A Reprint from the

PROCEEDINGS

Of SPIE-The International Society for Optical Engineering

Volume 635

Applications of Artificial Intelligence III

April 1-3, 1986
Orlando, Florida

Fast path planning in unstructured, dynamic, 3-D worlds

Martin Herman
Robot Systems Division
National Bureau of Standards, Gaithersburg, Maryland 20899

Fast Path Planning in Unstructured, Dynamic, 3-D Worlds

Martin Herman

Robot Systems Division
National Bureau of Standards
Gaithersburg, MD 20899

ABSTRACT

Issues dealing with fast motion planning in unstructured, dynamic 3-D worlds are discussed, and a fast path planning system under development at NBS is described. It is argued that an octree representation of the obstacles in the world leads to fast path planning algorithms. The system we are developing performs the path search in an octree space, and uses a hybrid search technique that combines hypothesize and test, hill climbing, A^*, and multiresolution grid search.

1. Introduction

In order to perform collision-free robot path planning, the following kinds of knowledge about the robot's world are relevant: (1) descriptions of the objects in the world, (2) the positions and orientations of these objects at some point in time, and (3) the motions of these objects as a function of time. In this section, we introduce some issues dealing with path planning in unstructured, dynamic environments.

A program that automatically generates collision-free paths must be able to represent the robot's world for two purposes, for accumulating the world description from outside sources, and for performing the path search.

The representation in which the world description is accumulated is called the ***world representation.*** This description may be obtained from a combination of various sources, including manual input, a priori object data bases, sensory recognition modules, and sensory description modules. Common forms for such representations are surface-based CAD models [Baer et al. 79, Requicha 80], swept volumes [Brooks 81], cellular arrays [Srihari 81], octrees [Meagher 82, Jackins & Tanimoto 80], and analytic surface equations.

The representation in which the search for paths is performed is called the ***search space representation.*** It may be either the same as or different from the world representation. Examples of search space representations that are usually different from world representations are configuration spaces [Lozano-Perez 81], Voronoi-based spaces [Canny 85], generalized cylinder free spaces [Brooks 83], and medial axis free spaces [Ruff & Ahuja 84]. If the search space representation is different from the world representation, a procedure that maps the world to the search space is required.

The world may be either static or dynamic. A dynamic world is one in which the poses of objects change over time, while a static world is one in which only the pose of the robot changes. In order to find and maintain collision-free trajectories in a dynamic world, the search space must be updated as changes occur. It is usually more efficient to perform this updating incrementally, rather than regenerating the whole search space whenever a small part of it has changed.

An ***unstructured*** world is one in which there are unknown objects, unknown poses of objects, or unknown motions of objects. In such cases, descriptions of portions of the world may have to be obtained using sensory processing before path planning can be performed.

Because path planning in unstructured, dynamic worlds involves situations which are not known beforehand, the path planning must be done online. This means that algorithms for acquiring the world description from sensors, for mapping the world description into a search space description (if the two are not the same), and for performing the path search must be fast.

2. Previous Work

In this paper, we consider some issues related to fast path planning in an unstructured, dynamic, 3-D environment. The term "fast" is used informally here to specify a practical time frame in which the robot can plan and execute motions. This paper suggests ways of obtaining speed, but it is not yet clear how fast the final algorithms will be.

Most previous path planning algorithms have operated in two dimensions, often for mobile robot applications. Many of these assume that rotation of the robot can be ignored, and consider translation only. The speed of these algorithms would probably decrease dramatically if applied to the extra degrees of freedom in three-dimensional motion.

Many of the planning algorithms developed for 3-D motion are inadequate for our purposes. The configuration space approach, for example, seems to be computationally very expensive. It requires, first, mapping a world description into a configuration space, i.e., generating the configuration space obstacles [Lozano-Perez 81]. In general, this step is very time consuming. Further, an algorithm for doing this incrementally has not yet been developed. Second, the search must be performed in a high-dimensional space. The explicit representation of the high-dimensional space can consume a large amount of memory, although the technique of slice projections [Lozano-Perez 81] reduces the memory requirements. Of course, searching a high-dimensional space can be very time consuming, but most approaches share this problem, whether they represent the space explicitly or implicitly.

The approach of explicitly representing free space with generalized cylinders [Brooks 83] seems to offer some speed in performing path planning. However, although the technique works well in 2-D, it is difficult to generalize to 3-D. It has been used in 3-D by performing path searches through 2-D slices of 3-D space [Brooks 83]. Unfortunately, this technique is useful only in limited situations, such as pick and place operations on a horizontal table.

The potential field approaches [Khatib 85, Buckley & Leifer 85] have rather limited path planning capabilities if used only by themselves. They suffer from being "too dumb"; they often get stuck at local minima in the potential field. However, when combined with smarter path planning algorithms, they should prove very useful because they offer possibilities for real-time obstacle avoidance. We will discuss this further in Section 10.3.

3. Using Octrees for Path Planning

In this paper, we will show how an octree representation of the world leads to fast path planning. An octree [Meagher 82, Jackins & Tanimoto 80] is a recursive decomposition of a cubic space into subcubes. Initially, the whole space is represented by a single node in the tree, called the root node. If the cubic volume is homogenous (completely filled by an object or completely empty), then the root is not decomposed at all, and comprises the complete description of the space. Otherwise, it is split into eight equal subcubes (octants), which become the children of the root. This process continues until all the nodes are homogeneous, or until some resolution limit is reached. Nodes corresponding to cubic regions that are completely full are called FULL leaf nodes. Nodes corresponding to empty regions are called EMPTY leaf nodes, and nodes corresponding to mixed regions (non-leaf nodes) are called MIXED nodes.

The techniques for path planning described here assume that octrees are used to represent Cartesian 3-space, and that path planning occurs in this space. The following properties of octrees lead to fast path planning algorithms:

1. Octrees provide a spatially-indexed representation of the world. That is, associated with each region of 3-space is a list of objects within the region. Therefore, the objects at each point or region in space can be very quickly retrieved. This leads to very fast collision detection algorithms. During path planning, if the hypothesized motion of an object is represented as the volume the object would sweep out, potential collisions can very quickly be found.
2. Ideally, the search for a path should be performed in a continuous search space. Of course, this potentially leads to an infinite number of paths to be considered during search. Octrees provide a decomposition of free space into cubes, each of which can be treated as a single symbolic unit (i.e., a node) in a search graph. Links are created between two nodes only if their respective cubes are adjacent. In this way, the infinite search space is converted into a finite one.
3. The hierarchical, multiresolution nature of octrees may be utilized to improve the speed of search algorithms [Kambhampati & Davis 85]. The idea is to represent primarily octants at a low resolution level as nodes in the search graph. High resolution octants are represented only when necessary. The resulting graph is much smaller and search proceeds more quickly.
4. Efficient algorithms exist for converting a polyhedral object described by its surfaces into an octree description [Hong 85].
5. Efficient algorithms exist for incrementally modifying octrees [Hong & Shneier 85b, Weng & Ahuja 85]. These techniques assume that a separate database of objects in the world exists. Each object in this database has an octree representation in the object's coordinate system. Incremental modifications to the world octree then consist of rotating, translating, adding, and deleting the object octrees.
6. Finally, octrees are often useful for tasks other than path planning. Because they offer a useful representation of 3-space, they can serve as the output representation for sensory interpretation algorithms [Hong & Shneier 85a, Connolly 84], they can be used to retrieve objects or object features lying in a given region of space, or to solve the hidden feature problem for verification vision or graphics display [Glassner 84, Meagher 82]. The point is that in a complete robot planning, control, and sensory system, octrees may serve in many different kinds of tasks [Shneier et al. 84]. The effective cost of generating the octrees thus becomes lower when compared to the cost in systems that must generate a different description for each task.

The primary disadvantages of octrees are, first, that they do not provide an exact representation of objects and, second, that they tend to require a lot of memory. The first disadvantage can be overcome by using an object's surface-based description when highly precise motions near objects are required. Since the nodes of the octree have pointers to objects contained in them, retrieving the objects in a given region of space is very fast.

The second disadvantage is more difficult to overcome, but techniques such as dynamically expanding the octree into higher resolution levels only when needed, or compressing the octree representation [Gargantini 82], may help. Although spatial decompositions that are irregular [Reddy & Rubin 78, Lozano-Perez 81] may result in smaller trees, operations on them such as locating volume elements, finding their positions, performing translation and rotation, and generating them from surface-based object descriptions are usually much slower.

4. NBS Path Planning System

The remainder of this paper describes a fast, three-dimensional, path planning system under development at NBS. The current implementation assumes that the robot's external world is static and structured. We plan eventually to extend the system to unstructured and dynamic worlds by incorporating sensory processing components.

The inputs to the path planner are (1) a description of the robot, (2) a description of the robot's external world, in the form of a single world octree, (3) the configurations of the robot in the start and goal states. The output of the path planner is a piecewise linear path in 3-space. Although pure translation is currently assumed, methods for incorporating rotation will be included in our discussion.

The path generated is always guaranteed to be collision-free, although it is generally not the shortest such path between the start and goal states. Finding the shortest path requires an expensive search. Fortunately, a "reasonably" short path is adequate for many tasks, and such a path can often be found quite quickly.

5. Searching for a Path

Several different search techniques are combined to perform the search through the octree space. The first is hypothesize and test, and involves hypothesizing a simple path for the robot by generating the volume it would sweep out during the motion. Using an algorithm to be described below, a collision between the swept-out volume and an object in the octree can very quickly be detected. Two kinds of simple paths have been considered: linear paths (corresponding to simple translations) and circular paths (corresponding to simple rotations). Any complex path can be approximated by an intermixed sequence of these two simple paths. These paths are hypothesized by the other search techniques, described next.

The second search technique is hill climbing. It uses a cost function whose value at any point in free space is proportional to the Euclidean distance from the point to the goal, and whose value at any point inside or on the surface of an object is infinite. The robot is then always made to move to a neighboring point whose cost is the minimum over all neighboring points. This search technique is very fast because (1) only information local to each robot position is used in deciding in which direction to move next, and (2) only a single path leading to the robot's current position is remembered; alternative paths are not remembered and therefore not considered. This technique is similar to the potential field technique mentioned earlier, and suffers from the same major problem: the algorithm can easily get stuck at a local minimum in the cost function, that is, a point that has a lower cost than any of its neighboring points.

The third technique we use is A^* search [Nilsson 71] -- a best-first, tree-structured search method. The technique is applied to a graph representation of the octree search space, and it performs a global search through the graph. A search tree is built up as the search progresses so that the algorithm can always proceed with the path with lowest cost. Portions of many alternative paths may therefore be explored before a solution path is finally found. A^* search is therefore more computationally expensive (on average) than hill climbing.

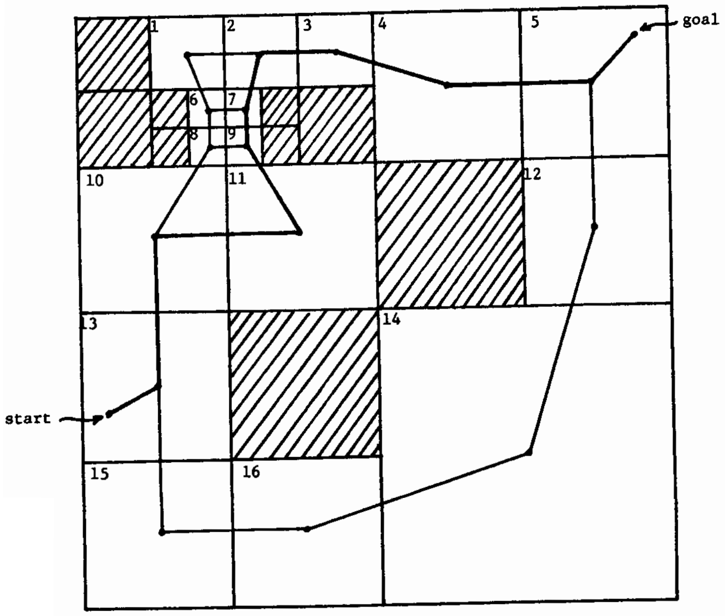


Figure 1: Initial search graph in which hill climbing and A^* searches are performed. The start and goal positions are shown in blocks 13 and 5, respectively. Hill climbing initially finds a path through blocks 13, 10, and 11. A^* is then invoked and finds a path through blocks 11, 9, 7, and 2. Hill climbing is then reinvoked and finds a path through blocks 2, 3, 4, and 5. As indicated in the text, a swept-volume path will actually be generated from block 2 to the goal, thus eliminating the need to use hill climbing from blocks 2 to 5. Also, due to multiresolution grid search, the actual final path will not necessarily proceed from center to center of blocks as shown here. See Fig. 2 for how the path might be altered.

However, the minimal cost path in the search graph is always found if $\hat{h}$, the heuristic value of the cost function at any given point, is always less than the actual mimimal cost path from the point to the goal. A search algorithm with this property is said to be ***admissible*** [Nilsson 71]. In the current implementation, the value of $\hat{h}$ at a point is the Euclidean distance from the point to the goal.

The fourth technique we use is multiresolution grid search. This technique offers a way of searching at a finer resulution than that of the octree search space. A high resolution grid is placed within the octants of the octree, and this grid is searched in a multiresolution fashion. More details will be described below.

All the techniques described above are combined in an attempt to achieve the greatest speed in finding free paths in a variety of world configurations. The search is performed on a graph initially obtained by connecting the centers of all adjacent EMPTY leaf octants in the octree. Fig. 1 shows the appearance of such a graph for a quadtree example. The graph is not explicitly created before the search begins; it is implicitly formed as needed during the search process.

The basic operation of the combined search algorithm is as follows. Beginning at the start state, hill climbing search is performed. If a local miminum is reached, A^* search is invoked, beginning at the point at which hill climbing got stuck (see Fig. 1). The purpose of the A^* search mode is to get out of the valley around this local minimum and over a peak or ridge. At this point, hill climbing may be reinvoked because the robot cannot return to the position at which the local minimum occurred. The process of switching between the two search modes continues until either the goal is reached, or it is determined that no path to the goal exists. The two search modes provide hypothesis paths for the hypothesize and test technique described earlier. This is done in two ways. First, at every node reached during the search, a linear swept volume for the robot is generated from the node to the goal. If this volume does not intersect an obstacle, a solution is obtained. Otherwise, the search proceeds as described above.

Second, in determining whether or not a path from one node to an adjacent node in the graph is valid, a linear swept volume is generated between the two node positions, which are initially assumed to be at the centers of the two octants. If no collision with an obstacle is detected, the path is valid. Otherwise, there still may be a path between the two octants, although the path may not be linear and/or it may not go from center to center. At this point, the multiresolution grid

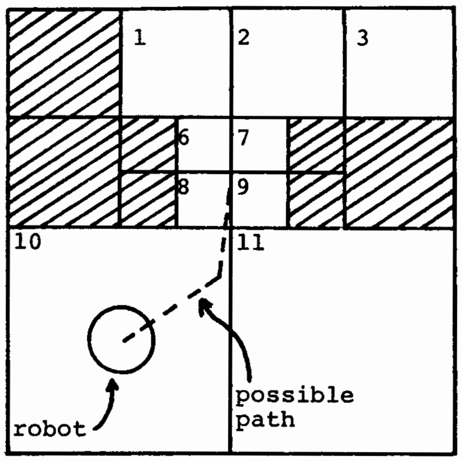


Figure 2: Using multiresolution grid search.

search mentioned above is invoked. A free path for the robot may be found by searching on a high-resolution grid placed inside the two octants. A hill climbing search from center to center of the two octants is performed on this grid in a multiresolution fashion. An example of this is shown in Fig. 2, which depicts block 1-3, 6-9, 10, and 11 of Fig. 1. The robot shown in block 10 cannot move to the center points of either blocks 8 or 9 without a collision. In order to navigate through the passage formed by blocks 6-9, the robot must move to a non-center point in one of these blocks. Furthermore, the robot may not be able to perform this move along a single linear path; it may require a piecewise linear path such as that indicated by the dashed line in Fig. 2. The high-resolution grid placed inside the octants provides the ability to find such a path.

6. Obtaining Successor Nodes During Search

As described above, the search graph is implicitly formed as needed during the search. This occurs by dynamically finding the neighbors, or *successors* [Nilsson 71], of a node in the graph when it is visited. Let n be the current node visited. A successor node, s_i , is a point contained in an EMPTY octant adjacent to the octant represented by n. Two octants are adjacent if a face of one lies against a face of the other. Let $d(m)$ be the cost at any node m (for hill climbing). To obtain a successor node during hill climbing mode, the following steps are taken.

1. Obtain all s_i such that
   - (a) s_i represents a leaf octant that is adjacent to the octant of n,
   - (b) the octant of s_i is EMPTY, and
   - (c) $d(s_i) < d(n)$, where $d(s_i)$ is measured at the center of the octant of s_i.
2. If the set $\{s_i\}$= NULL, a local minimum has been reached at node n, so exit.
3. Otherwise, set p to the potential successor s_j with minimum d.
4. If there is a free path from the octant of n to the octant of p, p is the successor node. The technique used to find such a free path involves multiresolution grid search, described in detail below.
5. Otherwise, remove p from $\{s_i\}$, and go to step 2.

During A^* search mode, there will, in general, be more than one successor for each node. A search tree is generated during the search wherein the successors of each node form the children in the tree. The node s_i is a successor of node n if

1. s_i represents a leaf octant that is adjacent to the octant of n,
2. the octant of s_i is EMPTY,
3. successors of s_i have not yet been obtained during the current invocation of A^*,**
4. there is a free path from the octant of n to the octant of s_i. (Again, this is obtained using multiresolution grid search; see below.)

7. Multiresolution Grid Search

This section describes the algorithm used in multiresolution grid search. Suppose n_1 and n_2 are two search nodes representing the adjacent leaf octants O_1 and O_2, respectively. We say that n_2 is a *valid successor* of n_1 if a free path for the robot exists from the initial point in O_1 to some point in O_2, and the path lies completely within the two octants. Let $\vec{p}_s$ be the initial starting point in O_1, and let $\vec{p}_c$ be the center point of O_2. Let $d(\vec{a},\vec{b})$ be the distance between two points $\vec{a}$ and $\vec{b}$. The following algorithm determines whether n_2 is a valid successor of n_1. If it is, the points on the path between the two octants is returned in the list *pathpoints*. The algorithm seeks the path using hill climbing search on a multiresolution grid.

1. Set $\vec{p}$ to $\vec{p}_s$.
2. Place $\vec{p}$ in *pathpoints*.
3. If there is a linear free path from $\vec{p}$ to $\vec{p}_c$, then n_2 is a valid successor of n_1. So add $\vec{p}_c$ to *pathpoints* and exit.
4. Otherwise, obtain the translation vector $\vec{T}$ from $\vec{p}$ to $\vec{p}_c$. Then set the grid increment value I to the maximum of $|\vec{T}_x|$, $|\vec{T}_y|$, and $|\vec{T}_z|$. This is the coarsest resolution level of the grid used in finding a neighbor of point $\vec{p}$. Set M to the (a priori) minimum allowable grid increment. This is the finest resolution level of the grid.
5. If $I < M$, go to step 12.
6. Otherwise, obtain the set $\{\vec{q}_i\}$ of all neighboring points of $\vec{p}$ by incrementing in the x, y, and z directions by the value I. This step dynamically generates the multiresolution grid. The six points in $\{\vec{q}_i\}$ are

   $$\vec{q}_1=\vec{p}+I\hat{x}, \quad \vec{q}_2=\vec{p}-I\hat{x},$$
   $$\vec{q}_3=\vec{p}+I\hat{y}, \quad \vec{q}_4=\vec{p}-I\hat{y},$$
   $$\vec{q}_5=\vec{p}+I\hat{z}, \quad \vec{q}_6=\vec{p}-I\hat{z}.$$
7. Remove from $\{\vec{q}_i\}$ any point $\vec{r}$ such that
   - (a) $\vec{r}$ is not inside O_1 or O_2, or
   - (b) $d(\vec{r},\vec{p}_c) \geq d(\vec{p},\vec{p}_c)$.
8. If $\{\vec{q}_i\}$= NULL, then a local minimum has been reached, i.e., no neighboring point of $\vec{p}$ (at the current grid resolution) is closer to $\vec{p}_c$ than $\vec{p}$. So set I to $I/2$, and go to step 5. The current step has determined that the current grid resolution is too coarse. The grid increment is therefore halved, and neighboring points of $\vec{p}$ are again sought, but on a finer resolution grid.
9. Otherwise, choose the point $\vec{r}$ in $\{\vec{q}_i\}$ with minimum $d(\vec{r},\vec{p}_c)$.
10. If there is not a linear free path from $\vec{p}$ to $\vec{r}$, remove $\vec{r}$ from $\{\vec{q}_i\}$, and go to step 8.
11. Otherwise, there is a linear free path from $\vec{p}$ to $\vec{r}$. If $\vec{r}$ is inside O_2, then n_2 is a valid successor of n_1. So add $\vec{r}$ to *pathpoints* and exit. We do not need to extend the path further towards $\vec{p}_c$. If $\vec{r}$ is not inside O_2, then set $\vec{p}$ to $\vec{r}$, and go to step 2. An attempt must be made to further extend the path toward $\vec{p}_c$.
12. At this point, the value of the grid increment I is less than M, the finest resolution level. This implies that a free path to O_2 has not been found. So n_2 is not a valid successor of n_1. Exit with failure.

8. Primitive Shapes

Thus far, we have discussed how the objects in the robot's world are represented in the form of a single world octree. In order to detect potential collisions, the swept volume representing the robot's path must also be represented.

**Since the *consistency assumption* for $\hat{h}$ is satisfied [Nilsson 71], a node need never be reexamined once its successors have been obtained.

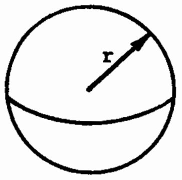
In our approach, the articulate parts of the robot, as well as the swept-volume paths of the robot, are approximated by a set of primitive shapes. Because the robot is always fully contained in its approximating shapes, a free path for the shapes is always a free path for the robot. The converse is not true.

There are three requirements for defining a primitive shape. The first is that computing whether or not the shape intersects an object in the octree should be fast. Our primitive shapes are therefore defined in terms of a ***spine*** -- either a simple curve or simple surface segment -- and a ***radius*** -- a single extent outward from the spine that defines the shape's surface. By representing octants in the octree as spheres, the intersection test merely involves determining the shortest distance from the center of a sphere to the spine of the primitive shape, and checking whether or not this distance exceeds the sum of the radii of the sphere and shape. More of this will be described below.

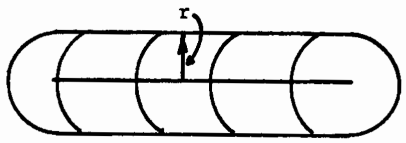
The second requirement for a primitive shape is that it should be a reasonable approximation to a part of the robot or a swept volume. The third requirement is that generating primitive shapes used to represent swept-volume paths should be very fast. This is because the particular shape must be dynamically generated during search. Many of the shapes are therefore defined as translational or rotational sweeps of some other primitive shape.

Some primitive shapes we have considered are the following:

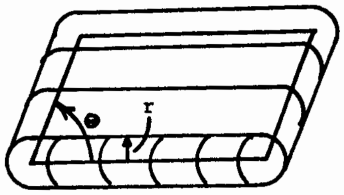
1. Sphere (Fig. 3a). Defined by a center point and a radius.
2. Cylsphere (Fig. 3b). The volume swept out by linear translation of a sphere. Defined by the radius of the sphere and the two end points of the line segment forming the spine.
3. Translation-swept cylsphere (Fig. 3c). The volume swept out by linear translation of a cylsphere. Defined by the radius of the cylsphere and the four end points of the parallelogram forming the spine.
4. Rotation-swept cylsphere. The volume swept out by rotation of a cylsphere about an axis intersecting and perpendicular to its spine. There are two types. For type 1 (Fig. 3d), the rotation angle is less than 180 degrees; for type 2 (Fig. 3e), the angle is greater than 180 degrees. Type 1 is defined by the radius of the cylsphere and the two wedge slices meeting at their apexes that form the spine. Type 2 is defined by the radius of the cylsphere and the pie segment that forms the spine.
5. Torus section (Fig. 3f). The volume swept out by rotation of a primitive shape about an axis that does not intersect the shape.



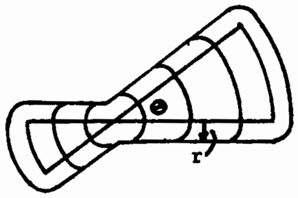
(a) Sphere



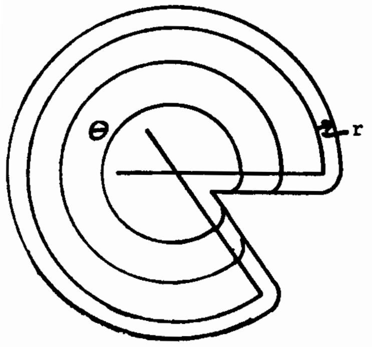
(b) Cylsphere



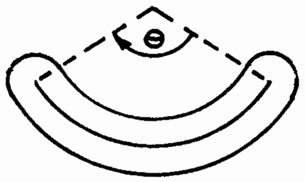
(c) Translation-swept cylsphere



(d) Rotation-swept cylsphere of type 1



(e) Rotation-swept cylsphere of type 2



(f) Torus section

Figure 3: Some primitive shapes.

9. Collision Detection

The following algorithm is used to determine whether or not a primitive shape intersects any obstacles represented in the world octree. First, associated with each swept-volume primitive shape is a set of curves within its volume that follow the sweep used to form the shape. If the shape is formed by translational sweep, the curves are straight line segments (Fig. 4a). If the shape is formed by rotational sweep, the curves are circular arc segments (Fig. 4b). The curves within the volume of the shape are individually tested to see if any intersects an obstacle. This test is extremely fast [Glassner 84]. If there is an intersection, then of course the shape also intersects the obstacle. If there is no intersection, a more detailed test must be performed on the shape, for some other part of the shape may still intersect an obstacle.

The more detailed test involves performing a breadth-first traversal of the octree, and checking for an intersection between each FULL node and the primitive shape. Using a breadth-first, rather than depth-first, traversal insures that if there is a FULL node at a low resolution level that intersects the shape, it will be found quickly, before much of the rest of the tree is examined.

To avoid visiting and checking for intersection with too many unnecessary nodes in the octree, the highest resolution octant totally containing the bounding box of the primitive shape is initially found (Fig. 5). The breadth-first traversal then occurs within the subtree rooted at this octant. In addition, the children of a MIXED node are not visited unless the node overlaps the bounding box of the primitive shape.

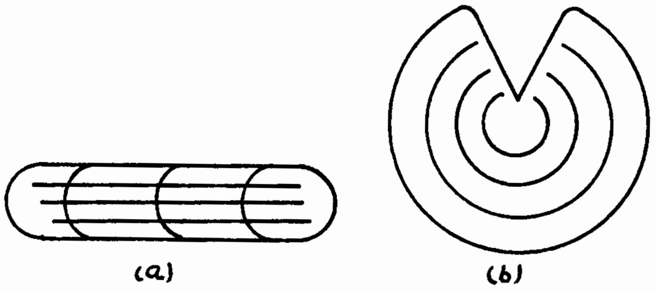


Figure 4: (a) Straight-line segments inside volume. (b) Circular arc segments inside volume.

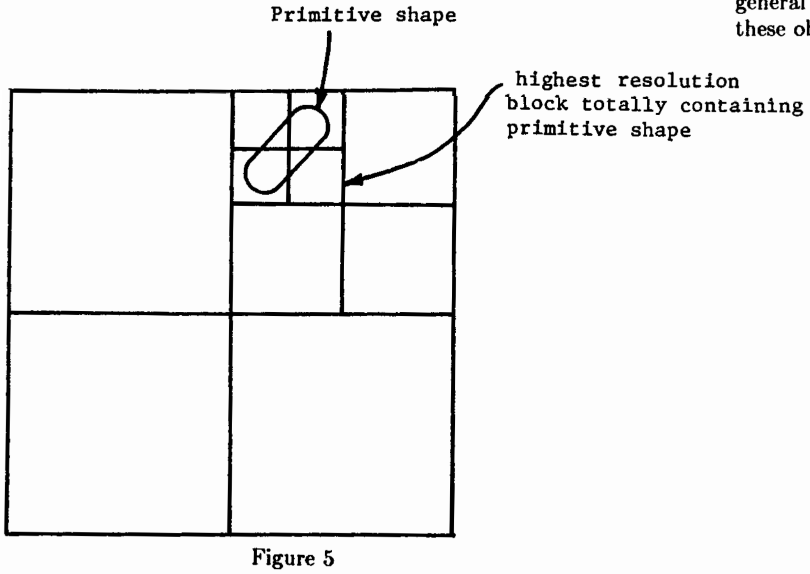


Figure 5

When a FULL node is reached during the traversal, the following tests are performed.

1. If the octant and bounding box of the primitive shape do not overlap, there is no intersection.
2. Let us define the ***outer sphere*** as the smallest sphere that completely contains the octant, and the ***inner sphere*** as the largest sphere contained completely within the octant (Fig. 6). If the primitive shape intersects the inner sphere, there is an intersection with the octant [Hong & Shneier 85b].
3. If the shape does not intersect the outer sphere, there is no intersection with the octant.
4. If the shape intersects the outer but not the inner sphere, then
   (a) if the octant is at the highest resolution level, assume an intersection,
   (b) otherwise, divide the octant into 8 suboctants, and for each suboctant, proceed from step 1.

10. Improving the System

This section discusses three ways in which our current system can be improved. The first two involve obtaining greater speed using the approaches of knowledge-based path planning and multiresolution search. The third involves dealing with dynamic worlds.

10.1. Knowledge-Based Path Planning

The system described here is very general in the sense that it uses a search algorithm that is independent of the task domain. For example, hill climbing and A^* are general search techniques. The system could be speeded up considerably in many domains if task-specific knowledge were used. One method involves using such knowledge to specify intermediate points on the path, or to specify intermediate directions in which to search. For example, many pick and place tasks can be satisfactorily carried out by a robot by moving up vertically (after grasping the object), then moving across horizontally, and then down vertically. If intermediate directions in the motion were specified (i.e., first move up, then over, then down), the search process would often be much faster than a general search from start to goal. This is because there are often more obstacles on the table than high above it, and the general search approach would try to find a path in between these obstacles.

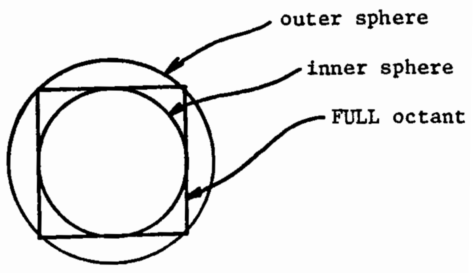


Figure 6

10.2. Multiresolution Search

It is often useful to impose a multiresolution structure on the search space and then apply a search method that takes advantage of this structure. Such a search method generally consists of searching at a coarse resolution level whenever feasible, and searching at a fine level only when necessary. The advantage of this is that there are fewer search-space elements at the coarse level, resulting in faster search time.

In the context of our robot path planning system, a multiresolution structure can be imposed on several components of the search space, including the following.

1. Translation space. This is the space of all possible x, y, z positions of the robot. Two types of multiresolution structures may be used here. The first is the octree itself, and the second is the multiresolution grid placed within the leaf octants, as described earlier. These two structures provide a multiresolution subdivision of 3-space [Kambhampati & Davis 85].
2. Rotation space. This is the space of all possible rotational positions of the robot. A technique similar to the multiresolution grid can be applied here, wherein increments between angular positions of the robot are dynamically determined during the search process in a multiresolution fashion.
3. Robot description. The robot description is used to hypothesize swept-volume paths which are tested during search. The robot may be described in a multiresolution fashion, wherein coarse levels of description involve fewer articulate parts, fewer primitive shapes, and fewer degrees of freedom in motion than fine levels [Marr & Nishihara 78].

In regions where the robot is far from obstacles, the path search may be performed at coarse levels of these various multiresolution structures. This means that the search process need only consider (1) low resolution EMPTY octants of the octree (high resolution EMPTY octants may be treated as FULL), (2) coarse grid increments in the multiresolution grid, (3) coarse rotational increments in rotation space, and (4) coarse levels of the robot description. The search needs to consider fine levels of the multiresolution structures only when the robot is moving near obstacles.

10.3. Dynamic Worlds

There are two important issues when dealing with dynamic worlds. The first is how to find out that a change in the world has occurred. The second is how to modify the robot's motion to accomodate the change. The first issue involves sensory interpretation, a very large topic which will not be discussed here. The second issue involves many topics, but a very important one is real-time collision avoidance between the robot and a moving object. After the collision has been avoided, either the initial path can be continued or a new path can be generated. The continuation or regeneration of a path need not occur in real time, although of course speed is important.

Real-time collision avoidance based on the potential field concept has been demonstrated to be quite effective [Khatib 85]. This type of technique is useful for avoiding imminent collisions. However, to avoid potential collisions further along the planned path, the collision detection algorithms described earlier in this paper can be used. If the planned path is stored in terms of swept volumes of the robot's motion, then any object that moves into the path can be detected, and appropriate path changes can be sought well in advance of the collision point.

Acknowledgements

The following people have greatly contributed to this work through comments and discussions: Tony Barbera, John Fiala, Len Haynes, Tsai Hong, Ted Hopp, Ernie Kent, Ron Lumia, Graham Morris, Don Myers, Mike Shneier, and John Wilkes.

References

1. Baer, A., Eastman, C., and Henrion, M. "Geometric modelling: a survey." *Computer-Aided Design,* 11, 1979, 253-272.
2. Brooks, R.A. "Symbolic reasoning among 3-D models and 2-D images." *Artificial Intelligence,* 17, 1981, 285-348.
3. Brooks, R.A. "Planning collision-free motions for pick-and-palce operations." *The International Journal of Robotics Research,* Vol. 2, No. 4, Winter 1983, 19-44.
4. Buckley, C.E. and Leifer, L.J. "A proximity metric for continuum path planning." *Proc. Ninth International Joint Conf. on Artificial Intelligence,* Los Angeles, CA, August 1985, 1096-1102.
5. Canny, J. "A Voronoi method for the piano-movers problem." *IEEE International Conf. on Robotics and Automation,* St. Louis, Missouri, March 1985, 530-535.
6. Connolly, C.I. "Cumulative generation of octree models from range data." *Proc. IEEE International Conf. on Robotics,* Atlanta, GA, March 1984, 25-32.
7. Gargantini, I. "Linear octtrees for fast processing of three-dimensional objects." *Computer Graphics and Image Processing,* 20, 1982, 365-374.
8. Glassner, A.S. "Space subdivision for fast ray tracing." *IEEE Computer Graphics and Applications 4* 10, October 1984, 15-22.
9. Hong, T.-H. Personal communication, 1985.
10. Hong, T.-H. and Shneier, M.O. "Incrementally constructing a spatial representation using a moving camera." *Proc. IEEE Conf. on Computer Vision and Pattern Recognition,* San Francisco, CA, June 1985a, 591-596.
11. Hong, T.-H. and Shneier, M.O. "Rotation and translation of objects represented by octrees." Robot Systems Division, National Bureau of Standards, Gaithersburg, MD, October 1985b.
12. Jackins, C.L. and Tanimoto, S.L. "Oct-trees and their use in representing three-dimensional objects." *Computer Graphics and Image Processing 14,* 1980, 249-270.

13. Kambhampati, S. and Davis, L. S. "Multiresolution path planning for mobile robots." Technical Report 127, Center for Automation Research, University of Maryland, College Park, MD, May 1985.

14. Khatib, O. "Real-time obstacle avoidance for manipulators and mobile robots." *IEEE International Conf. on Robotics and Automation,* St. Louis, Missouri, March 1985, 500-505.

15. Lozano-Perez, T. "Automatic planning of manipulator transfer movements." *IEEE Trans. on Systems, Man, and Cybernetics,* Vol. SMC-11, 1981, 681-698.

16. Marr, D. and Nishihara, H.K. "Representation and recognition of the spatial organization of three-dimensional shapes." *Proc. Royal Society of London B 200,* 1978, 269-294.

17. Meagher, D. "Geometric modeling using octree encoding." *Computer Graphics and Image Processing,* 19, 1982, 129-147.

18. Nilsson, N.J. *Problem-Solving Methods in Artificial Intelligence.* McGraw-Hill, New York, 1971.

19. Reddy, D.R. and Rubin, S. "Representation of three-dimensional objects." Technical Report CMU-CS-78-113, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, 1978.

20. Requicha, A.A.G. "Representations for rigid solids: theory, methods, and systems." *Computing Surveys,* 12(4), 1980, 437-464.

21. Ruff, R. and Ahuja, N. "Path planning in a three dimensional environment." *Proc. Seventh International Conf. on Pattern Recognition,* Montreal, Canada, July 1984, 188-191.

22. Shneier, M., Kent, E., and Mansbach, P. "Representing workspace and model knowledge for a robot with mobile sensors." *Proc. Seventh International Conf. on Pattern Recognition,* Montreal, Canada, July 1984, 199-202.

23. Srihari, S.N. "Representation of three-dimensional digital images." *Computing Surveys,* Vol. 13, No. 4, 1981, 399-424.

24. Weng, J. and Ahuja, N. "Octree representation of objects in arbitrary motion." *Proc. IEEE Cobference on Computer Vision and Pattern Recognition,* San Francisco, CA, June 1985, 524-529.