a tip node of the tree. In ordinary quadtrees, leaf nodes are always obstacle nodes or free nodes, but in pruned quadtrees (see below), they may also be gray nodes. For any gray node G, $S(G)$ denotes the subtree rooted at G. $L(G)$ denotes the number of leaf nodes in $S(G)$. The *gray content* of a gray node G is defined as the number of

0882-4967/86/0900-0135$01.00

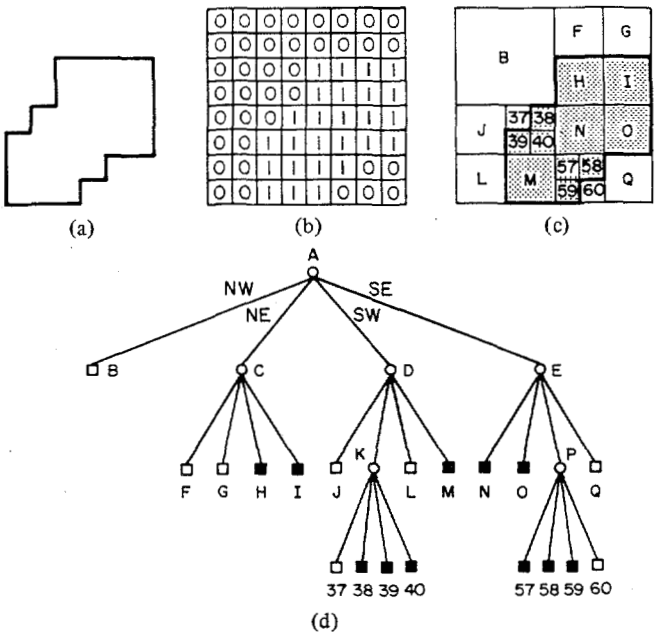


Fig. 1. Quadtree representation. A region, its binary array, its maximal blocks and the corresponding quadtree (from [16]).

obstacle pixels in the region represented by G, and the *grayness* of G is the percentage of obstacle pixels in that region.

When some of the gray nodes of a quadtree are made leaf nodes, thereby pruning the subtrees rooted at those gray nodes, the resulting structure is called a *pruned quadtree*; it represents the same space as the original quadtree but with a reduced resolution.

A Terminology:* A^* is a classical minimum cost graph search algorithm, whose optimality properties are well known [Nilsson 80]. In this algorithm OPEN is a list consisting of all the nodes in the search graph that are generated but not yet expanded. CLOSED is the list of nodes in the graph that have been expanded. *Best node* is the node that is currently being expanded in the search. This node has the best evaluation (i.e., minimal path cost) among the nodes on OPEN. The *predecessor* of a node N in the search graph is the node preceding N on the current best path to N (from the start node).

II. Quadtree-Based Path Planning

A. Representation Preprocessing

We have developed an algorithm for mobile robot path planning based on a quadtree representation of the robot's immediate environment. If there are large areas of free space (or obstacles), then those areas can be represented by a few large blocks in the quadtree and can be dealt with as units by the planning algorithm.

Given a binary array or raster representation of a robot's immediate environment we first grow the obstacles by the radius of robot's cross section [9] and then convert the raster into a quadtree representation using a raster to quadtree conversion algorithm [13]. This algorithm is of complexity $O(n)$, where n is the number of pixels in the image being converted. In the resulting quadtree blocks of zeros represent free-space nodes and blocks of ones represent obstacle nodes.

In the second stage of preprocessing, we compute the distance transform of the set of 0's, i.e., the free-space blocks. This determines, for each block of free space, the minimal distance between the center of that block and the boundary of a block of obstacles. Samet [14] describes an algorithm for computing this distance transform for quadtrees which is of complexity $O(n)$, where n is now the number of leaf nodes in the quadtree.

B. Path-Planning Algorithm

Given the start and goal points, we first determine the quadtree leaf nodes S and G, representing the regions of the image containing these points. Next, we plan a minimum cost path between S and G in the graph formed by the non-obstacle leaf nodes of the quadtree, using the well known A^* search algorithm with the evaluation function f of a node c defined as

$$f(c) = g(c) + h(c).$$

Here $g(c)$ represents the cost of the path from S to c and $h(c)$ represents the heuristic estimate of the cost of the remaining path from c to G.

Since the cost of a path should depend both on the actual distance travelled along the path and the clearance of the path from the obstacles, we define $g(c)$ as

$$g(c) = g(p) + \tilde{g}(p,\ c)$$

where $g(p)$ is the cost of the path from S to c's predecessor p on the path and $\tilde{g}(p, c)$ is the cost of the path segment between p and c. The latter function in turn is defined as

$$\tilde{g}(p,\ c) = D(p,\ c) + \alpha \cdot d(c)$$

with $D(p, c)$ representing the actual distance between nodes p and c, given as half the sum of the node sizes, and $d(c)$ representing the cost incurred by including node c on the path. $d(c)$ depends upon the clearance of the node c from the nearby obstacles. We chose a linear shape for the cost function d, defining $d(c)$ as

$$d(c) = 0_{\max} - 0(c)$$

where $o(c)$ is the distance of the node c from the nearest obstacle given by the quadtree distance transform and $0_{\max}$ is the maximum such distance for any node in the quadtree (so that $d(c)$ is always positive). α in the equation for $\tilde{g}(p, c)$ is a positive constant which determines by how far the resultant path will avoid obstacles.

Finally, $h(c)$ is calculated as the Euclidean distance between the midpoints of the regions represented by c and G. Clearly, this measure is a lower bound on the actual minimum cost path between c and G; thus an A^* search with this measure as its heuristic estimate is admissible. The power of this heuristic depends upon the average deviation of the minimum cost path from the straight line path. It is highest for the case where α is zero and decreases as α increases. It is of course possible to use more informed, but inadmissible, heuristics to speed up this search. For example, both "the number of obstacles

intersecting the straight-line path between c and G," and "the total area of the obstacles intersecting the straight line path between c and G" are more powerful heuristics than the one we are using, but they are not admissible.

The node expansion process involves finding the nonobstacle leaf nodes adjacent to the node being expanded. We accomplish this by using a neighbor finding strategy similar to that given by Samet [15] with two differences. First, only the neighbors in the horizontal and vertical directions are considered—diagonal neighbors, which share only single points with the current node, would result in inflexible paths which clip obstacle corners. Secondly, when one of the neighbors given by the quadtree neighbor finding algorithm is a gray node, we find the nonobstacle leaf nodes, if any, of the quadtree rooted at that gray node that are adjacent to the node being expanded and consider them as neighbors.

The result of applying the above A^* algorithm to the quadtree is a list of nodes from the quadtree (ordinarily of varying sizes) which define a set of paths between the start and goal nodes. If desired, an optimal path through these blocks can be computed, or the center points of consecutive blocks on the list can be connected to compute a negotiable path.

C. Results

Fig. 2 contains a simple example of a path obtained using this algorithm. Fig. 2(a) is a binary array with start and goal points marked, along with an indication of the path determined by the algorithm. Fig. 2(b) contains the tree data structure that represents the quadtree, in which the blocks on the computed path are hatched. It is important to note the reduction in the number of nodes achieved by the algorithm. Fig. 3(a) shows a path planned on a more complicated image map with the constant α set to one, and Fig. 3(b) shows the same example with α set to zero. Notice that the time taken in the former case is considerably higher than in the latter. This should be expected, since as noted in the last section, the heuristic power of h reduces as α increases.

It is also interesting to note that although it is true that the quadtree representation is sensitive to displacements of obstacles with respect to the grid boundaries, the savings in space and computation afforded by this method are still very high on the average. Further, Samet *et al.* [17] point out that for complicated images the positioning of the image origin is likely to have little effect on the number of nodes in the resultant quadtree.

D. Advantages of the Quadtree Approach

Compared to the first category of path-planning algorithms mentioned in the introduction, such as the grid search method, the path-planning cost for quadtree-based search will be substantially lower because the number of nodes to be searched in the quadtree approach is considerably smaller. In fact, the number of leaf nodes in a quadtree of an image map having polygonal obstacles is approximately $2/3 \cdot O(p)$ [16], where p is the sum of the perimeters of the (polygonal) obstacles in terms of the lowest resolution units, in our case pixels (or grid points). Thus A^* search will only have to deal with about $O(p)$ nodes in the case of a quadtree, instead of the

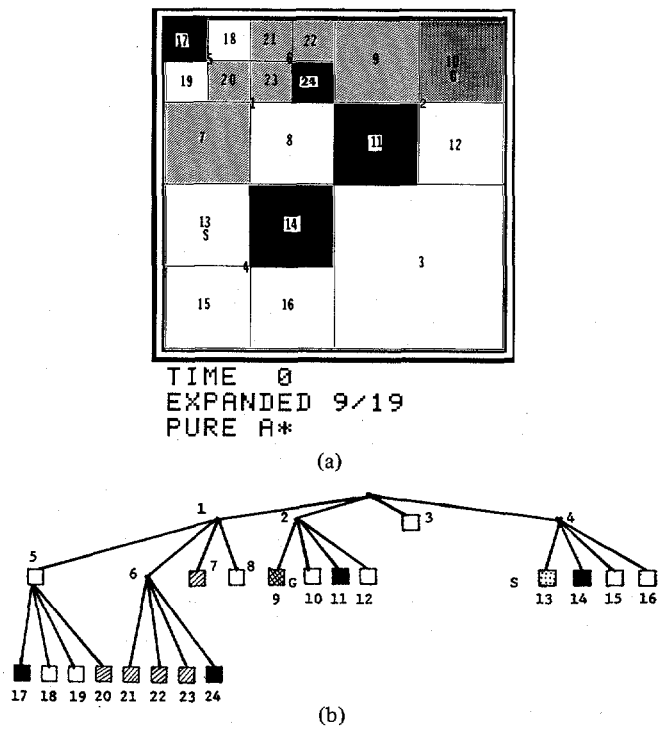


Fig. 2. (a) Single stage path planning on a quadtree representation. The figure shows a binary image with obstacles represented by black regions, start node indicated by S, and goal node indicated by G. The nodes on the path found by the algorithm are represented by hatched regions. All the node boundaries are outlined with black lines. (b) Tree data structure representing the quadtree of the binary image in (a). The black nodes correspond to the obstacle regions; S and G correspond to the start and the goal nodes; and the hatched nodes represent the nodes that fall on the path generated by the algorithm.

n^2 grid points in the case of a grid search, a substantial reduction. Similarly the "local-bound" behavior of the first category algorithms is absent in this approach, because the nodes are on the average much larger than single pixels and it is straightforward to determine the "nearness" of the nodes to the obstacles. Moreover, a hierarchy of different levels of description of the space that is available with quadtrees enables us to search for a path close to obstacles only when necessary. Corner-clipping inflexible paths are eliminated by considering only neighbors in the horizontal and vertical directions.

Unlike the second category of methods that involve a costly change of representation, the proposed approach has a very small representation overhead. As pointed out in Section II-A, both the representation algorithms involved are of complexity $O(n)$, whereas many methods of the second category have a representational cost that is far higher.

Thus quadtree based path planning is a good compromise between free-space algorithms and grid-search type algorithms. In addition, the path produced by the quadtree algorithm, although not "optimal", is a "negotiable" path which can be computed relatively quickly. Apart from this, the hierarchical nature of the representation gives many advantages in path planning. For example, we can easily constrain the path to satisfy certain conditions, such as specification of minimal clearance of the path. More importantly, we can make the search *staged*, i.e., plan a path at a coarser lever and subsequently refine it as needed, thus reducing the planning

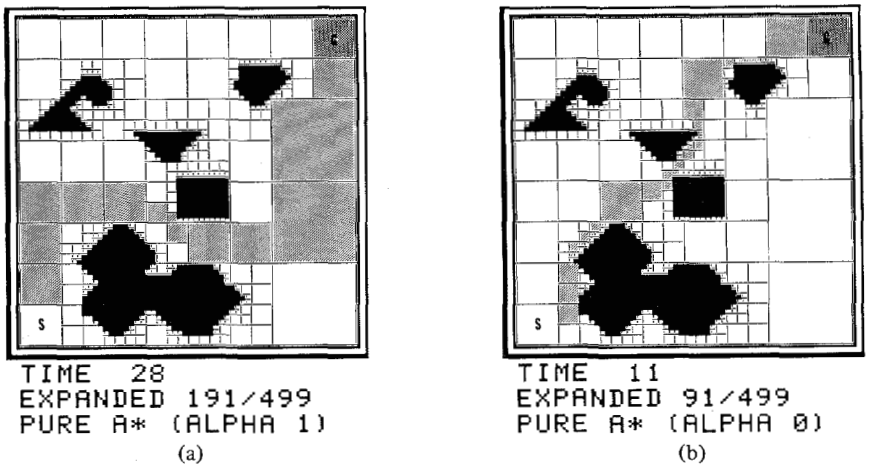


Fig. 3. Example of single stage planning. (a) Result of single stage A* search on the pure quadtree representation of an image, with α set to 1. S and G represent start and goal nodes and hatched regions represent nodes on the planned path. (b) The same example as in Fig. 3(a), but with α set to 0.

cost substantially. The advantage of the former has been discussed in Section II-B. We will discuss the latter at greater length in the next section.

III. Staged Path Planning

A. Motivation

Though the algorithm which we presented in the Section II is relatively efficient, it can be improved upon substantially. We often get undesirably small "black" (obstacle) nodes in the quadtree representation. One obvious source for this may be the existence of very small obstacles in a region of the environment that is otherwise obstacle-free. A more important source of these black nodes is the representation of irregular obstacles in quadtrees. Due to the recursive nature of the quadtree, these small black nodes will fragment the free space, giving rise to an undesirable increase in the depth of the quadtree and the number of leaf nodes and consequently increasing the cost of the search.

We can deal with this problem by first planning the path in a reduced-resolution quadtree, called a *pruned* quadtree, that contains gray leaf nodes, corresponding to mixtures of obstacles and free space. This implies that a node can now have gray neighbors. An algorithm capable of planning a global path at this coarser level, and subsequently developing the path inside the gray nodes (which are included in the global path) in the second stage, can give rise to savings in terms of computation without significant degradation of the path obtained. As mentioned in Section II-D, the number of leaf nodes is on the order of the sum of the perimeters of the obstacles, measured in the lowest resolution units. Thus conducting search at a resolution l levels below the pixel resolution reduces the "sum of the perimeters" and "number of leaf nodes" by a factor of 2^l thereby substantially reducing the time complexity of the search.

There are two aspects to this staged search that deserve detailed attention—the treatment of gray leaf nodes during planning and the generation of the pruned quadtree from the original quadtree. In the next two subsections we shall discuss these two aspects in detail.

B. Dealing with Gray Leaf Nodes

When planning a path through the pruned quadtree, we have to deal with gray leaf nodes. Specifically, the following three questions must be answered: 1) what is done when one of the neighbors of the current best node (the node that is currently being expanded in the A^* search) is a gray node? 2) how is the current path expanded when the current best node is a gray node? and 3) how is the first stage path, involving gray leaf nodes, processed to get the final path that contains free nodes exclusively?

We shall address these in the following subsections.

1) Gray Leaf Neighbors: If one of the neighbors N of the current best node B is a gray node, then before putting N on the OPEN list we must ensure that N can be entered from B. If B is a free node, then N can be entered if and only if there exists at least one free node m in $S(N)$, such that m is adjacent to B. In addition, if B itself is a gray node, then N can be entered from B as long as there exists a free node e in $S(B)$ such that e is adjacent to m. Note that checking this entry condition alone does not guarantee that the gray node N is passable, i.e., that a path from B through N to a third node C exists. For example in Fig. 4, N can be entered from B, through the free node m, but N cannot be exited, except back to B.

If we decide to put N on the OPEN list then we shall include in the heuristic value of N a measure of the "path complexity" $c(N)$ inside N. (This measure should be zero for a free node since the path inside the free node can be a straight line.) In general, it is difficult to give a measure which truly represents the complexity of a path inside the gray node, since at this point in the search the direction in which the path will be exiting the gray node is unknown. But in practice any measure depending upon the gray content (number of obstacle pixels inside the gray node) of the gray node will be a good choice. One such normalized complexity measure for the gray node N

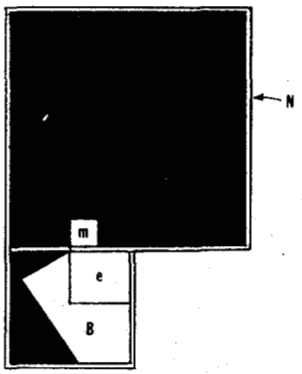


Fig. 4. Dealing with a gray neighbor. The gray neighbor N will be placed on OPEN since there is a free node m in $S(N)$ that is adjacent to e, the exit node of the best node B corresponding to N. Notice that the presence of a gray node on OPEN does not guarantee the passability of the node; for example, in the present case, N cannot be excited to any node other than B.

is

$$c(N) = \frac{\text{gray content } (N)}{\text{size } (N)}.$$

Given two gray nodes having the same gray content, the path complexity should intuitively be higher for the gray node representing a region with more obstacle nodes. Thus a better, although costlier, complexity measure of the gray node N will take into account the number of obstacle nodes in $S(N)$.

Once the heuristic value is calculated, the gray node is placed on the OPEN list and it can be selected for expansion whenever its f-value is the best among the nodes on the OPEN list.

2) Expanding Gray Nodes During Search: When the current best node B happens to be a gray node, expanding B becomes a more involved operation. After generating B's neighbors we must ensure that for each of these neighbors N there exists a path through B that connects B's predecessor P on the current path to N (see Fig. 5). We refer to this as the "reachability" analysis for neighbor N. Secondly, for each neighbor N that can thus be reached we have to record as N's g-value an estimate of the shortest path to N through B. This estimate should take into account the fact that the shortest path through B may not be a straight line path, since B is a gray node.

One way to achieve the above two objectives is by performing an A^* search rooted at B to determine if N can be reached from P. If the A^* search finds such a path to N, then we can use the cost of that path as the g-value of neighbor N. The advantage of this method is that we have the full power of A^* search. The principal disadvantage to this method is that we need to perform this A^* search once for every neighbor of B, a rather large price to pay for path optimality. To avoid this disadvantage, we follow a distance-transform-based gray-node-expansion strategy, described below.

Let f be a free node in $S(B)$ such that f is adjacent to B's predecessor, P. Notice that there can be more than one such free node in $S(B)$. If P is a gray node, then we require that f be adjacent to a free node in $S(P)$ (called an "exit node" for P). This exit node would have been determined when P was being

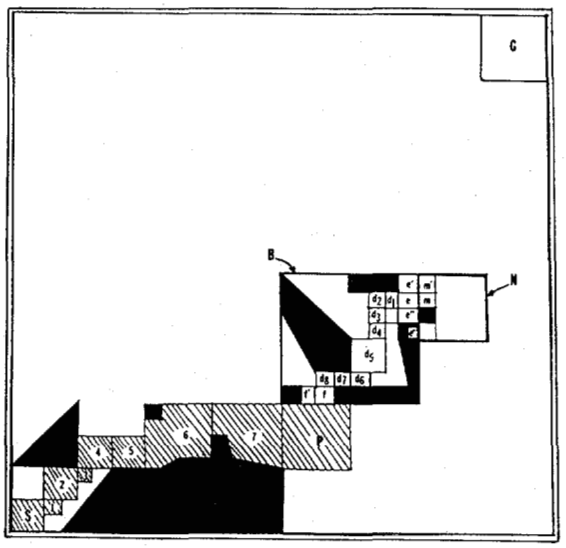


Fig. 5. Expanding a gray node. In the figure, B represents the current best node in the A* search; N is a neighbor of B. S and G represent start and goal nodes respectively. The path to B consists of nodes S—1—2—3—4—5—6—7—P—B in that order; thus P is B's predecessor on the current path. Of the two nodes f and f' that are adjacent to P, f is nearer to G; so f is the entry node of B. Of the four nodes e, e', e'', and e''' that are adjacent to node N, N cannot be entered from e'', and e''' cannot be reached from f. Thus e and e' are the possible candidates for exit nodes. e is chosen as the exit node for B corresponding to N since it is nearer to f than e'. e—d_1—d_2— $\cdots$ —d_8—f represents the path that will be developed inside B during the second stage of path planning.

expanded. We illustrate all this in Fig. 5. P is the predecessor of the best node B, and N is a neighbor of B. Both f and f' are free nodes in $S(B)$, and they are also adjacent to P. In such a situation, we choose the free node which has the least straight line distance to the goal node—in this case f. Thus the current path enters B through f. f is recorded as the *entry node* of B.

Next, we compute a distance transform of the region represented by B with respect to f. This involves recording for each free node f' in $S(B)$, f''s shortest distance (which we refer to as dis (f, f')) from f. To carry out this computation, we first initialize dis (f, f) to zero (see Fig. 5), and dis (f, f') for all other free nodes f' in $S(B)$ to ∞. Next, we carry out the propagation step: we find all the neighbors of f, f', which are in $S(B)$ and for each such neighbor f' calculate dis (f, f'), as the sum of dis (f, f) and the nodal distance between f and f', $D(f, f')$. To ensure that the path inside B will take clearance from the obstacles into consideration, we include the cost of the node $d(f')$ (see Section II-B) in dis (f, f'). We repeat this propagation step for all the neighbors of f, with the neighbors taking the role of f, and so on, until we exhaust all the free nodes in $S(B)$. The detailed procedure is given in an algorithmic fashion in Listing 1. and is, essentially, the familiar shortest path algorithm for the case of "single source multiple destinations" [5].

Having computed the distance transform of B with respect to f, as detailed above, we are now ready to continue with the expansion of B. For each of B's neighbors N, N is marked reachable if there exists a free node e in $S(B)$ which satisfies the following two conditions (see Fig. 5):

Listing 1. Distance Transform Algorithm.

```
Procedure Distrans (B, f);
/*B is the gray node representing the region in
which f is a free node. The algorithm computes the
distance transform of B with respect to f. */
begin
∀node ∈S(B) dis (f, node) ← ∞;
dis (f, f) ← 0;
distrans_OPEN ← list (f);
distrans_CLOSED ← nil;
until null (distrans_OPEN)
do
    f' ← first (distrans_OPEN);
    distrans_OPEN ← rest (distrans_OPEN);
    NBRS ← get_neighbors_inside_the_node (f', B);
    /* get the neighbors of f' that lie in S(B)*/
    foreach nbr ∈ NBRS
      do
      newdis ← D(f', nbr) + α·node_cost (nbr) + dis(f, f');
      if dis (f, nbr) > newdis and nbr ∈ distrans_CLOSED then
        distrans_CLOSED ← remove (distrans_CLOSED, nbr);
      dis (f, nbr) ← min {dis (f, nbr), newdis];
      if nbr ∉ distrans_OPEN and nbr ∉ distrans_CLOSED then
        distrans_OPEN ← append (distrans_OPEN, nbr);
      od
    od
end Distrans;
```

1) dis $(f, e) < \infty$. This ensures that there is a path between e and f inside $S(B)$.
2) N can be entered from e. As discussed in Section III-B-1, if N is a free node, this condition is satisfied as long as N and e are adjacent. On the other hand, if N is a gray node then the condition is satisfied if there exists a free node m in $S(N)$ such that m and e are adjacent.

The node e, satisfying the above two conditions is marked as the exit node of gray node B with respect to N. Notice again that there may be more than one such node. For example, in Fig. 5, e and e' satisfy both conditions, since there is a path from f to each of these nodes, and N can be entered from both the nodes. In such a situation, we select the node with smaller distance to f as the exit node. Thus, in Fig. 5, e would be chosen as the exit node of B with respect to N.

Neighbor N of the best node B is placed on the OPEN list only if there exists an exit node e, for B with respect to N. If N does go on to the OPEN list, the sum of the g-value of B's predecessor P, $g(P)$, and dis (f, e) is recorded as $g(N)$. If N is a gray node, we have to include in N's heuristic value $h(N)$ an estimate of the path complexity inside N as discussed in Section III-B-1. This completes the discussion of the expansion of the best gray node B.

At this point it is worth noting the advantages of using the distance transform in dealing with gray leaf nodes: First, it eliminates the necessity of multiple rooted A^* searches. The distance transform computation is efficient on the quadtree representation. Second, developing the path inside the gray nodes, after the first stage, is very simple.

3) Developing the First Stage Path Containing Gray Nodes: At the end of the first stage of the staged search the planned path may contain gray nodes as well as free nodes. The path inside the gray nodes is developed in the second stage.

If rooted A^* search were used in expanding gray nodes (as discussed in the previous subsection), then this second stage would simply amount to concatenating these paths through gray nodes with the free nodes.

If the distance transform is used instead of rooted A^* search, then the path development inside gray nodes is not as simple. The path development computation involves the following (refer again to Fig. 5):

For each gray node B on the path we retrieve B's entry node f (recorded while expanding B) and B's predecessor P and successor N on the path. Next, using N, we retrieve the exit node e, for B corresponding to N. Now developing the path inside B amounts to finding the shortest path between e and f and inserting it in between N and P. Finding the shortest path between e and f simply involves backing up to f through neighbors having smallest distance transform values. In Fig. 5, for example, the shortest path between e and f, as found by this method, is $e - d_1 - d_2 - \cdots - d_8 - f$.

C. Pruned Quadtree Generation Methods

The primary motivation for pruned quadtree based staged search, as noted in Section III-A, is to offset the disadvantages of the fixed grid uniform recursive decomposition involved in quadtree representation. By choosing an appropriate pruned quadtree, we can avoid a profusion of nodes in a region of the image map, which is relatively obstacle-free. This poses the question of how to decide when a region, or the gray node representing it, is relatively obstacle-free. None of the simple measures (such as grayness of the node) alone can answer this question entirely satisfactorily. For example, the grayness of a node tells us nothing about the distribution of the obstacles in the region represented by that node, and in the extreme case a small value of grayness may actually be the result of a streak of obstacle pixels through the middle of the node. More commonly, a small grayness value of a gray node may be due to a scattered obstacle distribution inside the gray node, which fragments the free space. In such a case, the gray node is obviously a bad candidate for a leaf node in the pruned quadtree. At the same time, we do not want to base our decision on a very involved analysis of the gray node, because this may increase the cost of pruned quadtree generation to the point where the staged search is, overall, less efficient than searching the original quadtree.

Keeping all these considerations in mind we experimented with the following pruned-quadtree-generation strategies.

1) *Using the Grayness of the Gray Node:* This method uses a threshold on grayness to identify leaf nodes of the pruned quadtree. The quadtree is traversed in a breadth first fashion and any gray node whose grayness falls below the threshold is made a leaf node of the pruned quadtree. Once a gray node G is chosen as a leaf node, the breadth first traversal ignores $S(G)$. Fig. 6(b) shows the pruned quadtree generated from the quadtree in Fig. 6(a), using this method, and also gives the result of a staged search on this pruned quadtree. This method sometimes chooses very large gray nodes, having a small grayness but a scattered obstacle distribution, as leaf nodes. This is undesirable, since the cost of the distance transform increases polynomially ($O(n^2)$) with the number of free nodes inside the region represented by the gray node. We

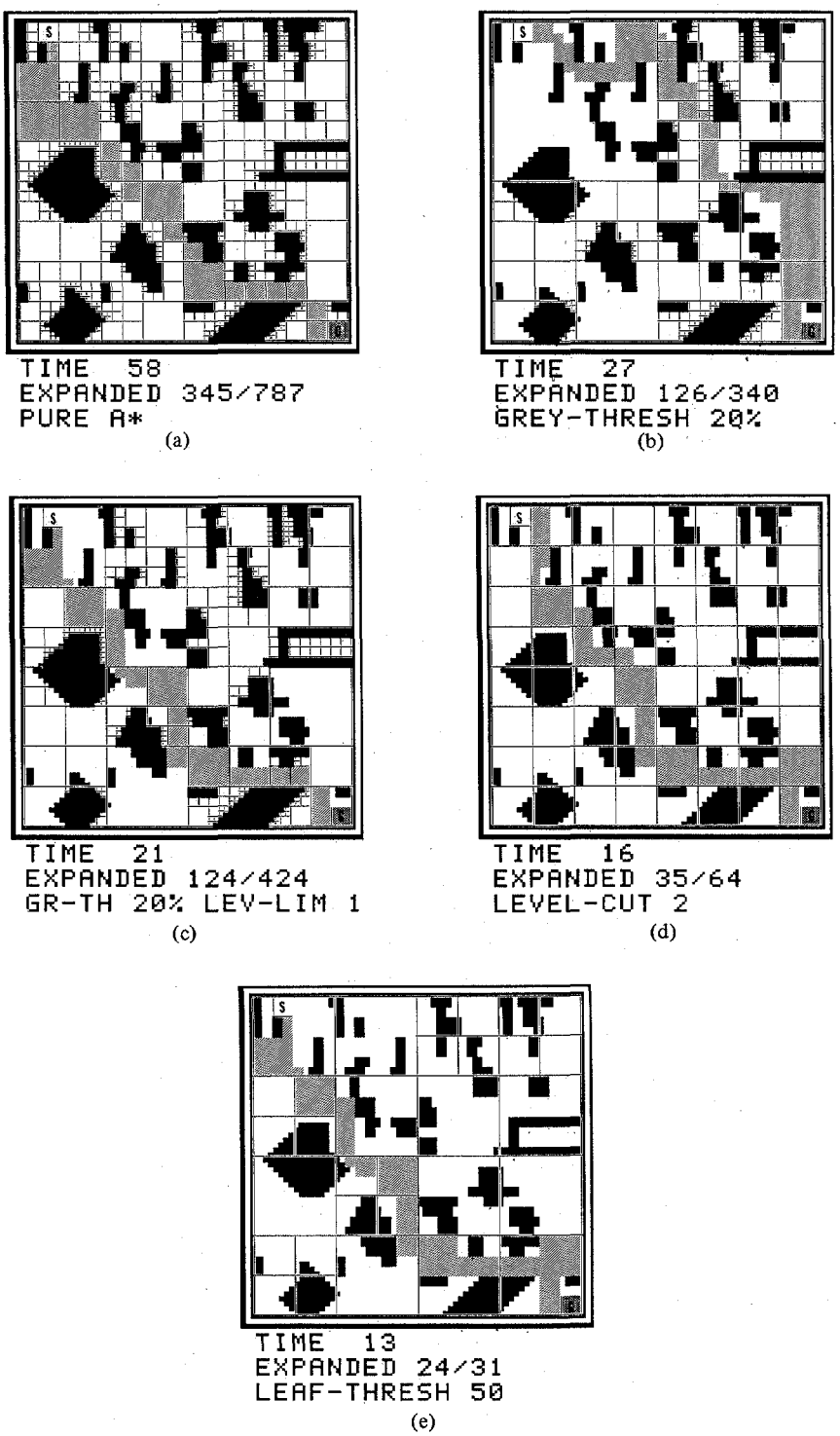


Fig. 6. Experiments with various pruned quadtree generation strategies. (a) An example of single stage planning on a pure quadtree. This is going to be compared to the various pruned quadtree generation methods. (b) An example of staged path planning, with grayness thresholding (method 1) as the pruned quadtree generation method; the threshold is 20 percent. The leaf node boundaries are outlined and the nodes on the path are hatched. Notice the large gray leaf node in the top left quadrant and the second stage path developed inside that node. (c) Staged path planning with a pruned quadtree generation method that takes both grayness and size information into account (method 2). Compare the results to (b). (d) Staged path planning where the pruned quadtree is generated by truncating the original quadtree below a fixed level (method 3); level threshold is two. (e) Staged path planning with leaf node thresholding (method 4) as the pruned quadtree generation strategy. This method is adopted as the pruned quadtree generation strategy for the subsequent experiments.

can partially offset this problem by adaptively setting the grayness threshold to lie between zero and the minimum of the grayness of the top few gray nodes of the quadtree (say, the root node and its four sons), thus ensuring that none of those top level gray nodes become leaf nodes. More importantly, pruned quadtrees generated using this method are potentially unstable with respect to the grayness threshold—a small change in the threshold may change the structure of the pruned quadtree drastically.

2) *Using Size Information with GraynessThreshold:* This is one way of avoiding the problem of overly large gray leaf nodes associated with the first method. This method also traverses the original quadtree in a top–down breadth–first fashion. When it encounters a gray node whose grayness falls below the grayness threshold, it checks that the size of the gray node falls below the size threshold before making the gray node a leaf node. This method is superior to the first one, because it will not choose large gray nodes with small grayness and scattered obstacle distribution as leaf nodes. Fig. 6(c) shows the pruned quadtree generated from the quadtree in Fig. 6(a), using this method and also gives the result of a staged search on this pruned quadtree.

3) *Truncating the Tree Below a Fixed Level:* This method makes any gray node of the original quadtree lying at a fixed level a leaf node of the pruned quadtree. Fig. 6(d) shows a pruned quadtree generated from the quadtree in Fig. 6(a) using this method and also gives the result of a staged search on this pruned quadtree. The level of truncation could be determined based on a histogram analysis on the obstacle node levels. This relatively straightforward method turns out to be unsatisfactory since depending upon the level of truncation it may make either of the following two undesirable decisions: a) it may choose very large gray nodes with scattered obstacle distributions as leaf nodes (like the first method) and b) it may prevent moderately large gray nodes, which represent regions with small grayness and very few free nodes, from becoming leaf nodes.

From the above three methods, it is obvious that the criteria for pruning should be independent of the size of a gray node and should instead depend mainly on the cost of gray-node evaluation. As observed already, the cost of the distance transform on a gray node G depends upon the level of fragmentation of the region represented by G. $L(G)$, the number of leaf nodes in $S(G)$, is a good measure of the fragmentation of the region: the higher $L(G)$, the higher the fragmentation and the costlier the distance transform. Notice that $L(G)$ does not depend on the size of G; thus larger nodes with relatively low fragmentation will also be included in the pruned quadtree as leaf nodes. Method 4, which we chose as our method of pruned quadtree generation, uses $L(G)$ as its basis for pruning.

4) *Using L(G), the Number of Leaf Nodes in S(G):* This method uses a threshold on $L(G)$ to identify leaf nodes of the pruned quadtree. Any gray node G, whose $L(G)$ is lower than the threshold, is made a leaf node of the pruned quadtree in a breadth-first traversal of the quadtree. Computation of $L(G)$ is straightforward and is in fact even cheaper than grayness computation. For a given threshold, there is an upper bound on the cost of gray-node evaluation based on the distance transform, and thus the cost of the staged search can be effectively controlled. One important advantage of this method is that the threshold on $L(G)$ is relatively independent of the specific image and depends only on global criteria such as maximum allowable gray node evaluation cost and maximum allowable suboptimality of the resultant path. Figure 6(e) shows a pruned quadtree generated using this method from the quadtree in Fig. 6(a), and it also gives the result of a staged search on this pruned quadtree.

An important difference between methods 1, 2, and 4 is that the latter may also include a gray node with very high gray value as gray leaf node, as long as its $L(G)$ value falls below threshold. This means that we are no longer assured of the fact that all gray nodes are relatively obstacle-free. Thus there is an increased need to penalize gray nodes on OPEN having higher grayness so that the search is inhibited from expanding these nodes unless absolutely necessary.

Finally, we observe that none of the above methods uses a criterion that properly reflects the distribution of obstacles inside the gray leaf node. This is not serious since the passability question is completely answered by gray-node evaluation based on the distance transform.

D. Results of the Staged Search

Figs. 7–10 depict the paths found by pure A^* search on the original quadtree, the first stage of staged search on the pruned quadtree (with gray nodes in the path), and the second stage of staged search (after paths inside the gray nodes are developed). The pruned quadtree used in the staged search is generated automatically, as discussed in the section on pruned quadtree generation. Each of the figures lists the cpu time taken for path planning, number of nodes considered by the search versus total number of leaf nodes, and details of the method of pruned quadtree generation used.

The path generated by the staged search is comparable to the optimal path generated by the pure A^* search. However, the total cpu time taken (with compiled Franz Lisp running on a VAX11/785) by staged search (for pruned quadtree generation, A^* search and second stage path development) is three to ten times less than that taken by the pure A^* search. (See Figs. 7–10) for detailed timings for the examples presented. The timings are in CPU seconds and involve substantial page swapping overhead.) Our experiments show that the computational savings are much higher for cluttered environments than for relatively free environments—compare Figs. 7 and 10, for example. This is reasonable since the fragmentation of free space is much higher in cluttered environments.

IV. Related Work

As pointed out in Section I, hierarchical representations have been used previously in manipulator findpath tasks. In this section we discuss some of that previous work in relation to our own.

Wong *et al.* [21] use a modified version of quadtrees to solve three-dimensional findpath problems by planning a path in the three orthogonal two-dimensional projections of the three-dimensional environment. Their approach essentially

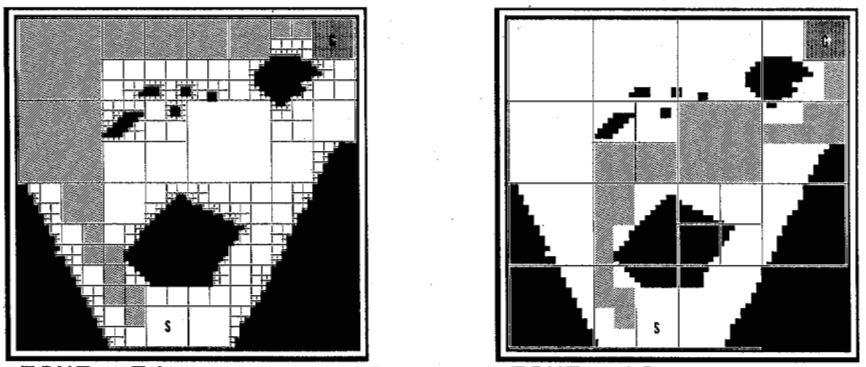


Fig. 7. Staged vs. single stage path planning (example 1). (a) Results of single stage planning. (b) Results of staged planning.

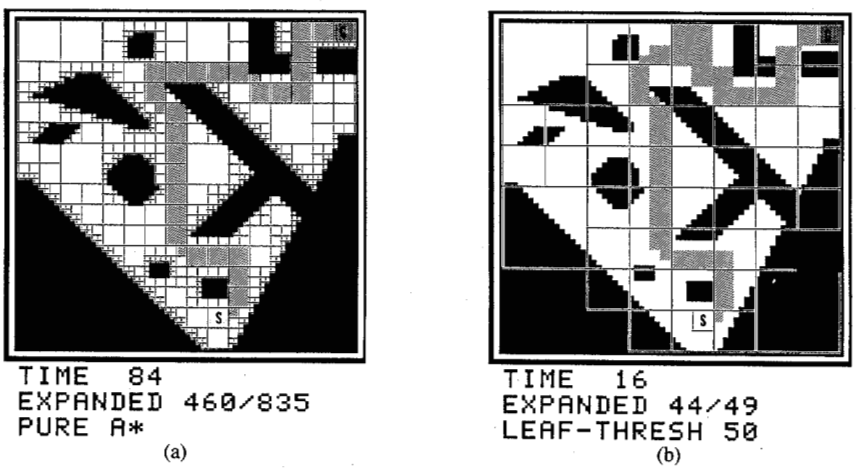


Fig. 8. Staged vs. single stage path planning (example 2). (a) shows the results of single stage planning and (b) shows the results of staged planning.

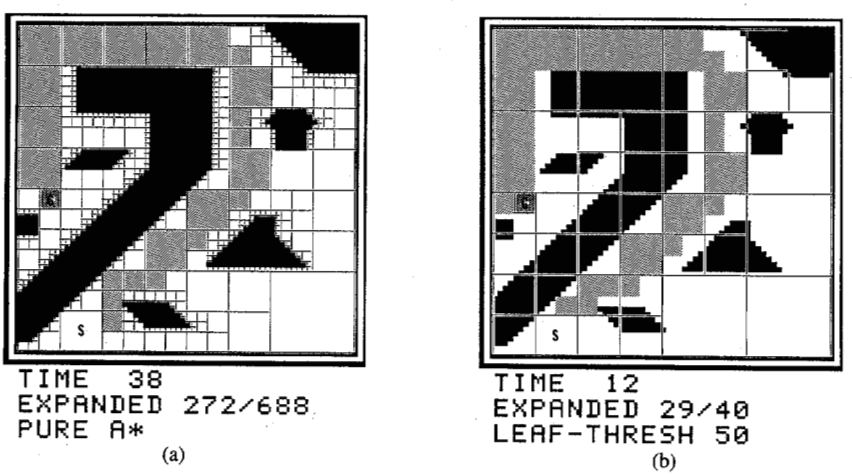


Fig. 9. Staged vs. single stage path planning (example 3). (a) shows the results of single stage planning and (b) shows the results of staged planning.

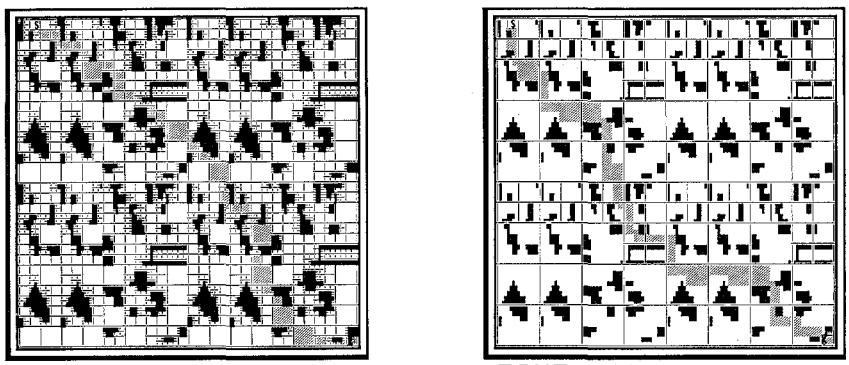


Fig. 10. Staged vs. single stage path planning (example 4). (a) Results of single stage planning. (b) Results of staged planning.

searches for a path in a "point-based" quadtree representation. (See [16] for a comparison between "region based" and "point based" quadtrees.) Faverjon [4] uses octrees (an extension of quadtrees to three-dimensions) for reducing the time complexity of the three-dimensional findpath problem for a six joint manipulator.

Lozano-Pérez [8] represented free space in the "configuration space" as a hybrid hierarchical structure consisting of rectanguloid and polyhedral cells. However, he planned a cell path strictly among the free cells of the representation, thus missing the computational advantages of hierarchical staged search. Another problem with his approach was that the path search could fail because the resolution of the representation was not fine enough. Brooks and Lozano-Pérez later remedied these problems in [2]. The approach presented in their paper comes closest to our "staged search" approach. They cut the free space hierarchically into full (obstacle), empty (free), and mixed rectanguloid cells, with the mixed cells representing areas of unexplored configuration space. They first try to plan a path exclusively through the free cells. If that fails, they then repeat the search, this time considering the mixed cells also. Next, for each mixed cell in the cell path, they try to develop a path through the mixed cell. If any of the mixed cells turns out to be impassable, then they may have to repeat the search again, finding another free-mixed cell path. Since they use the A^* search algorithm as the main engine for all these different searches, the overall process turns out to be very expensive computationally. Both [8] and [2] refine their cell paths into point paths, since the cell path in configuration space represents a set of possible solutions to the findpath problem.

V. Conclusion

In this paper we have presented methods of short range path planning for mobile robots, using quadtree hierarchical data structures. We demonstrated the merits of quadtree-based path planning and also discussed in detail a method of staged path planning with improved computational cost compared to pure quadtree based single stage path planning.

Lozano-Pérez [8] observes that the most important heuristic for a path-planning space representation is to avoid excess detail (and therefore time spent) on parts of the space that do not affect the planning operation. The quadtree representation naturally provides such a description of free space. Short-range planning for a mobile robot should be based on decomposition of free space into units larger than pixels for the search to be global. Hierarchical decompositions like the quadtree are a good way to achieve this, especially since the representation cost involved is small. The hierarchical decompositions may not be as descriptive as decomposing free space into channels or other such natural shapes, but the latter methods have a higher representation cost. Some of the suboptimality of uniform grid recursive decomposition involved in quadtree representation is offset by the staged version of the path planner.

Another important use of staged search in dynamic path planning is that it offers an elegant way of treating uncharted areas. These can be represented as gray nodes with very high cost, and when they get included in the search, further processing can be expended to "chart" those regions [12].

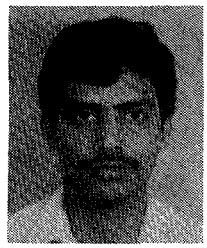
Looking further, the mobile robot needs to continually update the planned path, as it traverses it, in the light of new information. To do this efficiently, we need to be able to "add to" and "delete from" (or update) the representation of the image map with relatively low cost. In the context of dynamic path updating, one desirable property of a free-space representation is that the individual obstacles affect the representation only in their immediate locality. A disadvantage of quadtree representation of free space is that it does not localize the effect of obstacles on the representation. This is a general shortcoming of representations that cut free space into rectanguloid cells. In contrast, the generalized cone representation of free space described in [1] satisfies this property. One way of "adding to" the quadtree representation satisfying the localization property is discussed in [6] but this is not general enough to handle scroll updates.

References

[1] R. A. Brooks, "Solving the findpath problem by good representation of free space," *IEEE Trans. Syst. Man, Cybern.*, vol. 13, pp. 190–197, 1983.

[2] R. A. Brooks and T. Lozano-Pérez, "A subdivision algorithm in

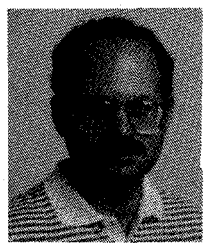
configuration space for findpath with rotation," in *Proc. Eighth Int. Joint Conf. Artificial Intelligence,* 1983.
[3] L. S. Davis, F. Andresen, R. Eastman, and S. Kambhampati, "Visual algorithms for autonomous navigation," in *Proc. IEEE Int. Conf. Robotics Automat.,* Mar. 1985.
[4] B. Faverjon, "Obstacle avoidance using an octree in the configuration space of a manipulator," in *Proc. IEEE Int. Conf. Robotics,* Mar. 1984.
[5] E. Horowitz and S. Sahni, *Fundamentals of Data Structures.* Rockville, MD: Computer Science, 1982, chap. 6.
[6] S. Kambhampati, "Multiresolution path planning for mobile robots," Masters thesis, Department of Computer Science, University of Maryland, College Park, 1985.
[7] T. Lozano-Pérez and M. A. Wesley, "An algorithm for planning collision-free paths among polyhedral obstacles," *Commun. ACM,* vol. 22, pp. 560–570, 1979.
[8] T. Lozano-Pérez, "Automatic planning of manipulator transfer movements," *IEEE Trans. Syst., Man, Cybern.,* vol. 11, pp. 681–698, 1981.
[9] H. Moravec, "Rover visual obstacle avoidance," in *Proc. Seventh Int. Joint Conf. Artificial Intell.,* 1981.
[10] N. J. Nilsson, "A mobile automation: an application of artificial intelligence techniques," in *Proc. First Int. Joint Conf. Artificial Intell.,* 1969.
[11] ——, *Principles of Artificial Intelligence.* Palo Alto, CA: Tioga, 1980, chap. 2.
[12] S. Puri and L. S. Davis, "Navigation algorithms for a quadtree based mobile robot system," Center for Automation Research, University of Maryland, College Park. Technical Report in Preparation.
[13] H. Samet, "An algorithm for converting rasters to quadtrees," *IEEE Trans. Patt. Analy. Mach. Intell.,* vol. 3, 1981, pp. 93–95.
[14] ——, "Distance transform of images represented by quadtrees," *IEEE Trans. Patt. Anal. Mach. Intell.,* 4, 1982, 298-303.
[15] ——, "Neighbor finding techniques for images represented by quadtrees," *Comput. Graphics Image Processing,* vol. 18, pp. 37–57, 1982.
[16] ——, "The quadtree and related hierarchical data structures," tech. rep. 23, Center for Automation Research, University of Maryland, College Park, Nov. 1983.
[17] H. Samet *et al.,* "Application of hierarchical data structures to geographical information systems: Phase III," tech. rep. 99, Center for Automation Research, University of Maryland, College Park, p. 59, Nov. 1984.
[18] A. M. Thompson, "The navigation system of the JPL robot," in *Proc. Fifth Int. Joint Conf. Artificial Intelligence,* 1977.
[19] C. Thorpe, "Path relaxation: path planning for a mobile robot," in *Proc. Nat. Conf. Artificial Intell.,* 1984.
[20] R. Wallace, "Two-dimensional path planning and collision avoidance for three-dimensional robot manipulators," in *Representation and Processing of Spatial Knowledge,* tech. rep. 1275, Department of Computer Science, University of Maryland, May 1983.
[21] E. K. Wong and K. S. Fu, "A hierarchical orthogonal space approach to collision-free path planning," in *Proc. IEEE Int. Conf. Robotics,* Mar. 1985.



Subbarao Kambhampati was born in Peddapuram, AP, India, on August 17, 1961. He received the B.Tech. degree in Electrical Engineering (Electronics) from the Indian Institute of Technology, Madras, in 1983 and the M.S. degree in Computer Science from the University of Maryland, College Park, in 1985.

He is currently a doctoral candidate in the department of Computer Science at the University of Maryland. Since June, 1984, he has been a research assistant in the Center for Automation Research there. His current research interests are Machine Learning and Robotics. He has authored three papers in Robotics.

Mr. Kambhampati is a member of the American Association for Artificial Intelligence and Association for Computing Machinery.



Larry S. Davis (S'74–M'77) was born in New York on February 26, 1949. He received the B.A. degree in mathematics from Colgate University, Hamilton, NY, in 1970, and the M.S. and Ph.D. degrees in computer science from the University of Maryland, College Park, in 1972 and 1976, respectively.

From 1977–1981 he was an Assistant Professor in the Department of Computer Science, University of Texas, Austin. He is currently an Associate Professor and Associate Chairman in the Department of Computer Science, University of Maryland. He is also the Head of the Computer Vision Laboratory of the Center for Automation Research at the University of Maryland and is the Acting Director of the University of Maryland Institute for Advanced Computer Studies.