To the Graduate Council:

I am submitting herewith a thesis written by A. K. Hyder Ali entitled "A 2-D AND 3-D ROBOT PATH PLANNING ALGORITHM BASED ON QUADTREE AND OCTREE REPRESENTATION OF WORKSPACE." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

R. C. Gonzalez

R. C. Gonzalez, Major Professor

We have read this thesis
and recommend its acceptance:

Donald W. Bouldin

Marshall O. Pace

Accepted for the Council:

Lew Minkel

Vice Provost
and Dean of The Graduate School

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a Master's degree at The University of Tennessee, Knoxville, I agree that the Library shall make it available to borrowers under rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of the source is made.

Permission for extensive quotation from or reproduction of this thesis may be granted by my major professor, or in his absence, by the Head of Interlibrary Services when, in the opinion of either, the proposed use of the material is for scholarly purposes. Any copying or use of the material in this thesis for financial gain shall not be allowed without my written permission.

Signature Hyder Ali

Date 11/23/87

A 2-D AND 3-D ROBOT PATH PLANNING ALGORITHM BASED ON QUADTREE AND OCTREE REPRESENTATION OF THE WORKSPACE

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

A. K. Hyder Ali

December 1987

ACKNOWLEDGMENTS

I would like to thank everyone who helped in the research and preparation of this thesis. I especially thank my major professor, Dr. R. C. Gonzalez, for giving me the privilege of working with him. His advice and encouragement throughout my graduate career have been invaluable.

I would also like to thank Drs. D. W. Bouldin, M. O. Pace, and M. A. Abidi for serving on my committee and teaching me throughout my association with them. I particularly would like to thank Dr. M. A. Abidi for suggesting a feasible topic for my thesis. I acknowledge his support during the course of this project. I also gratefully acknowledge the guidance of Dr. Mohan Trivedi during the early part of my work. I extend my thanks to Mr. Randy Mayhew, the Systems Manager of the Engineering Computing Facility, for his assistance with the computing resources.

I sincerely thank Suresh Marapane for his tremendous help in the preparation of this thesis. His comments and criticisms helped me produce a comprehensive thesis. I am appreciative of Rick Eason whose ideas helped me implement the path planning algorithm on the robot. My thanks to Maureen Delcroix who proofread this thesis. I would like to thank all the people in Room 212, Ferris Hall, who helped provide a pleasant atmosphere to work in.

Finally, I would like to gratefully acknowledge the support of the Department of Energy who sponsored this research under the contract DOE-DE-FG02.86NE37968.

ABSTRACT

A 2-D and 3-D path planning algorithm for robot navigation and manipulation is developed in this thesis. This method is of particular interest for robotic systems which require both 2-D navigation to reach the work site and 3-D path planning for on-site manipulation and inspection. This study extends the approach of 2-D path planning to 3-D with minimal effort. The quadtree and octree structures are used for 2-D and 3-D workspace representation, respectively. The quadtree and octree is generated from brightness image of the workspace obtained using a camera mounted on a robot. Path generation is based on the A^* algorithm. The spatial length and the clearance space available for the robot are used as constraints in the path generation. The distance transform is used as a measure to compute the clearance space along the path. A methodology is developed to compute the distance transform of the free nodes in the 3-D workspace that is represented by an octree. The results of this approach are illustrated through an experimental system setup on a robotic workstation.

TABLE OF CONTENTS

CHAPTER PAGE

LIST OF FIGURES

FIGURE PAGE

FIGURE PAGE

CHAPTER 1

INTRODUCTION

A fundamental goal of research in robotics is to provide a robot with the capabilities to transform sensor perception into accurate, intelligent, and safe actions. To this end, the current research efforts have been directed toward providing a robot with the intelligence to make it autonomous in its operation. The ability to plan a safe and collision-free path in a cluttered workspace is an important step toward autonomous operation. Path planning is used for the navigation of a mobile robot in two-dimensions (2-D) and manipulation of a robot arm in three-dimensions (3-D).

Most of the existing approaches to path planning are tailored to solve this problem for either manipulators in 3-D or mobile robots in 2-D. The distinction stems from the fact that the operational constraints of a manipulator are different from those of a mobile robot [1]. Examples of these differences can be summarized as follows:

1. While it is sufficient to plan 2-D paths for a mobile robot, it is necessary to plan 3-D paths for a manipulator.

2. For most applications, a manipulator operates in a static, structured environment and so has an accurate knowledge of its workspace. A mobile

robot, on the other hand, needs to dynamically construct a model of the workspace with the aid of sensors mounted on the robot.

3. A mobile robot usually negotiates a given path only once, and the paths are generated in real-time, while a manipulator executes the same path a number of times. In most cases, the manipulator paths are generated off-line, hence the time involved in generating these paths is not of great significance. Therefore, a mobile robot needs to generate a *negotiable* path quickly rather than a strictly *optimal* path which is computationally prohibitive.

4. A mobile robot needs to keep as far away from obstacles as possible to avoid severe occlusions because its proximity to obstacles results in a reduction in its field of view. A manipulator's reason for keeping away from obstacles is to avoid potential collisions.

However, there are applications in which a robot may need to plan both 2-D and 3-D paths to perform certain tasks. For example, an autonomous robot operating in a nuclear power plant may be required to perform tasks such as opening a valve and picking up a tool from a tool rack. Such tasks require the robot to navigate the terrain to reach the work site and to manipulate the instruments and tools.

A path planning system was developed in this study to plan robot paths for navigation and manipulation. A major effort was made to develop an algorithm

which extends the approach of 2-D path planning to 3-D so that generating 2-D paths is a simple case of the more general 3-D path planning.

The outline of this thesis is as follows: Chapter 2 is a review of the existing approaches to robot path planning. This chapter also points out the inadequacies of these methods to generate both 2-D and 3-D paths and characterizes some of the features desired of such a system. Chapter 3 is a detailed description of the algorithm developed in this work. It also gives an algorithmic outline of the path planning algorithm. Chapter 4 outlines a method developed in this study to determine the proximity of the free space to obstacles in 3-D workspace. Chapter 5 discusses the results of this algorithm. It also describes an experimental system set up on a robotic workstation to verify this approach. Chapter 6 outlines the conclusions of this study and a few possible extensions and modifications to this approach.

CHAPTER 2

RELATED WORK

All existing approaches to path planning, except the potential field approach [2], model the search space of the robot as a graph of possible paths. This graph is then used to search for a collision-free path using one of the standard graph search techniques. The three issues to be considered in any approach to path planning are (1) the method used to represent the search space, (2) the search techniques used to generate the "best" path, and (3) the method used to verify that the generated path is indeed collision-free.

The sensory input used in the generation of workspace representation for path planning is either visual [3,4] or range data [5,6]. An image representing the range data contains explicitly information about the geometry of a scene, while a brightness image is a measure of the reflectance of the objects in the scene. Path planning involves generating a path which does not collide with an obstacle. Thus, it is important to have a sensory input which explicitly represents the physical location of obstacles in the workspace. This explicit representation of the physical location of obstacles is more easily derived from range data than brightness data. Therefore, range data is preferred to visual data for path planning.

The traditional data structure used to represent an image, namely the array

representation, is usually not suitable for path planning. The first step in path planning, therefore, is to convert the array representation of the image into a structure suitable for this task. Conventional path planning algorithms can be broadly classified in two categories based on the transformations applied to the image of the workspace. The first category includes methods that make trivial changes to the image before planning a path. The regular grid search methods [1,7] and the vertex-graph methods [8,9,10] fall in this category. The methods in the second category make elaborate changes to the image of the workspace in order to transform it into a structure which lends itself to path planning. Free-space methods [11,6,12,13] fall in this category. The rest of this chapter discusses the relative merits and disadvantages of these techniques as they relate to solving the class of problems addressed in this study.

2.1 Regular-Grid Search Methods

These approaches represent explicitly the conflicts between short paths and obstacle avoidance. A typical regular-grid search method consists of overlaying a grid of points on the image map of the workspace. The points on the grid which constitute the nodes in the graph are connected with either their 4 or 8 neighbors to form a graph. Each node in the graph can be either an *obstacle* node or a *free node*, corresponding to the point on the grid being part of an obstacle or a free space, respectively. Given the start and goal positions of the robot in the grid, the graph is then searched for the "shortest" path using the

standard A^* algorithm [14]. The resulting path usually clips obstacle corners. One way to alleviate this problem is to introduce a margin of safety around the obstacles. The introduction of a margin of safety results in the elimination of paths which otherwise would have been feasible. Since a regular grid is used to model the workspace of the robot, the generated path is "jagged" and contains steep turns. Thorpe [1] overcomes this jagged path using a technique called *Path Relaxation.* Path relaxation is a two-step process in which the path generated by a regular-grid method is iteratively refined by allowing nodes on the path to move off the grid. This process adjusts or "relaxes" the position of each node to decrease the total length of the path.

Path planning using this method has been implemented only in two dimensions. Extending this approach to 3-D is a difficult task. This extension requires a 3-D binary array representation of the 3-D workspace on which the grids can be superimposed. The generation of the 3-D binary array representation is computationally intensive. This requirement is a general shortcoming of this approach for 3-D path planning.

2.2 Vertex Graph Methods

These methods approximate the obstacles by maximal bounding polyhedra [8,9,10,6]. A graph is formed by considering every pair of vertices of the polyhedra. The lines joining each of these pairs of vertices form segments of the graph if it does not intersect the obstacles. This graph is then searched for the short-

est path using the standard search techniques. *Pseudo-obstacles* are generated by expanding the obstacles by the size of the robot and shrinking the robot to a point [10]. These pseudo-obstacles are used to form the bounding polyhedra which, in turn, are used to generate the graph on which the search is performed. The concept of pseudo-obstacles is used to ensure that the robot does not collide with the obstacles along the generated paths.

A variation of the vertex graph approach as used by Moravec [10] consists of recognizing obstacles as a cloud of features which are then expanded to fit circles. This cloud of features is extracted using an "interest operator" from a set of 9 stereo images obtained from a sliding camera mounted on a robot. A new set of images is taken at every stop of the robot. These images are used to update the model of the workspace so that it reflects the current state of the environment. To avoid potential collisions of the robot, the obstacles in the workspace, represented as circles, are expanded the diameter of the robot and the robot is shrunk to a point. The "best" path for the robot from its current location to the goal is chosen as the shortest sequence of line segments that are tangent to the expanded circle. The paths thus generated have a tendency to go near the obstacles and may result in collisions due to positional errors in the control of robot path execution. Moreover, this approach is computationally intensive and does not provide a means for trading a longer path for more clearance; hence, it is not suitable for the class of problems addressed in this study.

2.3 Free-Space Methods

The class of algorithms which represent explicitly the free-space of the robot rather than the obstacles is classified as free-space methods. These methods force the path to run down the middle of the corridor between the obstacles. Brooks [11] proposed a solution to the "find path" problem using this approach. This solution, developed in a 2-D plane, involves fitting "generalized cylinders" into the space between the obstacles. This representation creates pathways in which the robot may travel without colliding with obstacles. However, it is not always possible to fit cylinders to exactly occupy the free space of the robot. Consequently, much of the free space is not utilized in path planning. In another variation of this approach, the Voronoi diagram [15] of the free-space is computed. The spine of the Voronoi diagram or the axes of the generalized cylinders constitutes a graph of potential paths which is then used to search for the shortest path. This approach to path planning, in general, is computationally intensive because of the mapping required between the workspace and the search space. Therefore, this approach is not attractive as a path planning method for real-time applications.

The other free-space approaches include the use of medial-axis-transforms [13] and maximum area convex polygons [12,5]. Though the latter method was developed to navigate a mobile robot in a dynamic scene, extending it to 3-D would not be a simple task.

2.4 Potential Field Approach

This approach, formalized by Khatib [16], conceptually turns the robot into a marble, tilts the floor towards the goal and determines the trajectory of the robot. Obstacles are represented by hills with slopes, the gradient of which depends upon how close the robot can get to that location without colliding with it. The "hill climbing" [17] search technique is used to find the best path. The resultant path will keep as far away from obstacles as possible while seeking passes between adjacent hills. There are sophisticated extensions to this approach which provide the robot with a momentum while taking into consideration the energy required to accelerate, decelerate, and turn[2]. The search technique used in this approach, namely, the hill-climbing, is not robust. Paths are susceptible to pitfalls at a local minimum in the potential field.

This treatment of some of the existing approaches to robot path planning can be summarized as follows: A system designed to perform 2-D and 3-D path-planning in real-time or near real-time should be based on a compromise between the approaches outlined above. The trade-offs required in this process may be to sacrifice a strictly *optimal* path, a computationally intensive task, for a *negotiable* path. The issues to be considered in the design of such a system are given below.

- The choice of the data structure used for workspace representation plays a crucial part in the robustness and performance of the system. The data structures used have to be robust to handle both static and dynamic scenes. Incremental changes to the data structures have to be very efficient.

- A path planning algorithm designed to operate in real-time should be rapid relative to the motion of the robot and its controllers. It is desirable to have an algorithm which will lend itself to parallelism.

- In generating a collision-free path, factors such as spatial length of the path, proximity of the path to obstacles and distance of the path in uncharted areas should be treated as constraints on the path. The "best" path is obtained by optimizing these constraints so that the resultant path has minimal cost.

CHAPTER 3

A PATH PLANNING ALGORITHM

A path planning algorithm was developed in this study while taking into consideration three major issues. They are (1) the speed of execution, (2) the robustness of the data structure used for workspace and search space representation, and (3) the adaptability of the algorithm to changes in the dimension (2-D or 3-D) in which the path is generated. The system is capable of generating safe 2-D and 3-D robot paths for navigation and manipulation. The implementation of this algorithm assumes a static environment. This approach can be extended to generate paths in a dynamic workspace where the position and orientation of the objects in the scene may change over time. These changes can be incorporated by applying incremental transformations to the workspace representation. The ability to efficiently update the data structure representing the workspace in order to reflect these changes is an important consideration in the design of a path planning system for dynamic scenes. Algorithms have already been developed which enable such incremental changes to be made efficiently on the data structure chosen in this study for workspace representation [18,19].

The two major functional components of this algorithm are (1) Workspace Representation, and (2) Path Generation. The first module provides a spatial representation of the workspace in a form suitable for path planning. The rep-

resentation is derived from images which are either obtained from the sensors mounted on the robot or provided by the sensory recognition system. A camera (visual sensor) mounted on the robot arm was used in this system to acquire the image of a workspace. The Path Generator uses this representation to generate optimal paths using the well-known A^* algorithm [14].

It should be pointed out that path planning is performed with respect to a reference point which does not collide with the obstacles along the generated path. Without loss of generality, a robot can be represented by a reference point obtained by shrinking the volume occupied by the robot. Correspondingly, the obstacles in the workspace are enlarged to accomodate the finite dimensions of the robot or the moving object. The implication of this assumption is that a path generated for the reference point which does not intersect an obstacle is a collision-free path for the entire robot. The enlarged obstacles in the workspace are called *pseudo-obstacles.*

The generation of the pseudo-obstacles in 2-D path planning consists of enlarging the obstacles in the image which represents the 2-D workspace. The resulting image consisting of the pseudo-obstacles is used in the generation of the data structure representing the workspace. The pseudo-obstacles in the 3-D workspace are generated by finding pseudo-obstacles in the image representation of the three orthogonal 2-D projections. Given an image representing a 2-D workspace or an orthogonal projection of a 3-D workspace, the pseudo-obstacles of the obstacles within this image are generated by a simple region growing technique outlined below. The method consists of first determining α, the factor

(in number of pixels) by which the obstacles are to be expanded. This factor is determined by the size of the robot and the distance of the sensor (camera) from a reference point in the workspace. This reference point is equi-distant from the center of the camera lens in all the three viewing directions in case of 3-D path planning. The image array representing the 2-D workspace or a projection of the 3-D workspace is scanned, and a square box of size 2α of the same intensity as the obstacle is drawn around every obstacle pixel. This method presupposes that the edges of the image do not interfere with the expansion process. Since this assumption may be unrealistic, the portion of the image within a distance α from the edge of the image is retained without any expansion. The complexity of this method is of $O(N)$ where N is the number of pixels in the image. This method is simple, fast and easy to implement.

The following sections detail the two functional components of this algorithm: Workspace Representation and Path Generation.

3.1 Workspace Representation

This module provides the path planner with a description of the workspace using the images obtained from sensors. Previous work in describing and identifying 3-D objects in a scene using sensors can be classified as (1) model-based approach and (2) non-model-based approach [20]. In a model-based approach, a model of the 3-D objects is stored in the computer and used as a reference in recognizing or identifying objects in the environment [21]. This assumes a

certain amount of *apriori* knowledge of the workspace. In a non-model-based approach, attempts are made to reconstruct the 3-D objects in a workspace based on information such as the object's shade, surface orientation, and range information obtained from sensory data [22]. This method assumes no knowledge of the workspace. Path planning can be performed based on the reconstructed model of the workspace obtained from either of these approaches. However, the recognition and the reconstruction processes inherent in the above approaches are computationally intensive.

This approach to path planning does not seek to recognize or reconstruct a 3-D model of the obstacles in the workspace, but rather is concerned with the presence or absence of obstacles at every point in the workspace. The choice of octree (quadtree in 2-D) as a data structure to represent the 3-D workspace of the robot is a direct consequence of the above fact.

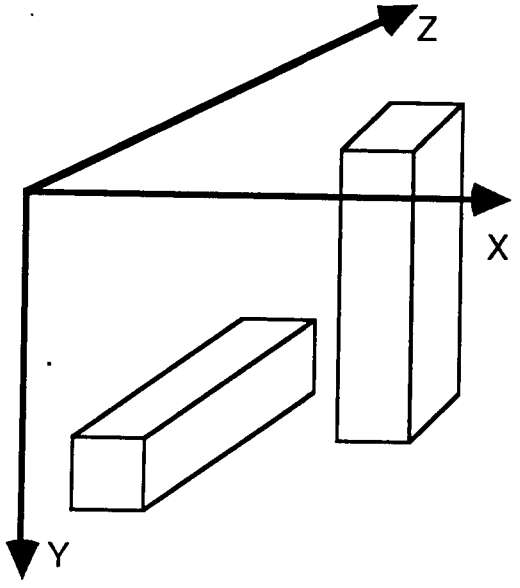
3.1.1 Data Structure

This section describes the data structure used for workspace representation, namely, octrees and quadtrees, and provides a justification for such a choice.

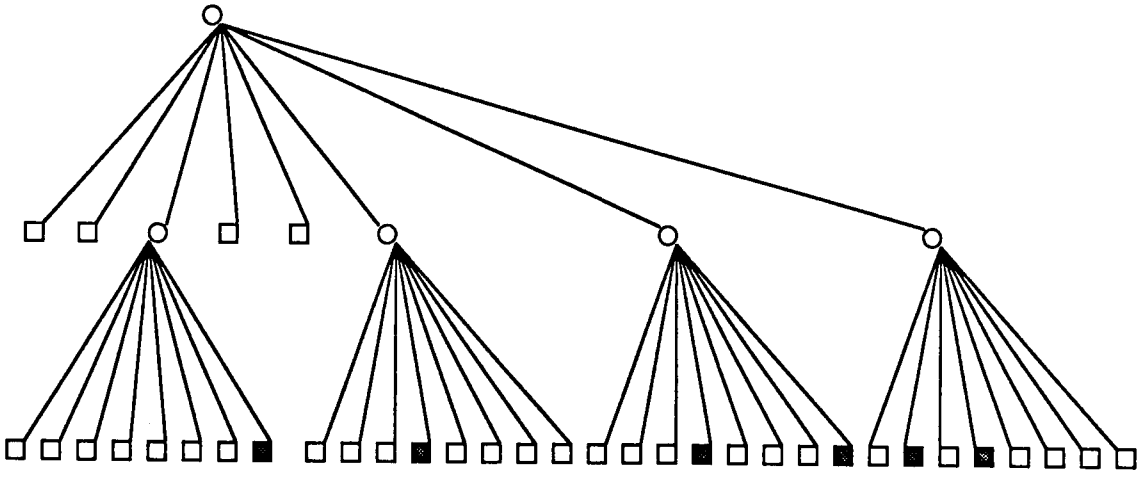
The use of an octree data structure for the representation of 3-D objects has been extensively investigated [23,24]. An octree can be generated [23,24] by a recursive decomposition of a 3-D space, generally a cube, into octants until all voxels (volume elements) in each octant lie entirely inside or outside an object. Thus, each node in the octree represents a region of the 3-D space. The octree is built by first considering the entire 3-D space (3-D image array) as a single

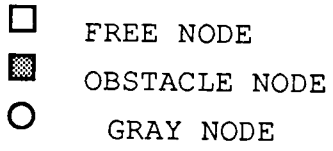
node. This node is called the *root node*. If the node is not homogeneous, *i.e.*, the space represented by the node is not completely occupied by the free-space or the object, it is divided into 8 octants. Each octant is further subdivided if it is not homogeneous. The process of subdivision continues until all the octants are homogeneous. A node in the tree is either a terminal node or a non-terminal node. A non-terminal node has 8 children. A terminal node is also called a *leaf node*, and it uniquely represents a volume in the workspace of the robot. A leaf node is called a *free node* if it represents a free-space and an *obstacle node* if it is occupied by obstacles. Non-terminal nodes are called *gray nodes*, and they represent a mixture of free-space and obstacles. In normal octrees, the leaf nodes are either free nodes or obstacle nodes. However, in pruned octrees, the tree is pruned at a predetermined level by making certain gray nodes become terminal nodes. Pruning can be based on the grayness of the node or depth of the tree. The basic idea of using this structure is to provide a representation at several levels of resolution of the image. Quadtrees are the 2-D equivalents of octrees in that they are used to represent 2-D images. They follow the same principle as octrees except that 2-D image blocks are divided into 4 sub-blocks instead of 8. Figures 3.1 and 3.2 give 2 simple examples of the octree and quadtree representation of a 3-D and 2-D workspace, respectively.

The choice of octrees as a data structure for representation has a number of advantages in this application. Since quadtrees share the properties of octrees,



(a)





(b)

Figure 3.1: An example showing (a) a 3-D workspace and (b) its octree representation.

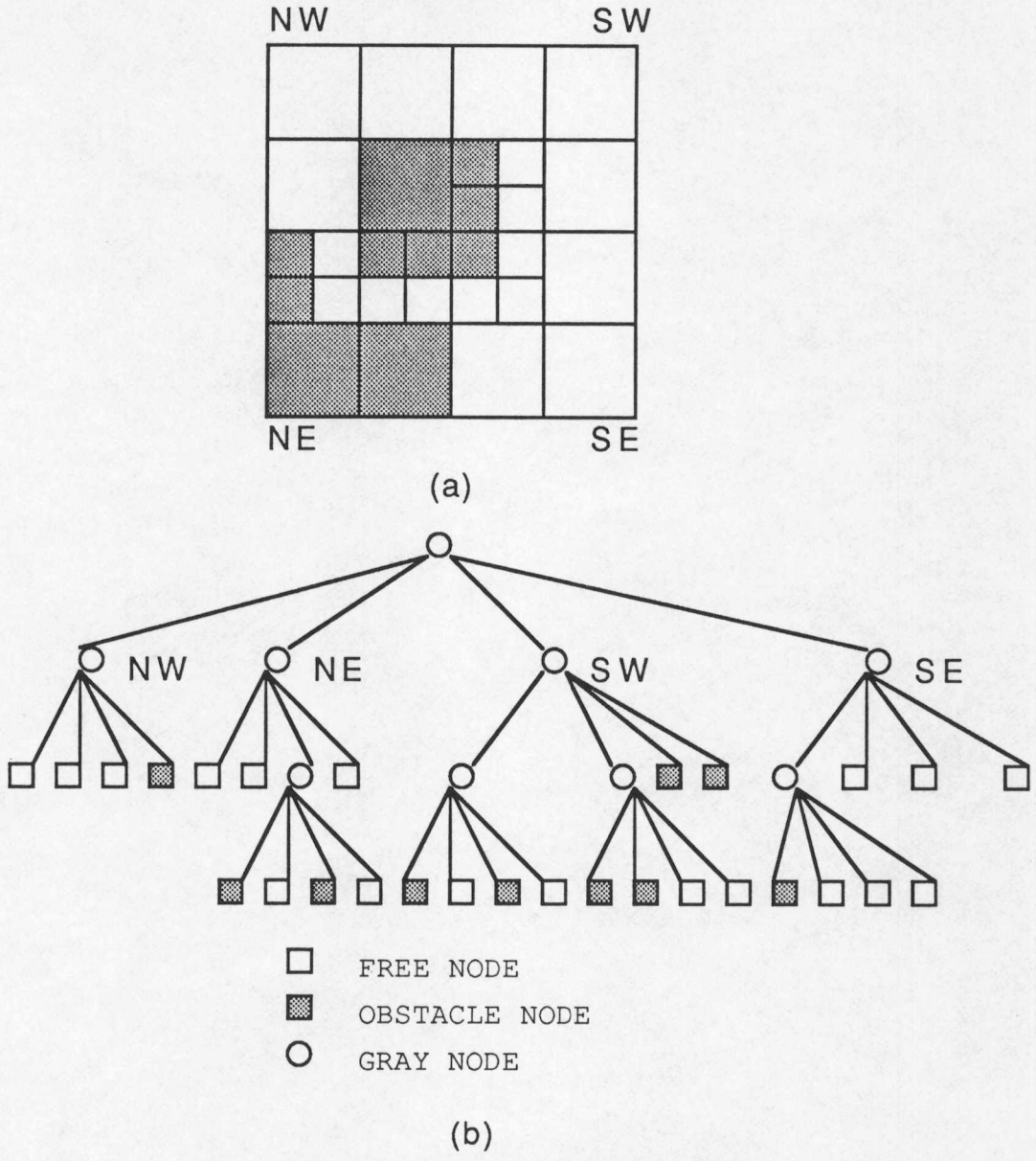


Figure 3.2: An example showing (a) a binary image of a 2-D workspace and (b) its quadtree representation.

this discussion holds true for the quadtrees as well. The features of the octrees considered suitable for path planning are illustrated below.

1. Octree representation of the 3-D space is more memory efficient than an array representation.

2. An octree is a pre-sorted data structure and hence provides a spatially-indexed representation of the obstacles in the workspace.

3. Searching a continuous search space potentially leads to a large number of paths to be considered. Octrees provide a means for decomposing the free-space into cubes which can be treated as symbolic units. Each unit can be considered as a node in the search graph. The successors of a node in the graph are the adjacent free neighbors along the 6 faces, namely, North, East, South, West, Front and Back, of the octant representing the node. Thus, octrees provide a means for reducing the number of directions along which the successors of a node are sought during the search.

4. One of the most important considerations in spatial representation for path planning is to avoid excess detail on parts of space that do not affect the planning operation [25]. Hierarchical decompositions like octrees are a good way to achieve this, since the representation cost involved is small. Moreover, they facilitate path planning at multiple resolutions of the image [3], thereby producing an overall savings in computation.

5. Efficient algorithms do exist for incrementally modifying octrees [18,19].

These techniques are necessary to efficiently update the changes in the spatial representation of a dynamic scene.

6. Finally, octrees are often useful for tasks other than path planning. They can serve as useful representation for sensory interpretation algorithms and solve the hidden feature problem in vision verification and computer graphics [26,27]. Thus, in a complete robot planning, control, and sensory system, octree representation can serve as a versatile data structure.

3.1.2 Generation of 3-D Representations

One of the problems with the octree structure is its generation. The octree generation techniques used in [23,24] are based on an underlying assumption that a 3-D (binary) array description of the workspace is available. However, the acquisition of such a 3-D array is a difficult task. Since the goal of this study is to perform path planning in both 2-D and 3-D, a logical approach would be to extend the techniques used for 2-D path planning to 3-D. Hence, an octree generation algorithm based on reconstructing the 3-D representation from a quadtree representation of the three orthogonal views was adopted [28,26]. This method involves generating the *swept volumes*, which are obtained by sweeping the three 2-D projections along their respective viewing directions. The swept volumes thus generated are then intersected to yield a volumetric representation of the 3-D object. The resulting volume generated is called the *maximal volume*. Figure 3.3 illustrates the volume intersection technique. It can be observed that the

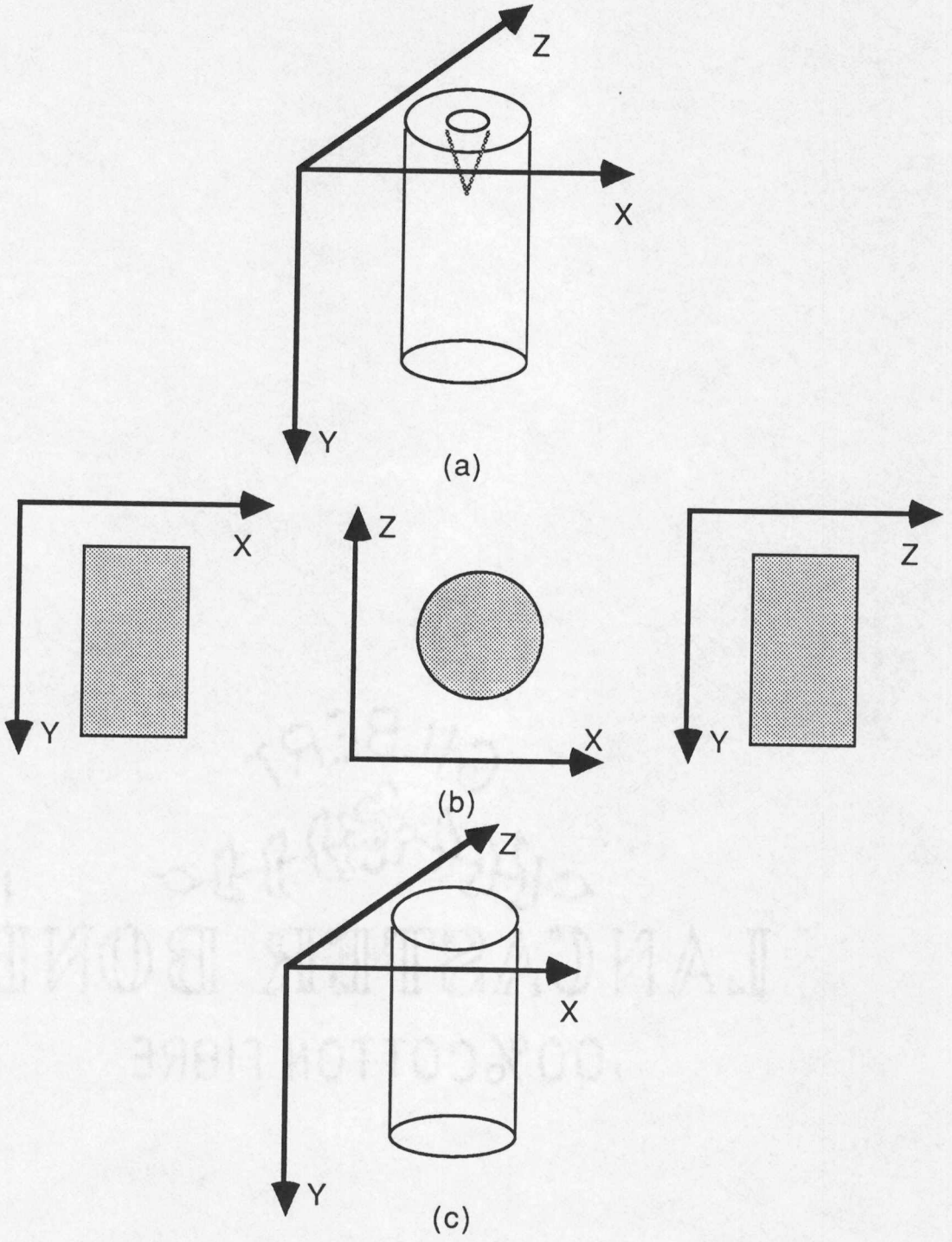


Figure 3.3: An example showing (a) a 3-D object (b) its 3 orthogonal projections and (c) the maximal volume reconstructed by volume intersection.

maximal volume completely encloses the object. A 3-D point which is a part of a 3-D object in the workspace will always be within the maximal volume generated by the volume intersection technique. The converse is usually not true. This limitation is acceptable since the task of path planning does not require the recognition of an object but rather involves avoiding a collision with it.

The steps involved in the generation of an octree from quadtree representation of the three orthogonal 2-D projections are given below.

Step 1:

Generate the quadtrees of the three orthogonal (front, top, and side) views of the scene.

Step 2:

Generate the *swept volumes* of the three projections from their corresponding quadtree representations. While an octree structure is used to represent the *maximal volume*, the *swept volume* is stored in an intermediate structure called *pseudo-octree*. Since a pseudo-octree represents a 3-D volume, each node in this structure has eight off-springs (identical to an octree). The realization that the two volumes obtained by bisecting a *swept volume* by a plane perpendicular to the viewing direction are identical is crucial in the generation of the pseudo-octree. The implication of this observation is that each node in the pseudo-octree will have two identical off-springs. Thus, to generate a pseudo-octree, it is enough to determine only the 4 non-identical off-springs of each node. These 4 off-springs are the four children of the corresponding node in the quadtree from which the pseudo-octree is created. Therefore, each quadtree node corresponds to two

nodes in the pseudo-octree. This correspondence is established through a look-up table. However, it should be noted that three such tables are required, one for each viewing direction. Since the pseudo-octree structures are not explicitly created and stored, the processing time and storage requirements are reduced.

Step 3:

Generate the octree representing the *maximal volume* through the intersection of the three pseudo-octrees generated in the previous step. The intersection procedure has been outlined in [26].

The algorithm is efficient in terms of storage requirement and processing time because of the compactness of the octree structure and efficiency of the tree traversal procedure. The generation of a quadtree from a 2-D image is straightforward. The image is converted into a binary image by simple thresholding. The quadtree representation of this binary image is obtained using the algorithm given in [29].

3.1.3 Implementation

A linked list data structure is used to represent the quadtree and octree. Each element of the linked list represents exactly a node in the tree structure. Each node contains pointers to its children and a pointer to its own parent node. The children of a node correspond to the sub-octants or sub-quadrants within an octree or a quadtree node, respectively.

The other information contained in a tree node is its size (in number of pixel units), the coordinates of the center of the region represented by the node, and

a label describing the contents of the node. A node can have one of three labels, namely, WHITE, BLACK, and GRAY. They correspond respectively to a region represented by the node as a part of a free space, an obstacle, or a combination of the two. In addition, WHITE (free) nodes contain the distance from the center of the node to the boundary of the nearest BLACK (obstacle) node. This distance is called the *distance transform* [30]. The usefulness of this parameter will become apparent during the discussion of path generation. As might be expected, the distance transform is undefined for BLACK and GRAY nodes and will be taken as zero. These parameters of a node in the octree or quadtree, namely, its size, its label (or color), and the distance transform, will be referred to as SIZE(X), COLOR(X), and DIST(X), respectively, where X is the node under consideration.

Another parameter that will be needed later is the *neighbor* of a node in the octree or quadtree. For the purpose of this algorithm, a node N will be considered a neighbor of node X if the following two conditions are satisfied [30].

1. The nodes X and its neighbor N touch each other at least at one point.

2.

$$SIZE(N) \geq SIZE(X) \quad if \quad COLOR(X) \in \{BLACK, WHITE\}$$
$$SIZE(N) = SIZE(X) \quad if \quad COLOR(X) \in \{GRAY\}$$

This definition was adopted based on the fact that it results in evaluation of the distance transform with a knowledge of the fewest possible neighbors. From

the above definition of a neighbor, it follows that an octree node can have a maximum of 26 neighbors: 6 of them adjacent to 6 faces, 12 of them sharing one of the 12 edges, and 8 of them touching one of the 8 corners of the octant represented by the octree node. Figure 3.4 illustrates the 26 possible directions of an octree node along which a neighbor can be sought.

The naming convention adopted to identify the 26 directions for an octree node are as follows: The 6 faces of the octant will be labeled N, S, E, W, F, B, corresponding to the first letter of the words North, South, East, West, Front, and Back, respectively. The 12 edges of the octant are identified by a three-letter name in which the first-two letters correspond to the direction of the 2 faces which intersect to form the edge, while the third letter E is used to indicate that the direction is an Edge of an octant. The corners are also identified by a three-letter name which corresponds to the direction of the 3 faces which intersect to form a corner [Fig. 3.4].

3.2 Path Generation

Given the 2-D or 3-D representation of the workspace in the form of quadtree or octree structure, this module returns an optimal collision-free path from the current location of the robot to its destination. The path generated by this module is a set of points in the workspace from which a path is reconstructed by joining consecutive points in straight lines. This module has the flexibility of

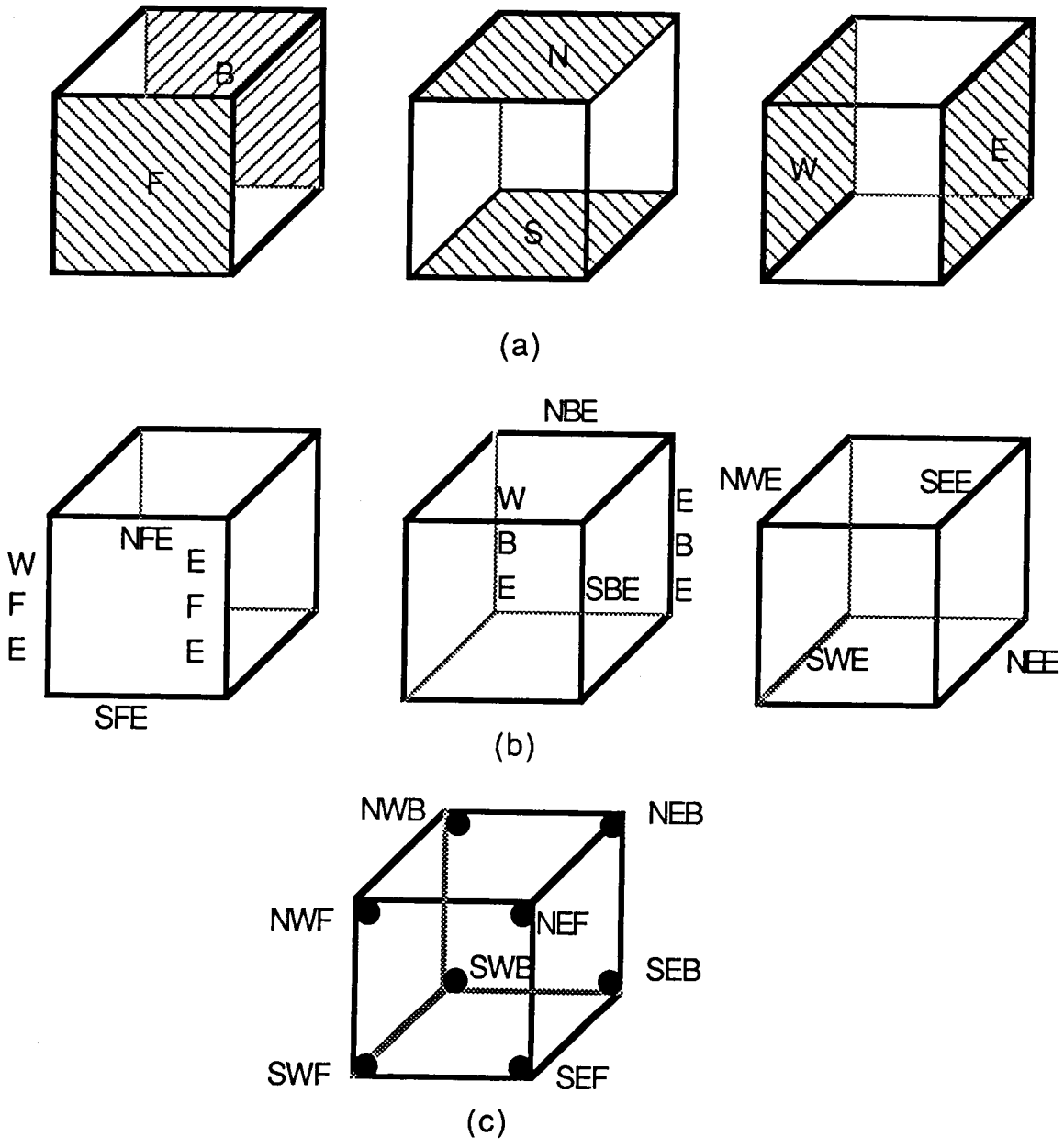


Figure 3.4: An illustration of the 26 possible directions along which a neighbor can be sought for an octree node. (a) 6 Faces, (b) 12 Edges, and (c) 8 Corners of the octant representing the octree node.

generating either 2-D or 3-D paths given the description of the workspace in appropriate dimension.

The task of path generation begins by obtaining the start and the goal coordinates of the path to be generated. The leaf nodes S and G, representing the regions containing the start and the goal points, respectively, are then identified. A path consisting of a set of free nodes in the octree (quadtree in 2-D) which constitute a path between S and G is found using the A^* search algorithm [31]. A straight line path is reconstructed by joining the centers of these nodes in straight lines. Figure 3.5(a) illustrates a simple 3-D path generated using this algorithm between two points S and G on a 3-D workspace (same as the workspace shown in Fig. 3.1). To see that the 3-D path is indeed collision-free, the orthogonal projections of the path and the objects in the workspace are given in Fig. 3.5(b) through Fig. 3.5(d). A 3-D path is collision-free if every point along the path is collision-free in at least one of the three orthogonal projections of the scene. Thus it can be seen from Fig. 3.5 that this path is indeed collision-free. Figure 3.6 contains a binary image and its quadtree representation, which illustrates a 2-D path generated by this approach between the points S and G.

Two issues need to be addressed in path generation. They are (1) the determination of *successors* of a node which are explored during the search and (2) the evaluation of the cost associated with the node. The determination of successors consists of finding the nodes horizontally or vertically adjacent to the node which is being expanded. Diagonal neighbors are ignored. The choice of horizontal or

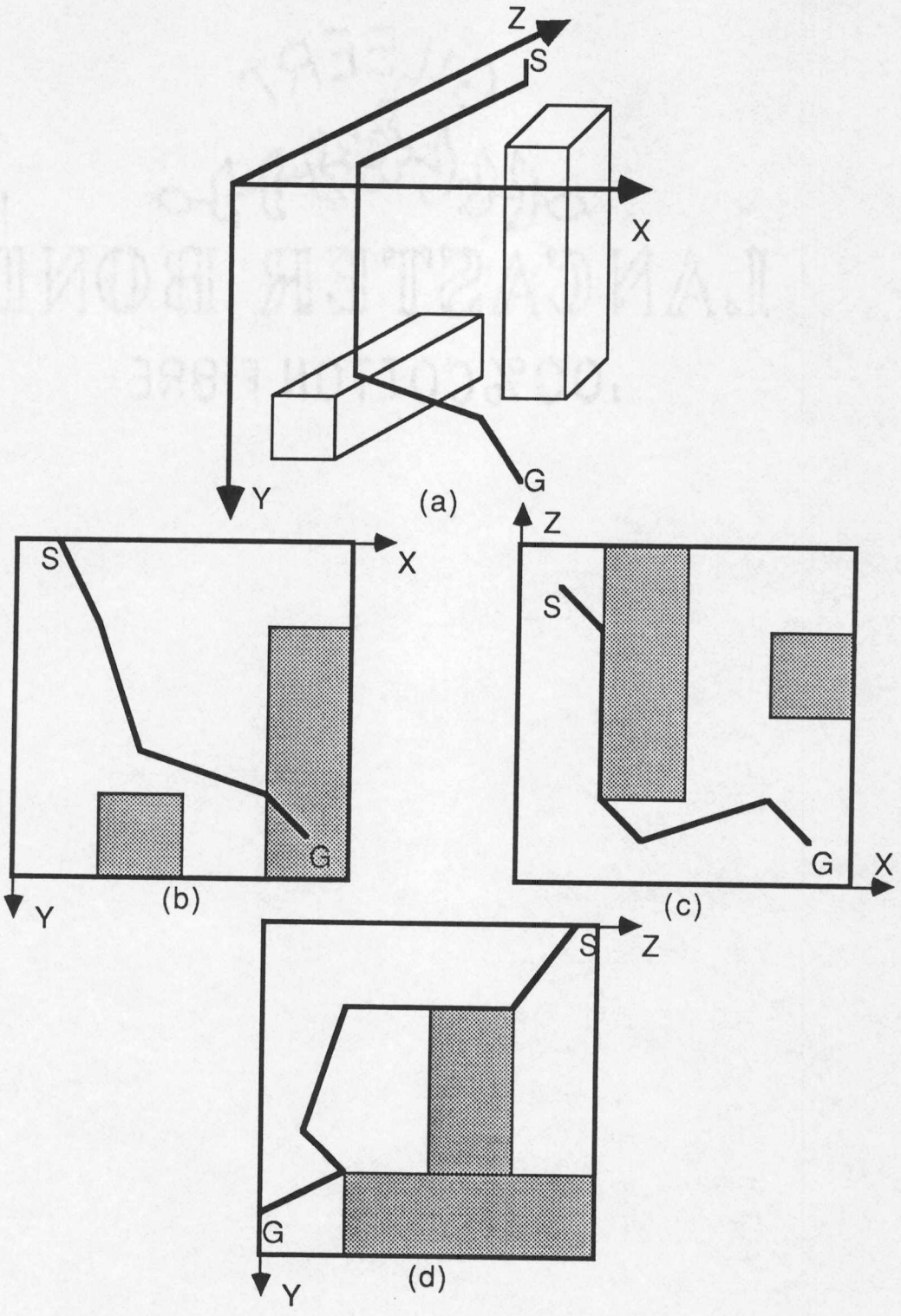


Figure 3.5: An illustration of (a) a simple 3-D path generated by the algorithm and (b)-(d), its three orthogonal projections (front, top, and side views, respectively).

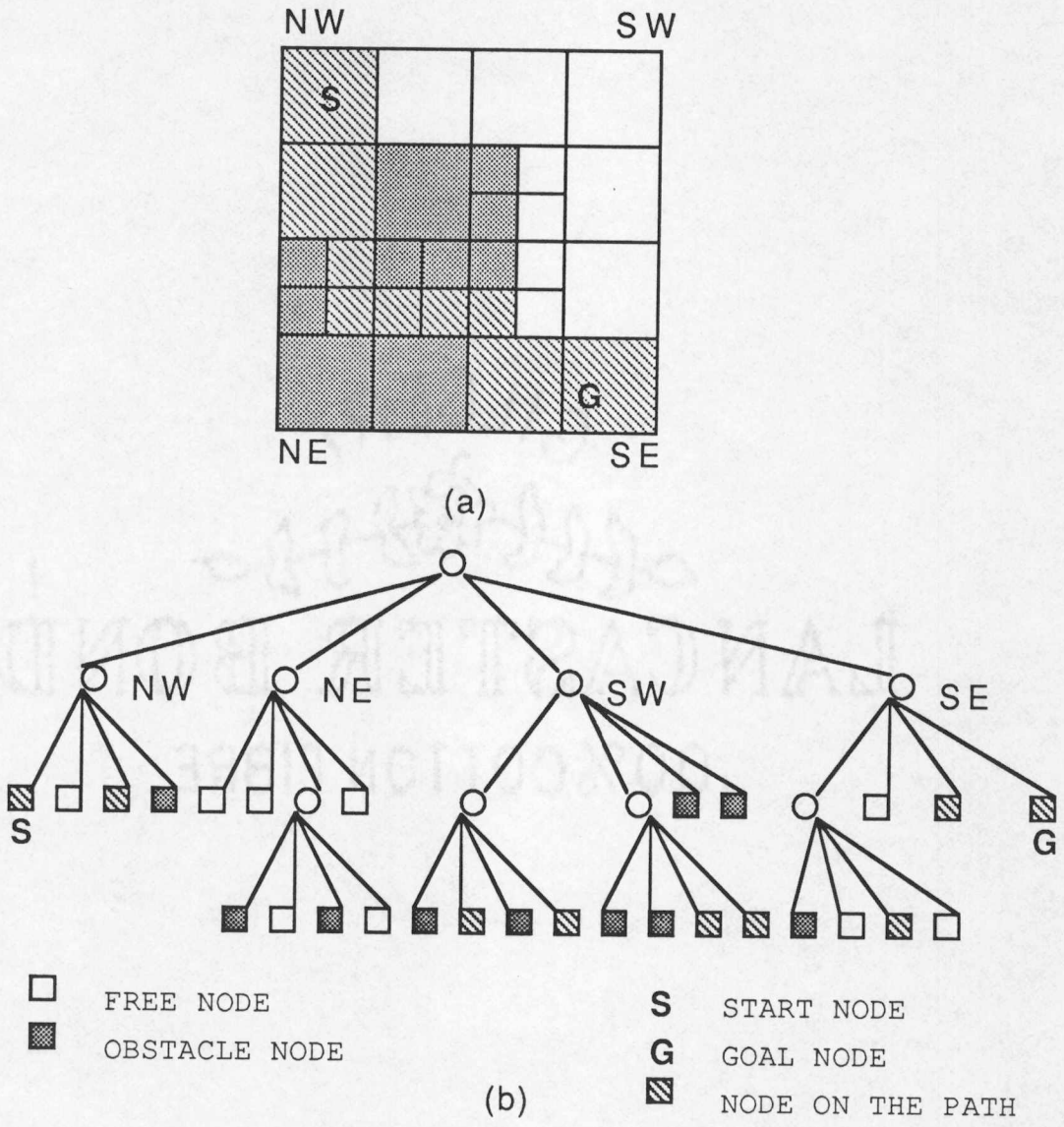


Figure 3.6: An illustration of (a) a simple 2-D path generated by the algorithm (path shown shaded with slanted lines) and (b) the quadtree representation of the 2-D workspace.

vertical neighbors serves to avoid paths which may otherwise clip obstacle corners [3]. The number of directions along which successors will be sought are 6, corresponding to 6 faces of an octree node. On the other hand, the number of such directions for a quadtree node is four. This is the only difference between 2-D and 3-D path generation. The second issue in path generation, namely, the evaluation of cost of a node, will be discussed in the next section.

3.2.1 Cost of a Path

Every successor of the node expanded during the search is assigned a cost function $f(c)$ where c is the successor node being evaluated. The generated path is optimized with respect to this function. The function of $f(c)$ is given by

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

where $g(c)$ is the cost incurred in traversing the path from the start node S via its current parent in the search. The cost component $h(c)$ is a heuristic estimate of the remaining path from c to the goal node G. The Euclidean distance measure is used as the heuristic estimate. Since this estimate is a lower bound on the actual cost of the path, this measure as a heuristic is *admissible* [32].

The cost component $g(c)$ is defined as a function of those factors which actually affect the nature of the path. The two factors considered in this study to influence the cost of the path are:

1. the spatial length of the path, and

2. the proximity of the path to obstacles.

The resulting path that is generated is a compromise between these two factors. The first component of $g(c)$ is computed by determining the spatial distance traversed along the path to arrive at the node c. Since the generated path consists of a sequence of nodes, the distance between two nodes is taken as half the sum of the node sizes. The second component of $g(c)$, namely, the clearance space available for the robot along the path, also called the *clearance factor* (CF), can be computed using the distance of a successor node (all successors are free nodes) from the boundary of its nearest obstacle node. This distance is also called the *distance transform.* This distance information is used in such a way that a *successor* node, which is closer to an obstacle and hence offers little clearance space for the robot, is penalized more than those *successors* which are further away from the obstacles [3]. The resulting path due to this component will consist of nodes with the largest clearance of the obstacles.

It can easily be seen that the first component has the effect of pulling the path as close to the obstacles as possible without causing a collision in order that the spatial length of the resultant path be a minimum. On the contrary, the second component of $g(c)$ has the effect of pulling the path away from the obstacles. As a result, the eventual path will keep as far away from obstacles as possible while maintaining the shortest possible distance.

What makes this approach more attractive is its ability to weigh the latter component, namely, the *clearance factor* (CF), anywhere in the range of 0 and 1. Therefore, the search algorithm totally ignores the clearance information for those paths for which the CF was set to 0. The ability to weigh the CF to

plan paths with different clearances is particularly important if some *apriori* knowledge of the distribution of the obstacles in the workspace is known. If the workspace is cluttered with obstacles, it might be safe to keep as far away from obstacles as possible (CF is set to 1), while the same might not be true when the workspace is sparsely populated with obstacles. Another instance in which this consideration can be useful is the case for which it is not possible to find the location of the robot with enough precision, which requires that CF be set to 1. The latter condition may arise due to inaccuracy of the robot controllers, traction of the wheels, or the smoothness of the floor.

The problem that still remains to be solved is the evaluation of the *distance transform*. This problem has been investigated for a quadtree by Samet [30]. The metric distance used was the *chessboard distance*, or the *maximum value metric*, in which the distance between two points $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$ in XY Cartesian space is given by

$$\overline{P_1P_2} = max\{|x_1 - x_2|,\ |y_1 - y_2|\}.$$

While Samet's algcrithm [30] was adopted for 2-D path planning, an approach based on those lines was formulated for octrees. The evaluation of the distance transform of the free nodes in an octree is more complicated than its quadtree counterpart since it involves exploring neighbors in 26 directions of a free node, as opposed to 8 directions for quadtrees. The details of this approach are outlined in the next chapter.

3.2.2 Implementation and Data Structures

The space in which the search is performed is called the *search space*. This search space can be represented as a tree structure, also called the *search tree*. Each node in the search tree corresponds to a node in the octree or quadtree representation of the workspace. The root of the search tree corresponds to the octree node which represents the region containing the starting point of the path to be generated. The search tree is dynamically built as the nodes are expanded during the search. Thus, the children of a node in the tree are its *successors* which were generated when the node was explored.

The data structure used to represent the search space is a general tree in which each node in the tree has an arbitrary number of children. This arbitrariness in number is due to the fact that the successors of a node are determined dynamically and that their number varies from one node to another. It may be recalled that the path generation module is common to both 2-D and 3-D path planning. In other words, the 2-D and 3-D search spaces are represented by a single data structure. Thus, a node in the search tree corresponds to a 2-D or 3-D search space depending upon the dimension in which the search is performed. The implementation of this structure was facilitated by the use of "union" data types in C language. The information contained within each node of the search tree consists of the components of the cost function which are associated with that node and a pointer to the node in the octree or quadtree which it represents. In addition, a node has two other pointers: one pointing to its parent node in the

search tree and the other pointing to a linked list consisting of its successors. The path, as determined by this module, consists of a set of nodes in the *search tree* beginning at the start node (also known as the root of the tree) and terminating at the goal node.

3.3 Algorithmic Outline of the Path Planning Approach

This section gives an algorithmic outline of the path planning approach developed in this study.

1. Since the algorithm can generate both 2-D and 3-D paths, it is necessary to specify the dimension in which the path planning is to be performed.

   (a) To generate 2-D paths, the quadtree representation of the 2-D workspace has to be generated. The generation of the quadtree structure is outlined below.

   - Obtain an image of the 2-D workspace. Convert this image to a binary image by simple thresholding or any other segmentation technique.
   - Increase the size of the obstacles within this binary image by a factor corresponding to the size of the robot. The resulting image consisting of *pseudo-obstacles* is used in the generation of the quadtree representation of the 2-D workspace.

- Generate the quadtree representation of the 2-D workspace from the image obtained in the previous step.
- Compute the size and the coordinates of the terminal nodes in the quadtree.
- Compute the distance transform of the free nodes using the quadtree distance transform algorithm.

(b) To generate 3-D paths, the octree representation of the 3-D workspace has to be generated. The quadtree representations of the three orthogonal 2-D projections (front, top, and side views) of the 3-D workspace are required to build the octree. Three images representing these orthogonal views are used as the 2-D projections. The generation of the octree structure is outlined below.

- Obtain the three views from a camera or cameras positioned at three orthogonal locations with respect to a reference point in the 3-D workspace. Generate the corresponding binary images by simple thresholding or any other segmentation technique.
- Increase the size of the obstacles within these binary images by a factor corresponding to the size of the robot. These images are used in the generation of the octree representation of the 3-D workspace.
- Generate the quadtree representation of the images generated in

the previous step. They correspond to the projections of the 3-D workspace on the three orthogonal planes.

- Generate the octree representation of the 3-D workspace from the three quadtrees generated in the previous step through volume intersection.
- Compute the size and the coordinates of the terminal nodes in the octree.
- Compute the distance transform of the free nodes using the octree distance transform algorithm.

2. Determine the start and goal leaf nodes in the tree which represent the regions of the workspace containing these points.

3. Set the "clearance factor" for the path to be generated.

4. Determine the path using the A^* search algorithm. If the path does not exists, exit the program. Otherwise, display the generated path and drive the robot.

CHAPTER 4

OCTREE DISTANCE TRANSFORM

This chapter describes an algorithm developed as a part of this study to compute the distance transform of free nodes in the octree. The distance transform of a free node in the octree is a measure of the proximity of the node to an obstacle.

Definition: The distance transform of a free node in the octree is the distance from the center of the region represented by the node to the boundary of its nearest obstacle node. It is undefined for GRAY and BLACK nodes and will be taken as zero. The metric chosen to measure this distance was the *chessboard distance* (also called the *maximum value metric*). Given 2 points $P_1(x_1, y_1, z_1)$ and $P_2(x_2, y_2, z_2)$ in the XYZ Cartesian space, the distance between P_1 and P_2, according to this metric, is defined as

$$\overline{P_1P_2} = max\{|x_1 - x_2|,\ |y_1 - y_2|,\ |z_1 - z_2|\}.$$

Since the image is represented by an octree, the use of this metric over other metrices (Euclidean distance and Absolute value metric) results in considerable simplicity in computation [30].

The question that remains to be answered is the procedure for determining the distance transform. This can be subdivided into two sub-tasks.

1. The neighbors of an octree node are first identified.

2. Given the neighbors of a WHITE (free) node, the distance transform is then computed. This consists of processing one neighbor at a time to compute the distance of the octree node under consideration to the nearest boundary of the neighbor. After all the neighbors are processed, the minimum over all such distances is taken as the distance transform. Processing a neighbor is dependent on the color of the neighbor and is described below.

   (a) Since the distance transform of a free node is the distance to its nearest obstacle block, all WHITE neighbors can be ignored.

   (b) For a BLACK neighbor, the distance from the center of the free node under consideration to the nearest boundary of the region representing this neighbor is found. This distance is half the size of the free node itself.

   (c) Processing a GRAY neighbor is more complicated than processing others neighbors since it involves exploring the leaf nodes within this GRAY octant. For each GRAY neighbor, the distance from the center of the octree node under consideration to the closest boundary of the nearest obstacle block contained in this GRAY octant is found.

Since a BLACK neighbor is guaranteed to return such a distance, further computations can be ceased as soon as a BLACK neighbor is found. This can lead to a considerable saving in computation.

It is worth mentioning that for any WHITE (free) node in the octree, all its neighbors cannot be WHITE. This an extension of a theorem stated in [30] for quadtrees. The implication of this theorem is that for any free node in the octree the *distance transform* exists.

Notation Used:

It may be recalled every node in the octree, except the root, has a parent associated with it. A node shares three faces in common with the octant representing its parent (Fig.4.1). This feature is used to identify a node by a three-letter name which is also called the *octant type* of the node. The three letters in the name correspond to each of the three faces. The convention used to identify the faces of an octant representing an octree node was given in the previous chapter. From here on, an octant will be frequently identified by its octant type.

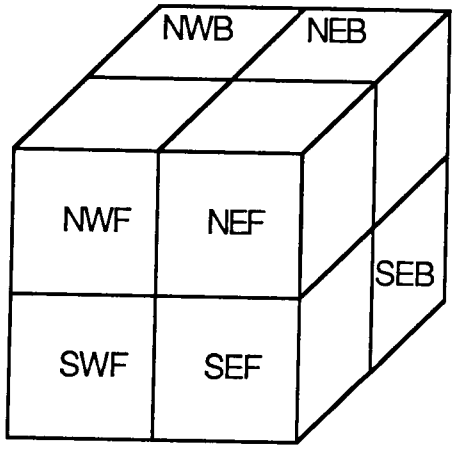


Figure 4.1: The naming convention used to identify the 8 sub-octants contained in an octree node (the sub-octant SWB is not shown).

Since the octree distance transform algorithm involves conceptualizing many ideas in three dimensional space, an effort is made to illustrate the steps involved with appropriate figures for each test case. The choice of one octant over others in these examples have no significance except that it is based on eliminating occlusion effects in these figures so that the point is well-illustrated. The node labeled O always refers to the octree node under consideration while the node labeled N refers to the neighbor being sought.

4.1 Identification of Neighbors in Octrees

A neighbor can share with an octree node a common face, an edge, or a corner. These neighbors will be referred to as *face-neighbors*, *edge-neighbors*, or *corner-neighbors*, respectively. The identification of neighbors belonging to each of these categories is discussed below.

The procedure for identifying a neighbor which shares a common face with an octree node O is similar to that used in the quadtree distance transform algorithm [30]. It consists of traversing the ancestor links of the octree node O until a common ancestor of the node O and its neighbor N is found. The previous path is then retraced, with a modification that each step is a reflection of the corresponding previous step about the common boundary between the regions represented by the two nodes.

The procedure for identifying a neighbor which shares a common edge or a corner with an octree node O is dependent on the octant type of the node O

and the edge or the corner along which a neighbor is being sought. Accordingly the edges and corners of an octree node can be classified into groups so that a common procedure can be adopted to find the neighbors along these directions.

The edges of an octant can be classified into three groups [Fig. 4.2] and the method to find the neighbors for each of these groups is discussed below.

Group I:

The neighbor of an octree node O along an edge belonging to this group lies in the same octant as the parent of the node O. There are three such edges for any octree node [Fig. 4.2(a)].

Group II:

The neighbor adjacent to an edge belonging to this group is such that the ancestors of the neighbor and the octree node O share a common face. There are 6 such edges for any node [Fig. 4.2(b)]. Finding this neighbor, therefore, consists of traversing the ancestor links of the node O until an ancestor is found that is horizontally or vertically adjacent (sharing a common face) to the ancestor of the sought neighbor. The next step consists of retracing the previous path making directly opposite moves with respect to the edge along which a neighbor is sought. For a node whose octant type is NWF, the opposite link with respect to the edge SEE is its off-spring having octant type SEF.

Group III:

The neighbor of an octree node O adjacent to an edge belonging to this group is such that the ancestors of the neighbor and the node O touch each other along a common edge. There are 3 such edges for any octree node [Fig. 4.2(c)]. It

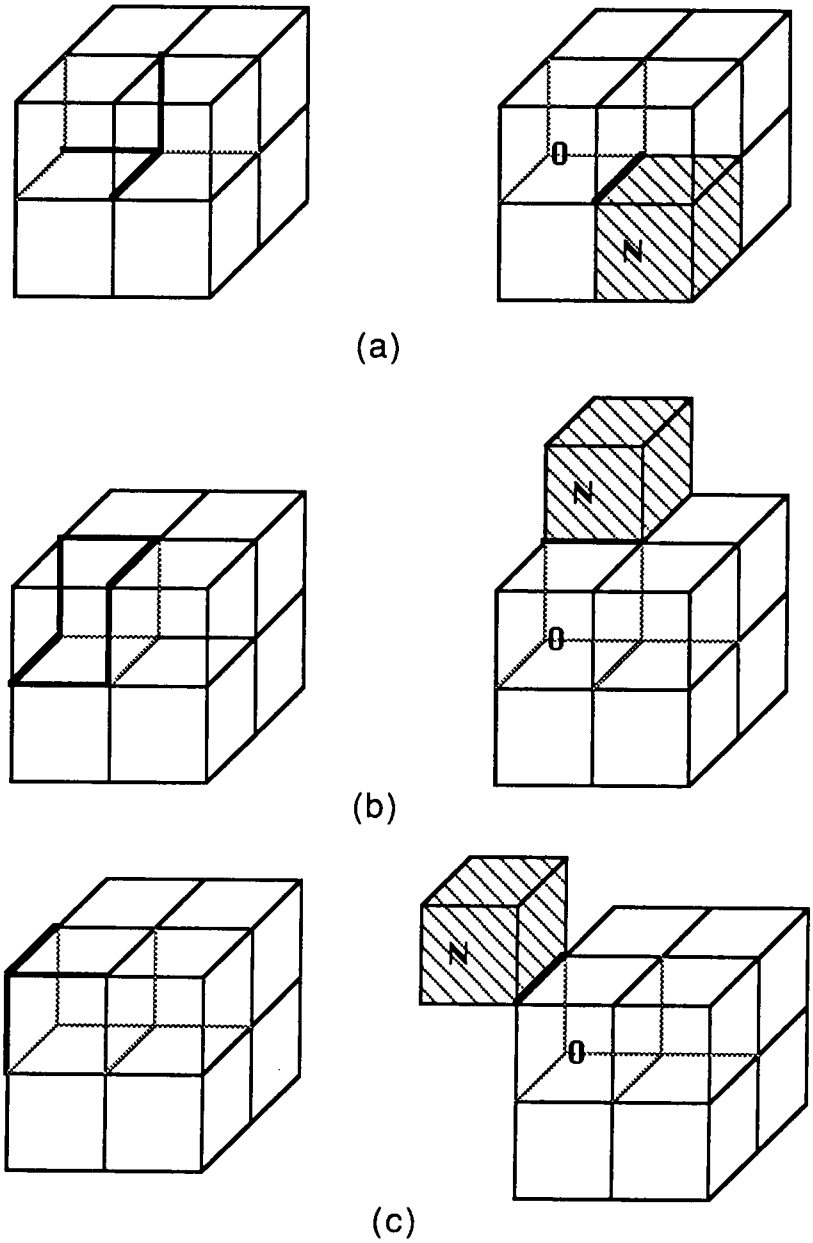


Figure 4.2: An example of the classification of edges into 3 groups of an octree node O whose octant type is NWF. A neighbor adjacent to one such edge belonging to each of these of groups is also shown.

may be observed that neighbors belonging to this group are similar to those of the previous group except that the ancestors share a common edge rather than a common face. Therefore, a similar approach can be adopted to find these neighbors.

The corners of a node can be classified into 4 groups for the purpose of finding the neighbors which touch at a corner [Fig. 4.3]. The method to find the neighbor along a corner belonging to each of these groups is discussed below.

Group I:

The neighbor of an octree node O adjacent to a corner, belonging to this group is a suboctant within the octant represented by the parent of node O and it lies diagonally opposite to the node O. Each node has one such corner. Figure 4.3(a) illustrates such a neighbor along the corner SEB for a node whose octant type is NWF.

Group II:

The neighbor adjacent to a corner belonging to this group is such that the ancestors of node O and the neighbor share a common face. Each node has 3 such corners. Identification of this neighbor, therefore, consists of traversing the ancestor links of node O until an ancestor is found that is horizontally or vertically adjacent (sharing a common face) to the ancestor of the sought neighbor. The next step consists of retracing the previous path making diagonally opposite moves (with respect to the corner). For a node whose octant type is NWF, the opposite link with respect to the corner SEB is its off-spring having octant type

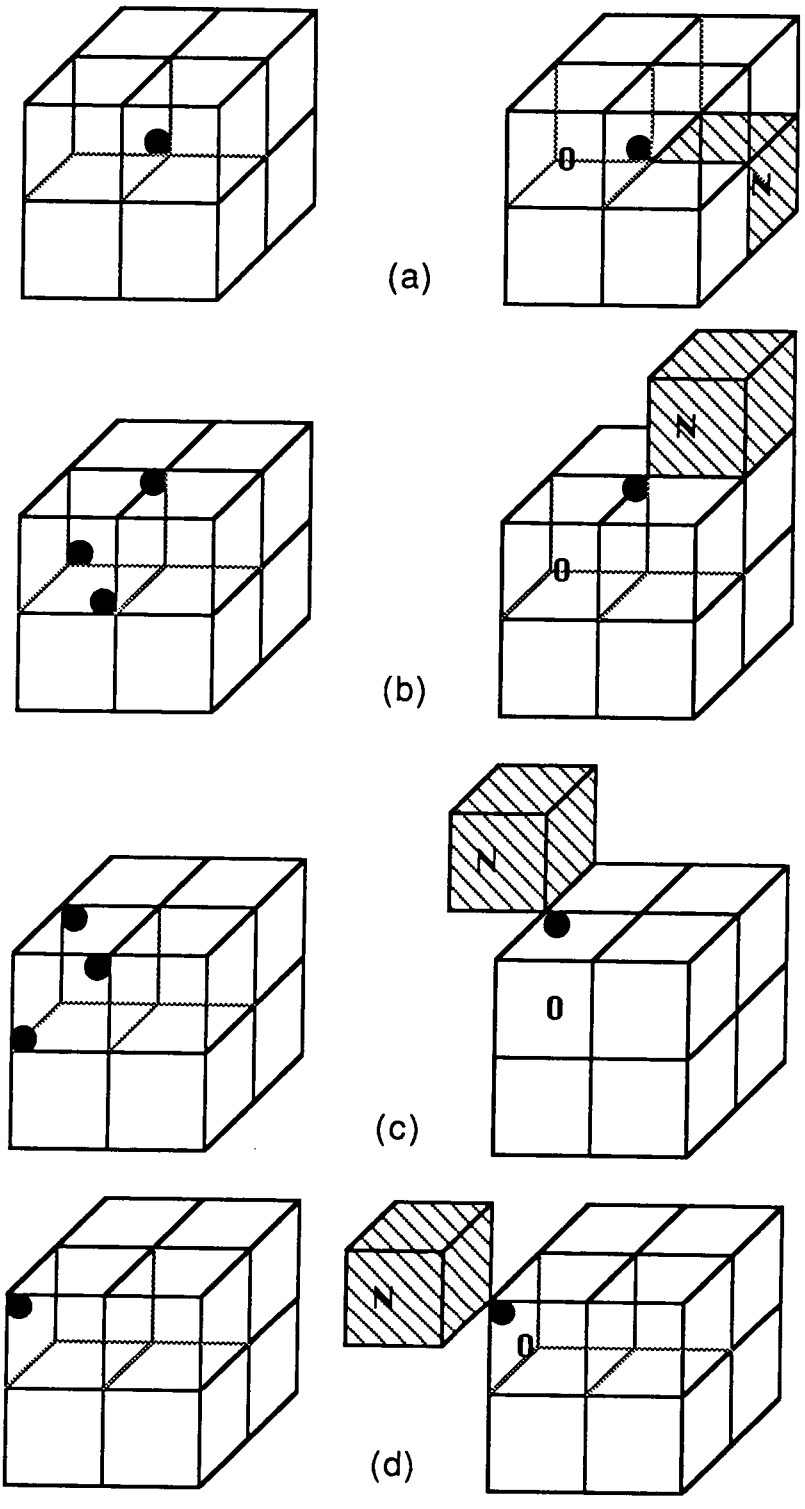


Figure 4.3: An example of the classification of corners into 4 groups of an octree node O whose octant type is NWF. A neighbor adjacent to one such corner belonging to each of these groups is also shown.

SEB. Figure 4.3(b) illustrates such a neighbor along the corner NEB for a node whose octant type is NWF.

Group III:

The neighbor adjacent to a corner belonging to this group is similar to corner-neighbors of the previous group except that the ancestors of octree node O and its neighbor share a common edge [Fig. 4.3(c)] rather than a common face. Each node has 3 such corners. Identification of this neighbor, therefore, consists of traversing the ancestor links of the node O until an ancestor is found that shares a common edge with the ancestor of the sought neighbor. This path is then retraced making diagonally opposite moves. Figure 4.3(c) illustrates such a neighbor along the corner NWB for an octree node whose octant type is NWF.

Group IV:

The neighbor touching a corner and belonging to this group is also the corner-neighbor of some of its ancestors. These ancestors are of the same octant type as the node O itself. The procedure for determining this corner-neighbor consists of finding the ancestor not having the same octant type as its offspring and retracing the previous links making diagonally opposite moves.

4.2 Computation of Distance Transform

Having identified the neighbors of a node, the next step involves exploring each of these neighbors to compute the distance from the center of the octree

node to the boundary of the nearest obstacle block contained in these neighbors. If this distance for an octree node is a lower bound, then further processing of other neighbors can be terminated since this distance is the *distance transform.* WHITE neighbors can be ignored while this distance for a BLACK neighbor is half the size of the octree node itself. What remains to be discussed is the processing of GRAY neighbors. This consists of examining the sub-octants within this GRAY neighbor in a given sequence based on the proximity of the sub-octant to the common boundary between the octree node and its GRAY neighbor. If the sub-octant is a WHITE node, then it can be ignored. For a sub-octant representing a BLACK node, the distance from the center of the octree node to the nearest boundary of this sub-octant is returned. If the sub-octant is itself a GRAY node, say G, then the procedure for exploring its ancestor GRAY neighbor is recursively applied to the sub-octants within this node G.

The sequence (order) in which the sub-octants within a GRAY neighbor is examined is dependent on the common boundary being a face, an edge or, a corner. The ordering is given below for each of the three cases. The ordering is in decreasing order of priority. But it should be pointed out that if at any time during the processing of a GRAY neighbor, a sub-octant representing a BLACK node is found, then further processing of this GRAY neighbor is immediately terminated.

The sub-octants contained in a GRAY neighbor sharing a common face with an octree node are examined in the following order [Fig. 4.4(a)].

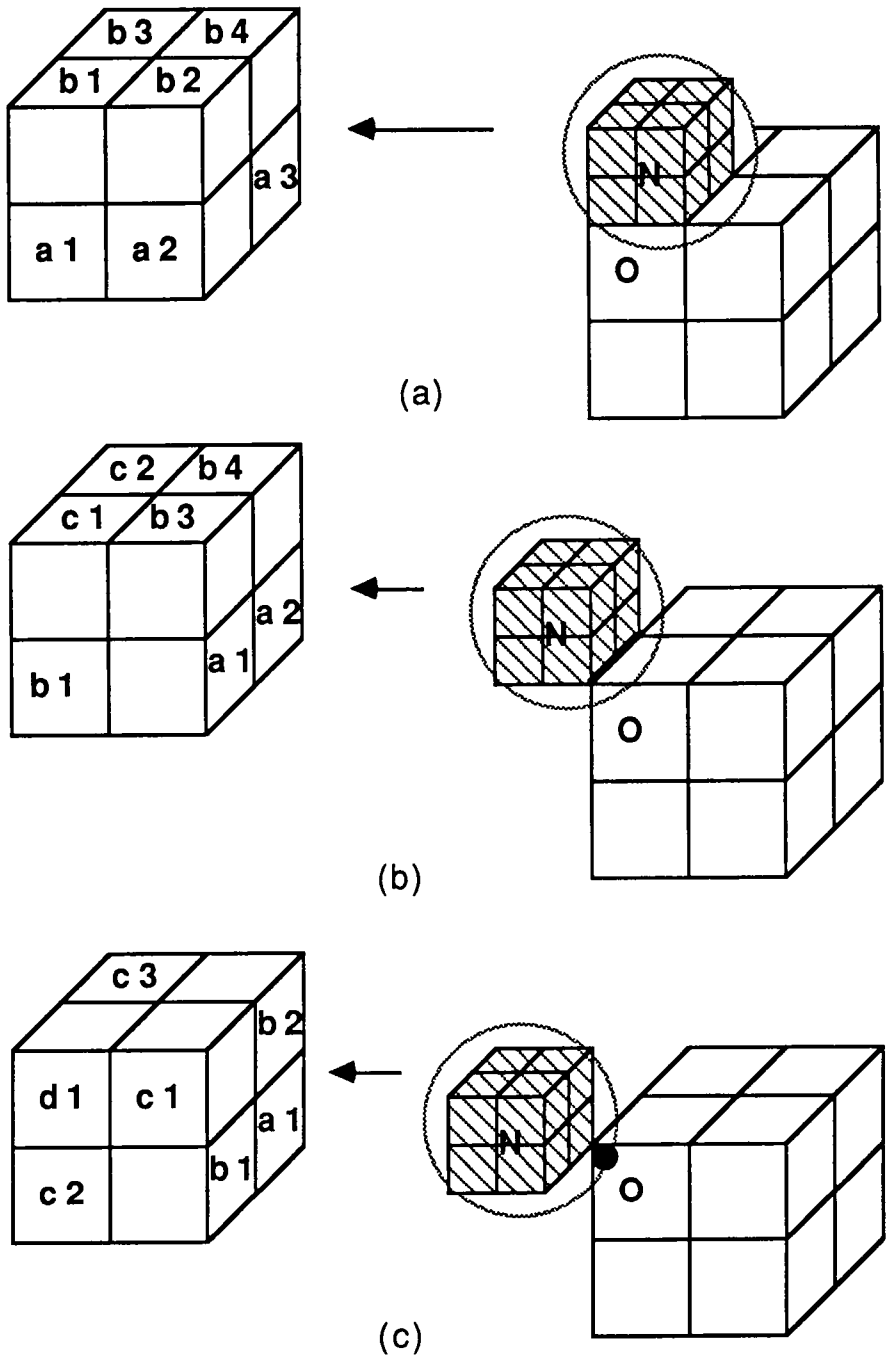


Figure 4.4: Ordering of the sub-octants contained in a GRAY neighbor which shares (a) a common face, (b) a common edge, and (c) a common corner with an octree node. These sub-octants are processed according to this ordering.

1. The 4 sub-octants that are adjacent to the common face shared by the octree node and its neighbor are first examined. The suboctants named a_1 through a_4 in Fig. 4.4(a) (a_4 not shown) of a GRAY neighbor adjacent to the face N (North) of a node whose octant type is NWF, belong to this group.

2. The remaining 4 suboctants that are further away from the common face shared by the node and its neighbor are the second set of nodes to be explored. The suboctants named b_1 through b_4 in Fig. 4.4(a) of a GRAY neighbor adjacent to the face N (North) of a node whose octant type is NWF, belong to this group.

The sub-octants contained in a GRAY neighbor sharing a common edge with an octree node are examined in the following order [Fig. 4.2(b)].

1. The two sub-octants that are adjacent to the common edge shared by the octree node and its neighbor are first examined. The sub-octants named a_1 and a_2 in Fig. 4.4(b) of a GRAY neighbor adjacent to the edge NWE of a node whose octant type is NWF, belong to this group.

2. The sub-octants of this group are either horizontally or vertically adjacent to the sub-octants belonging to the previous group (nodes a_1 and a_2 in Fig.4.4(b)) and share a common face with the latter. There are 4 such sub-octants within a GRAY edge-neighbor. The sub-octants named b_1 through b_4 in Fig. 4.4(b) (b_2 not shown) of a GRAY neighbor adjacent to the edge NWE of a node whose octant type is NWF, belong to this group.

3. The remaining two sub-octants that are further away from the common edge shared by the node and its neighbor are then examined. The sub-octants named c_1 and c_2 in Fig. 4.4(b) of a GRAY neighbor adjacent to the edge NWE of a node whose octant type is NWF, belong to this group.

The sub-octants contained in a GRAY neighbor sharing a corner with an octree node are examined in the following order. [Fig. 4.4(c)].

1. The sub-octant that touches the corner shared by the octree node and its neighbor belongs to this group. The sub-octant named a_1 in Fig. 4.4(c) of a GRAY neighbor adjacent to the corner NWF of a node whose octant type is NWF, belongs to this group.

2. The sub-octants of this group are horizontally or vertically adjacent and share a common face with the sub-octant of the previous group (node a_1 in Fig. 4.4(c)). There are 3 such sub-octants within a GRAY corner-neighbor. The sub-octants named b_1 through b_3 in Fig.4.4(c) (b_3 not shown) of a GRAY neighbor adjacent to the corner NWF of a node whose octant type is NWF, belong to this group.

3. The sub-octants of this group are also adjacent to the sub-octant of the first group (node a_1 in Fig. 4.4(c)), but share a common edge rather than a common face. There are 3 such sub-octants within a GRAY corner-neighbor. The sub-octants named c_1 through c_3 in Fig. 4.4(c) of a GRAY neighbor adjacent to the corner NWF of a node whose octant type is NWF, belong to this group.

4. The remaining sub-octant that is the furthest away from the common corner shared by the octree node and its neighbor belong to this group. The sub-octant named d_1 in Fig. 4.4(c) of a GRAY neighbor adjacent to the corner NWF of a node whose octant type is NWF, belongs to this group.

4.3 Implementation

This section gives the implementation of this approach in an algorithmic form. A procedure called *3D-Distance-Transform (Octnode, Size)* was written to compute the distance transform of the free nodes in the octree. *Octnode* is a pointer to the root of the octree and *Size* is the size of the 3-D image array (in number of pixel units) which the octree represents. The algorithm traverses the tree in a post order and controls the exploration of the 26 neighbors of a WHITE node. The algorithm, in Pascal-like syntax, is given below. Explanation of the individual steps follow the algorithm.

```
Procedure 3D_DIST_TRANSFORM(Octnode, Size);
  begin
    if OCTANT_TYPE(Octnode) == GRAY then
      for I ∈ {NWF, NEF, SWF, SEF, NWB, NEB, SWB, SEB} do
        DIST(Octnode) = 0;        /* Undefined for GRAY nodes */
        3D_DIST_TRANSFORM(CHILD(Octnode, I), Size/2);
    else
```

```
    if OCTANT_TYPE(Octnode) == BLACK then
      DIST(Octnode) = 0;        /* Undefined for BLACK nodes */
    else        /* free node; compute distance transform */
    begin
      L_bound = Size/2;          /* Lower bound on distance transform */
      Sofar_dist = 1.5 × Size;    /* Upper bound on distance transform */
      Sofar_dist = DIST_FACE_NEIGHBORS(Octnode,L_bound,Sofar_dist);
      if Sofar_dist > L_bound then
       Sofar_dist=DIST_EDGE_NEIGHBORS(Octnode,L_bound,Sofar_dist);
      if Sofar_dist > L_bound then
       Sofar_dist = DIST_COR_NEIGHBORS(Octnode,L_bound,Sofar_dist);
      DIST(Octnode) = Sofar_dist;
    end else;
  end procedure;

Function DIST_FACE_NEIGHBORS(Freenode, L_bound, Sofar_dist);
  begin
    Found = FALSE;
    while not Found and Face ∈ Lfaces do
    begin
      Neighbor = FACE_NEIGHBOR(Freenode, Face);
      if EXISTS(Neighbor) and OCTANT_TYPE(Neighbor) ≠ WHITE then
        if OCTANT_TYPE(Neighbor) == BLACK then
```

```
            begin
               return( L_bound );
               Found = TRUE;
            end;
            else          /* GRAY face-neighbor */
            begin
               New_dist = DIST_GRAY_FACE_NEIGHBOR(Neighbor, Face);
               if New_dist == L_bound then
               begin
                  return( L_bound );
                  Found = TRUE;
               end;
               else
                  if New_dist < Sofar_dist then
                     Sofar_dist = New_dist;
            end else;
         Face = NEXT(Face);
      end while;
      return( Sofar_dist );
   end function;
```

L_bound is the lower bound for the distance of a free node from an obstacle block in the octree. This distance is given by one-half the size of the node itself. Since the distance of a free node from a BLACK neighbor is one-half the size of the

free node, a BLACK neighbor is gauranteed to return the *distance transform.* Therefore, further processing of the remaining neighbors of the free node can be terminated. This is implemented by setting a flag called *Found. Sofar_dist* is the distance of a free node from the nearest neighbor among the neighbors sofar examined. This parameter is computed by propagating the smallest value of distance among the distances computed for each neighbor. Initially, *Sofar_dist* is set to the largest possible value for distance transform of the node. This upper bound on the distance transform is given by 1.5 times the size of the node.

The function DIST_FACE_NEIGHBORS () explores the neighbors of a free node along the 6 possible faces. It determines the nearest neighbor from among the *face-neighbors* and other neighbors examined sofar and returns the distance of this neighbor from the octree node under consideration. This function is implemented as a set of lower level functions called FACE_NEIGHBOR () and DIST_GRAY_FACE_NEIGHBOR () which explore one neighbor at a time. The function FACE_NEIGHBOR (octnode, face) returns the *face-neighbor* of the "octnode" along the direction "face". If this neighbor is a GRAY neighbor, then the function DIST_GRAY_FACE_NEIGHBOR () is invoked. It returns the distance from the center of the "octnode" to the boundary of the sub-octant contained in this GRAY neighbor closest to "octnode". It may recalled that WHITE neighbors are ignored in the evaluation of the octree distance transform.

Lfaces is a list consisting of all the faces ordered in a pre-defined sequence. The function NEXT(Face) returns the member of the list next to "Face", corresponding to this ordering. The Boolean function EXISTS(Octnode) returns true when "Octnode" is a valid node in the octree.

The functions DIST_EDGE_NEIGHBORS () and DIST_COR_NEIGHBORS () are similar to DIST_FACE_NEIGHBORS () described above except that they explore the edge and corner neighbors of an octree node. Hence, they are not given here.

CHAPTER 5

EXPERIMENTAL RESULTS

In this chapter, the experimental results of using the path planning approach developed in this study are presented. The algorithm was implemented in C language on a VAX 11/780 computer. The size of the images used in these experiments is 128 × 128.

5.1 2-D Path Planning

The quadtree representation of a 2-D workspace was generated from an image of the scene. The path generated by this algorithm is shown by solid lines. The lines were generated by joining the centers of the regions representing the quadtree nodes along the path. Figure 5.1 shows the image of a simulated scene with blocks, and its thresholded binary image is shown in Fig. 5.2. Figure 5.3 is the image obtained by enlarging the obstacles in Fig. 5.2 corresponding to the size of the moving object (robot). The enlargement factor used was 5 pixels. The time taken to increase the size of the obstacles for a 128 × 128 image is typically about 1 second. The paths generated using this image from a start point S (top left) to a goal point G (bottom left) for different clearances are illustrated in Figs. 5.4 through 5.6. It can be observed that, with an increase in the *clearance*

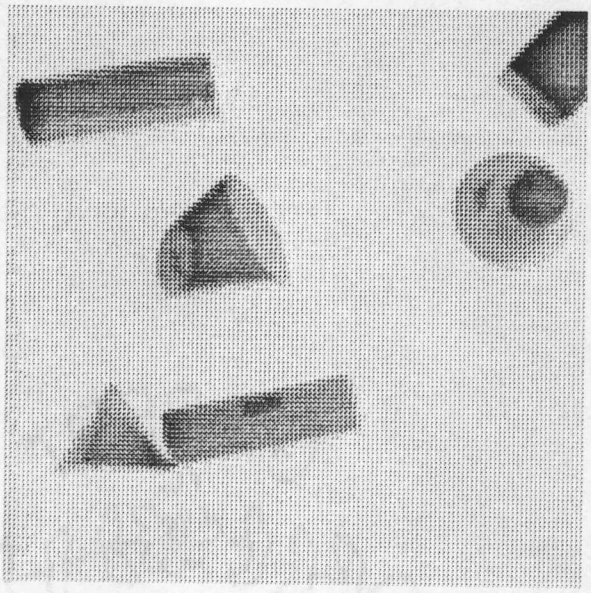


Figure 5.1: Original image of a scene (set # 1).

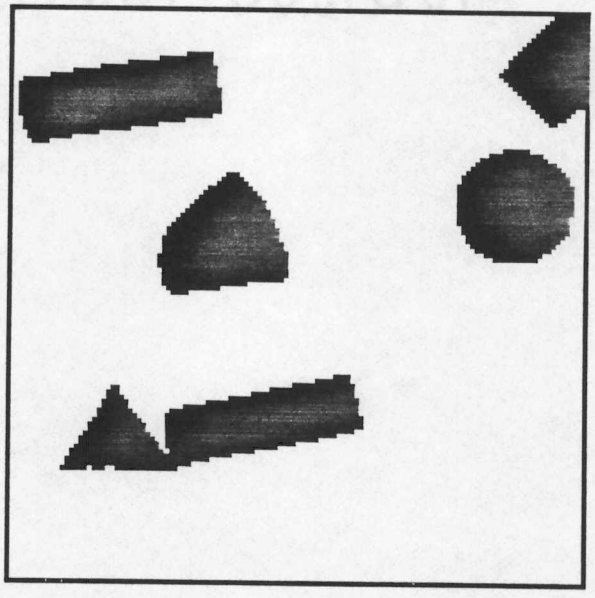


Figure 5.2: Binary image generated by thresholding.

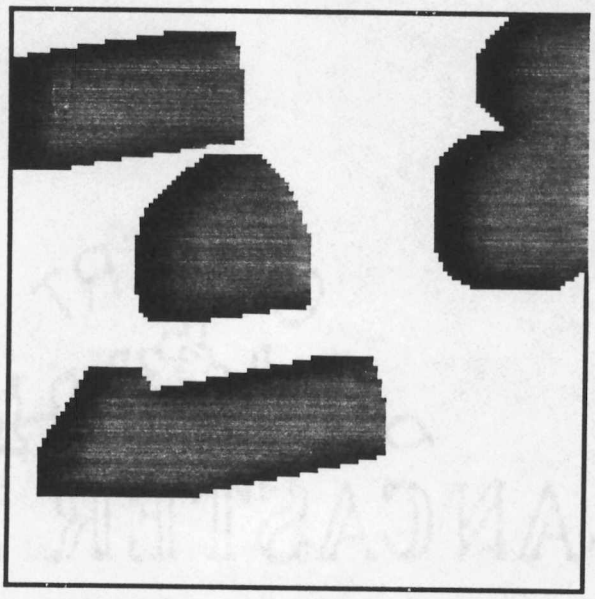


Figure 5.3: Binary image showing pseudo-obstacles.

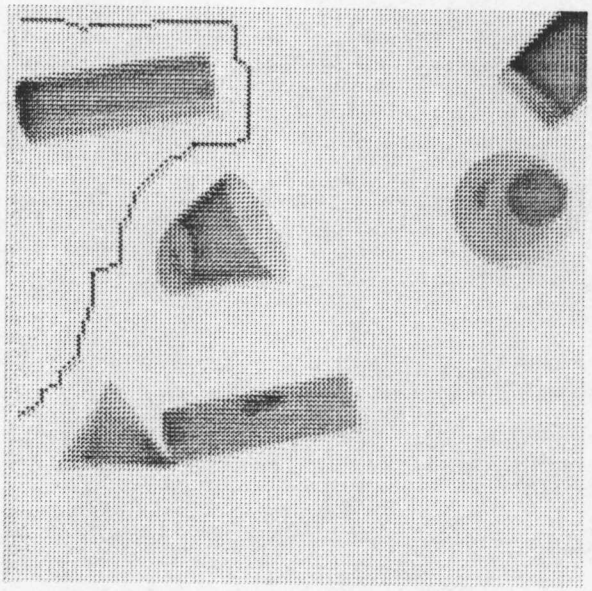


Figure 5.4: Generated path shown by a solid line (CF = 0.0).

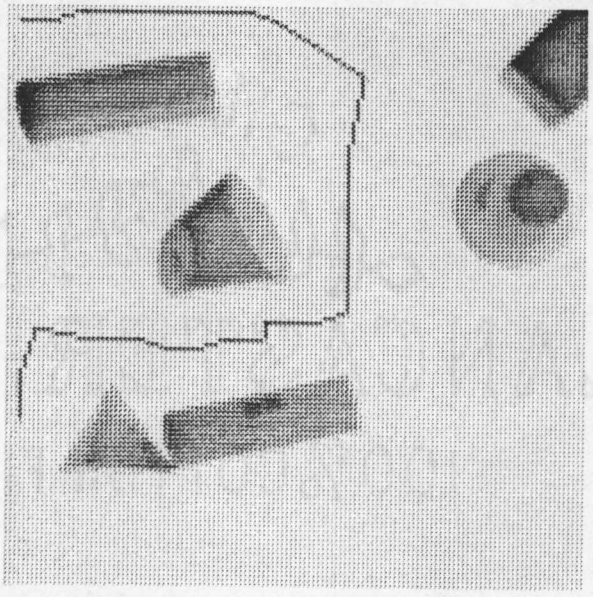


Figure 5.5: Generated path shown by a solid line (CF = 0.5).

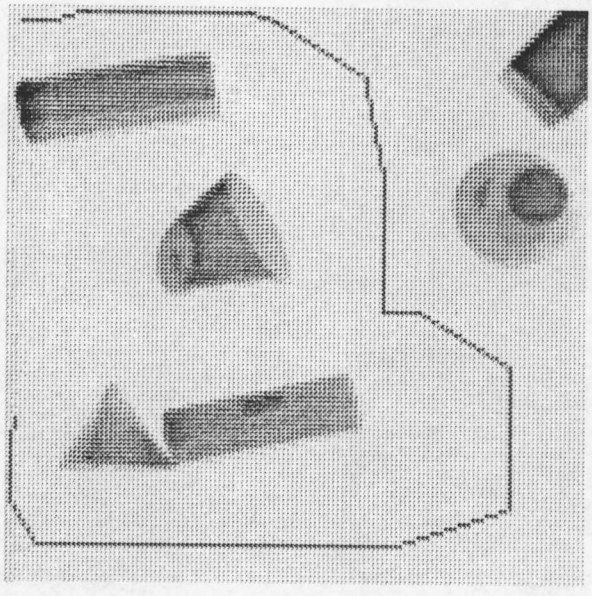


Figure 5.6: Generated path shown by a solid line (CF = 1.0).

factor (CF), the distance along the path is sacrificed for better clearance. The image of a cluttered workspace and a path generated using this algorithm for a CF = 1.0 are shown in Figs. 5.7, and 5.8, respectively. An increase in the search time in this case is due to an increase in the number of nodes expanded during the search.

Figure 5.9 gives the CPU time (in seconds) required to generate the quadtree and the time taken to generate the path, the number of nodes in the quadtree, and the number of nodes expanded during the search. Clearly, the search time is a function of the number of nodes expanded during the search. As expected, the processing time is greater for large values of the clearance factor. This increase in time is due to the fact that the power of the heuristic used in the A^* search depends upon the average deviation of the minimum cost path from the straight line path. The deviation is highest for a clearance factor of 0 and decreases as the value of the clearance factor increases. Since the path generation is based on the A^* search, the algorithm will find a path if one exists and will indicate failure if no path exists.

5.2 3-D Path Planning

The 3-D description of the workspace was generated using three orthogonal projections of the scene. Images representing the three views (front, top, and side) of the workspace were used as the three projections. Ideally, the three views have to be obtained from 3 identical orthogonal cameras. The optical axes

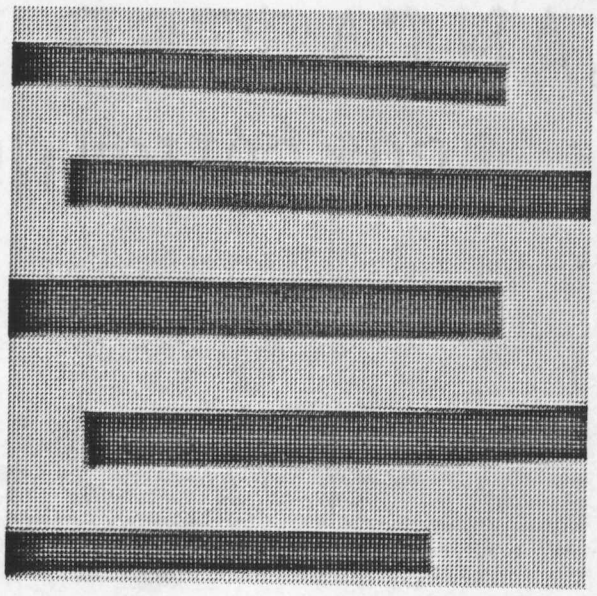


Figure 5.7: Original image of the scene (set # 2).

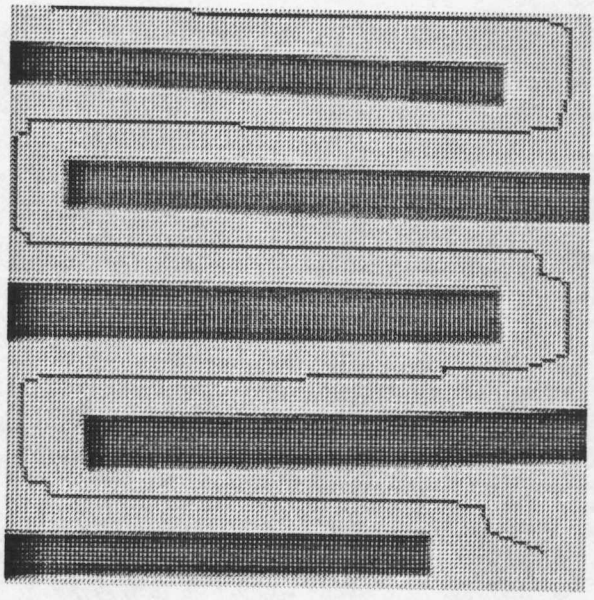


Figure 5.8: Generated path shown by a solid line (CF = 1.0).

Fig. No.	*No. of Quad-tree nodes*	*Time to build quadtree*	*Clearance Factor*	*No. of Nodes expanded*	*Search time*
5.4	1581	6.25	0.0	169	0.90
5.5	1581	6.25	0.5	567	4.85
5.6	1581	6.25	1.0	524	4.24
5.8	3033	6.55	1.0	1047	9.95

Figure 5.9: Statistics of 2-D Path Planning.

of the cameras should be perpendicular to each other. The point at which the three axes intersect on the 3-D workspace should be equi-distant from the center of the camera lenses. The three images were generated using a single camera mounted on a robot. The images were taken at three orthogonal positions of the camera lens with respect to the workspace. Then a path was generated using the algorithm described in the previous sections between a start point S (top left and closer to the back plane on the 3-D workspace) and a goal point G (bottom right and closer to the front plane) for a clearance factor of 0 and 1. The generated path consists of nodes in the octree which constitute a path between S and G. A straight path was reconstructed by joining the centers of successive nodes along the path. For the purpose of illustration, this 3-D path was decomposed into a set of three 2-D path segments. These path segments correspond to the projections of the 3-D path on the three orthogonal planes. Figures 5.10 through 5.12 show the 2-D path segments of the 3-D path generated

between S and G for a clearance factor of 0. Figures 5.13 through 5.15 illustrate the 2-D path segments of the 3-D path between S and G for a clearance factor of 1. Figures 5.10 through 5.15 were generated by superimposing these 2-D path segments on the corresponding views of the 3-D workspace. A 3-D path will be considered collision-free if every point along the path is collision-free in at least one of the three orthogonal projections. It can be observed that the paths shown in Figs. 5.10 through 5.15 are indeed collision-free. Figure 5.16 outlines the statistics of some of the parameters observed in generating the 3-D path given in Figs. 5.10 through 5.15.

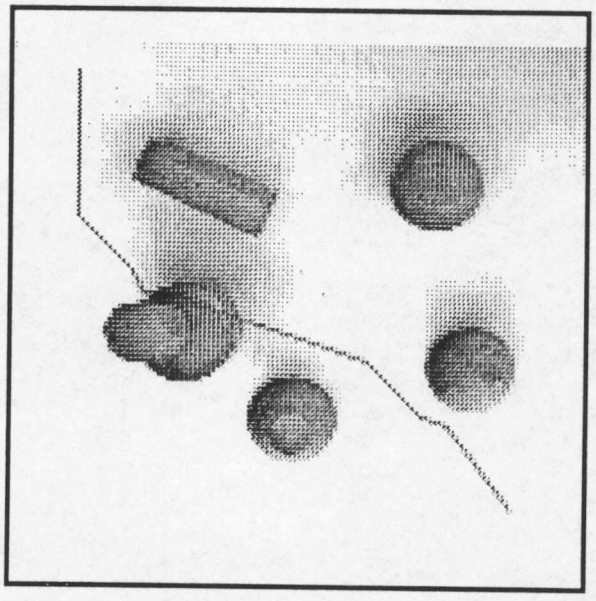


Figure 5.10: Top view of a 3-D workspace. The projection of the 3-D path on this plane is shown by a solid line (CF = 0.0).

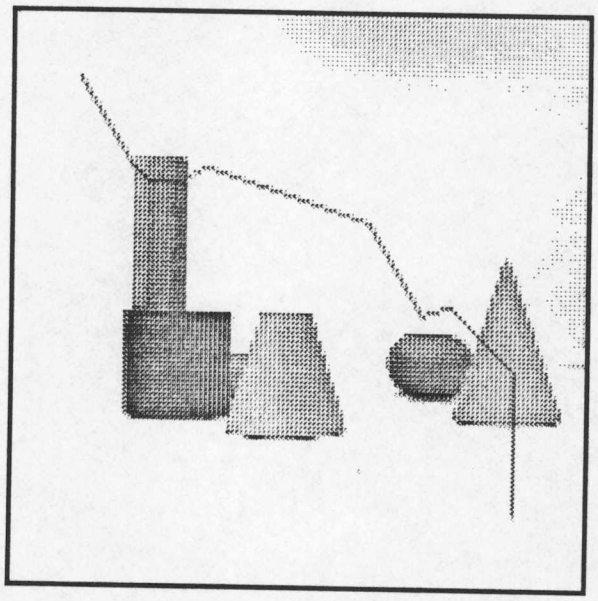


Figure 5.11: Front view of a 3-D workspace. The projection of the 3-D path on this plane is shown by a solid line (CF = 0.0).

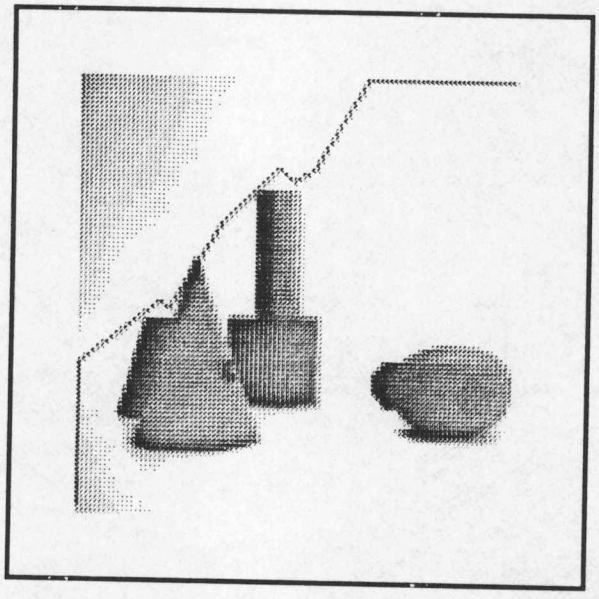


Figure 5.12: Side view of a 3-D workspace. The projection of the 3-D path on this plane is shown by a solid line (CF = 0.0).

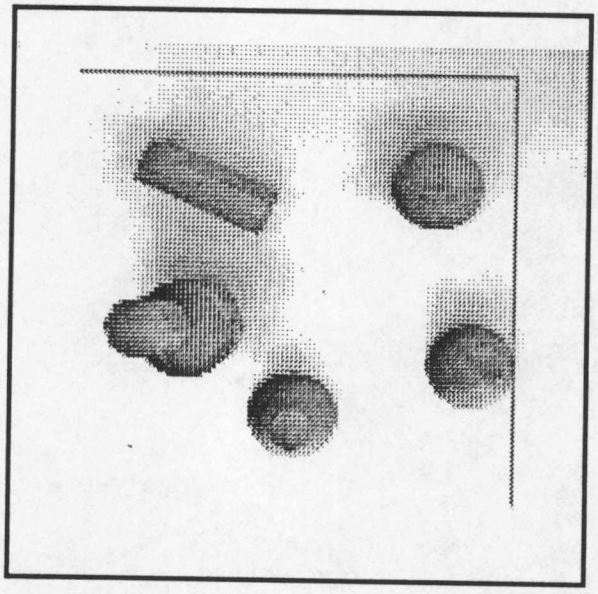


Figure 5.13: Top view of a 3-D workspace. The projection of the 3-D path on this plane is shown by a solid line (CF = 1.0).

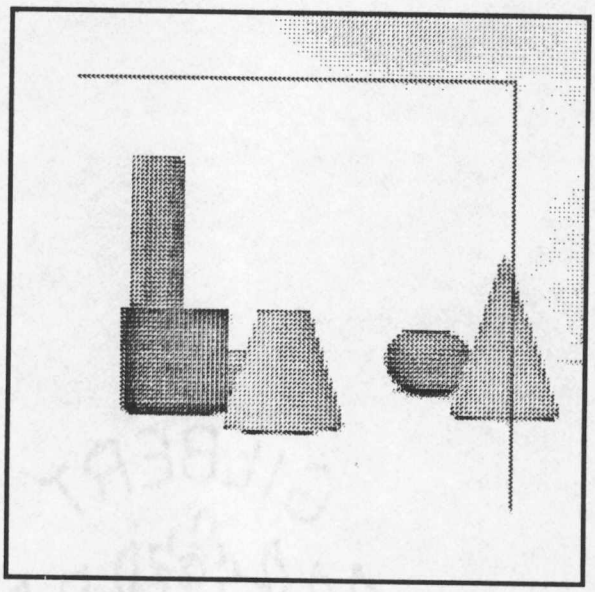


Figure 5.14: Front view of a 3-D workspace. The projection of the 3-D path on this plane is shown by a solid line (CF = 1.0).

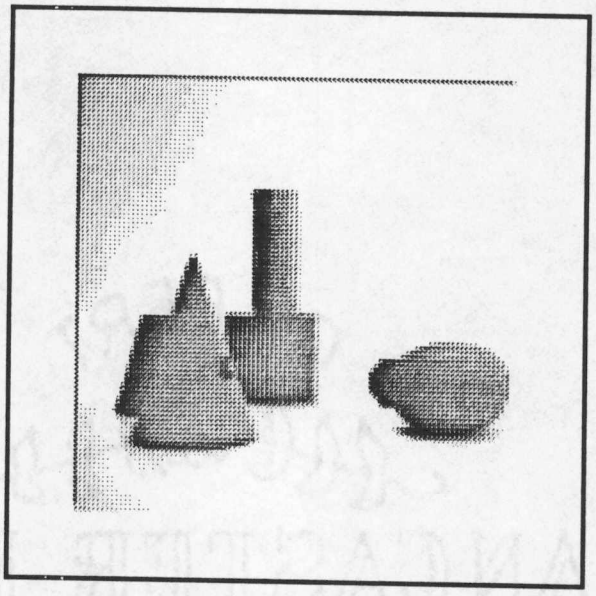


Figure 5.15: Side view of a 3-D workspace. The projection of the 3-D path on this plane is shown by a solid line (CF = 1.0).

View	Time (sec)	Number of Nodes
Front	6.37	905
Top	6.5	1313
Side	6.44	1121

(a) Quadtree Generation of three views of the 3-D workspace.

Time for octree generation (sec)	10.68
Time to compute dist. transform (sec)	43.39
Number of nodes in octree	28089

(b) Octree Generation from Quadtrees.

	$CF = 0$	$CF = 1$
Time for path generation (sec)	42.41	773.37
No. of nodes expanded during search	1123	3681

(c) Path Generation.

Figure 5.16: Statistics of 3-D Path Planning.

5.3 Implementation of the Algorithm on the Robot

An experimental system was set up on a robot (Cincinnati Milacron T^3 726) to verify this approach to path planning. The robot consisted of an arm, on which a CCD camera was mounted, and a gripper [Fig. 5.17]. The experiment consisted of moving a tool attached to the gripper in a 2-D or a 3-D workspace without colliding with the obstacles.

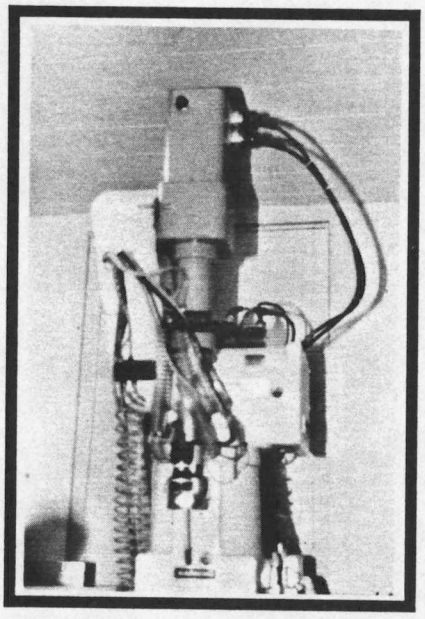


Figure 5.17: Cincinnati Milacron T^3 726 robot used in the experiment.

A 2-D and a 3-D workspace was chosen for path planning based on 2 considerations: (1) the accessability of every point in the workspace to the robot, and (2) the ability of the robot to acquire the image(s) of the workspace (while a single image of the workspace was sufficient for 2-D path planning, 3-D path planning

required 3 orthogonal views of the workspace). A path was planned between 2 points in the workspace using the algorithm developed in this thesis. The path thus generated, consisting of a sequence of points, was defined with respect to an image coordinate system. This is called the *pixel* coordinate system in the case of 2-D path planning. The image coordinate system in 3-D path planning is called the *voxel* (volume element) coordinate system. The origin and direction of the axes of these coordinate systems in the corresponding workspaces are given in Fig. 5.18. The distance between 2 neighboring pixels or voxels was taken as the unit length, and, consequently, the distance between 2 points in these coordinate systems is measured in pixel or voxel units.

Once a path was determined, the task then consisted of moving the tool along this path. This was accomplished using the robot motion routines. The motion routines move the tool to a given position in a given coordinate system. However, this coordinate system had to be defined with respect to one of the known robot coordinate systems. Therefore, a new coordinate system, called the *world coordinate system*, was defined for both 2-D and 3-D path planning. The distance between 2 points in this coordinate system was measured in inches. The relative position of the world coordinate system (2-D and 3-D) with respect to the image coordinate system (pixel and voxel) is given in Fig. 5.19.

The problem was then reduced to transforming a point defined in the pixel or voxel coordinate system to the 2-D or 3-D world coordinate system so that the robot motion routines could be used to move the tool along the path. The

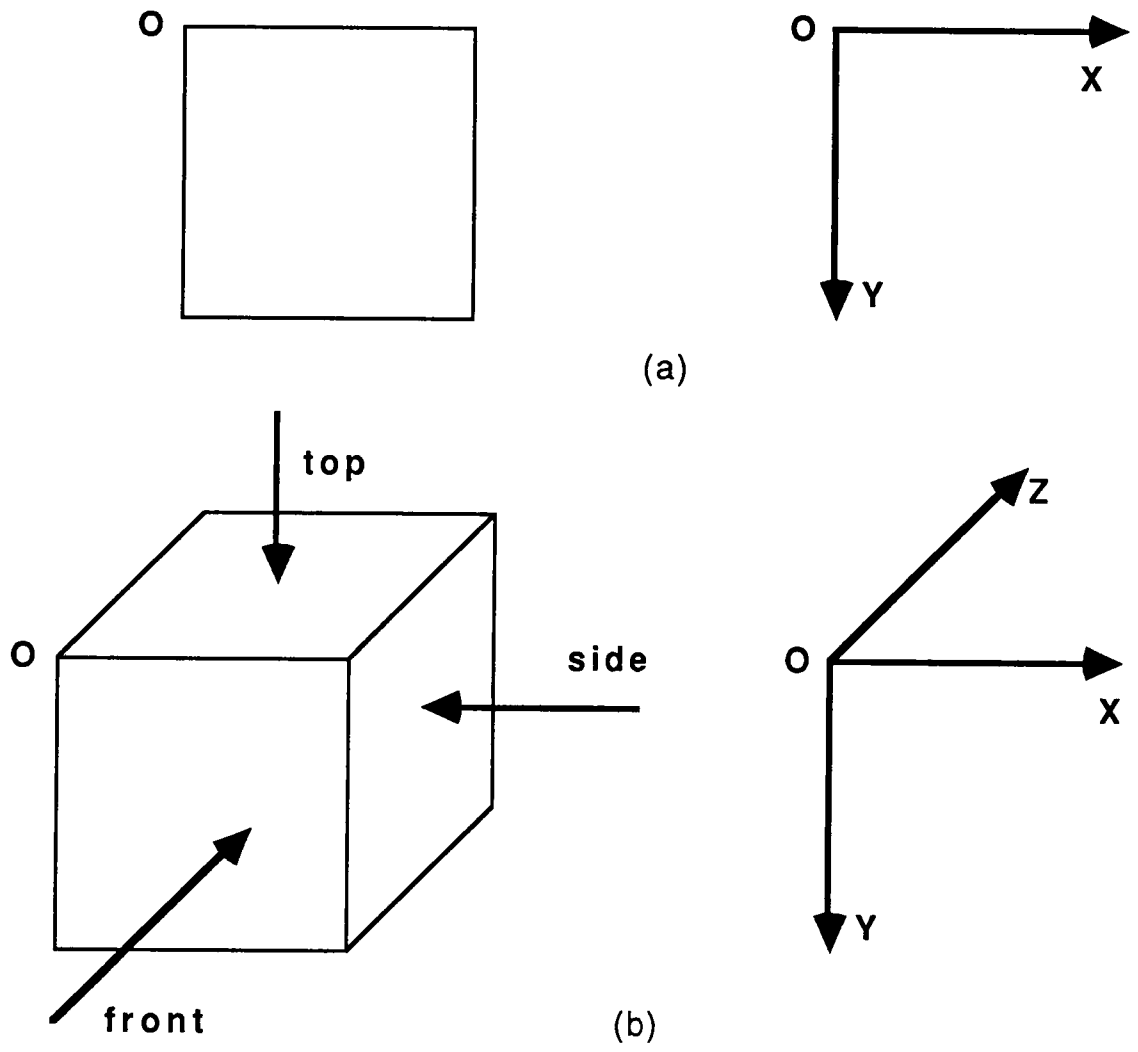


Figure 5.18: (a) The 2-D workspace and the *pixel* coordinate system, and (b) the 3-D workspace and the *voxel* coordinate system.

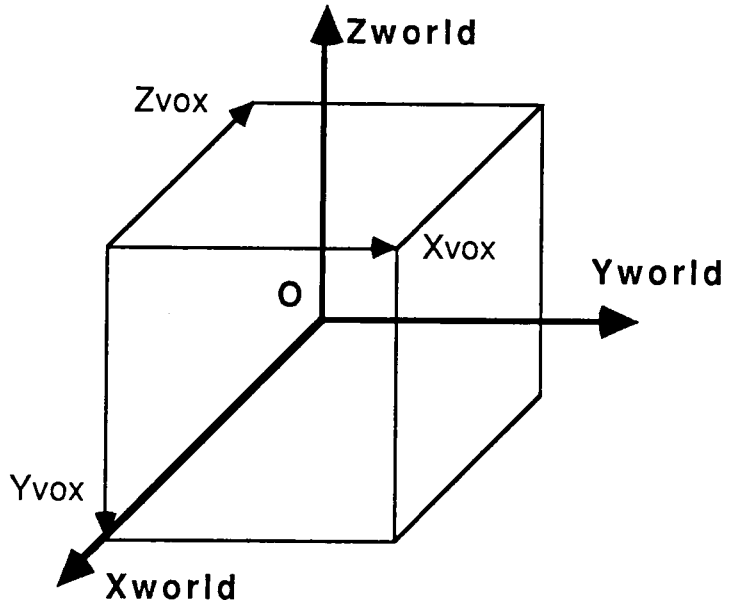


Figure 5.19: Relative position of the *world* coordinate system with respect to the (pixel and voxel) image coordinate system.

transformation of a point in the voxel to the 3-D world coordinate system is discussed below. However, it has to be pointed out that this procedure is similar to the transformation of a point defined in the pixel coordinate system to the 2-D world coordinate system, except that the third dimension, namely, the Z axis, is set to a constant value.

The transformation of a point in the voxel to the 3-D world coordinate system involved computing the *ratio* of the distance between 2 points in the voxel coordinate system to the corresponding distance (in inches) in the 3-D world coordinate system. This ratio will be denoted by RWV (Ratio of world to voxel coordinate systems).

The geometry and optical characteristics of the camera were used to compute this ratio. From Fig. 5.20, it follows that

$$\frac{r}{\lambda} = \frac{L}{D} \tag{5.1}$$

where

D Distance of the origin of the 3-D world coordinate system from the center of the camera lens (in inches)

L Length of the workspace (in inches)

λ Focal length of the camera (in inches)

r Image size (in inches) at the focal plane of the camera

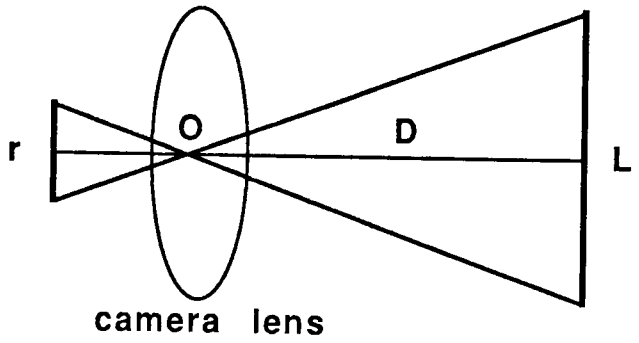


Figure 5.20: Image formation at the focal plane of the camera lens.

The parameter r can be computed from the relationship:

$$r = \frac{img\_res}{pp\_inch},$$

where

img_res Resolution of the image (in pixels) at the focal plane of the camera lens.

pp_inch Pixels per inch on the focal plane of the camera.

From Eqn. 5.1,

$$L = \left(\frac{D}{\lambda}\right) r. \tag{5.2}$$

From Eqn. 5.2, it follows that

$$RWV = \frac{L}{Size}$$

where *Size* Size of the image (in pixels) used in path planning.

Thus, to compute RWV, the parameters λ, *img_res*, and *pp_inch* have to be determined. These parameters can be obtained from the specifications of the

camera lens. The transformation of a point in the voxel coordinate system to a point in the 3-D world coordinate system is given by the following relationships.

$$X_{world} = -Z_{vox} * RWV + \frac{L}{2} \tag{5.3}$$

$$Y_{world} = X_{vox} * RWV - \frac{L}{2} \tag{5.4}$$

$$Z_{world} = -Y_{vox} * RWV + \frac{L}{2} \tag{5.5}$$

The image coordinate points generated by the algorithm were then transformed to world coordinates (using the Eqns. 5.3 through 5.5), and the motion routines were invoked to traverse the generated path.

Figures 5.21 through 5.27 show the robot at various stages of path planning. The robot is shown with its camera positioned to acquire three orthogonal views of a 3-D workspace [Figs. 5.21 through 5.23]. The obstacles in the workspace consisted of a set of cylinders placed adjacent to each other to form a barrier. The task of this experiment was to move the tool attached to gripper across this barrier. The robot with the tool at the start, an intermediate, and the goal positions along the path are shown in Figs. 5.24 through 5.26. The robot acquiring the image of a 2-D workspace, and the robot at the start, an intermediate, and the goal positions of a 2-D path generated by the algorithm are shown in Fig. 5.27.

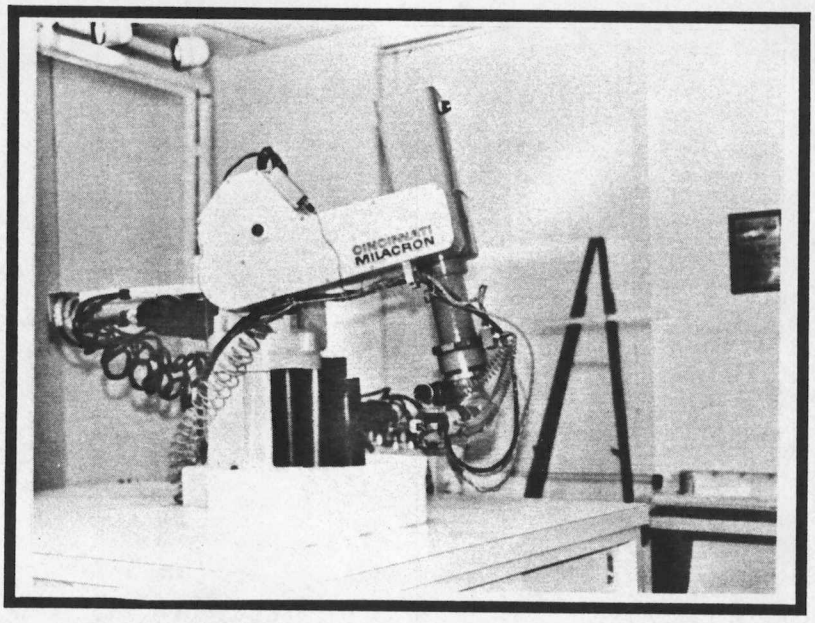


Figure 5.21: The robot acquiring the front view of a 3-D workspace.

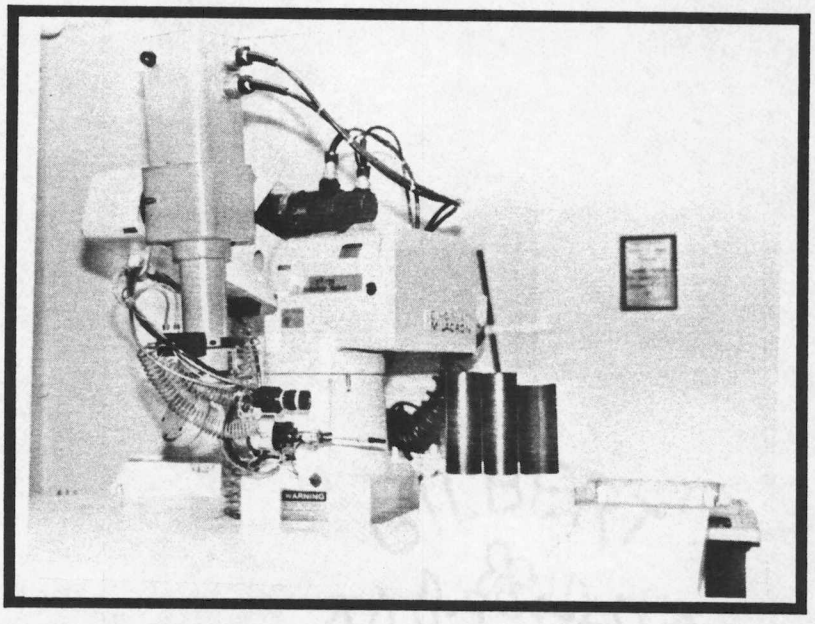


Figure 5.22: The robot acquiring the side view of a 3-D workspace.

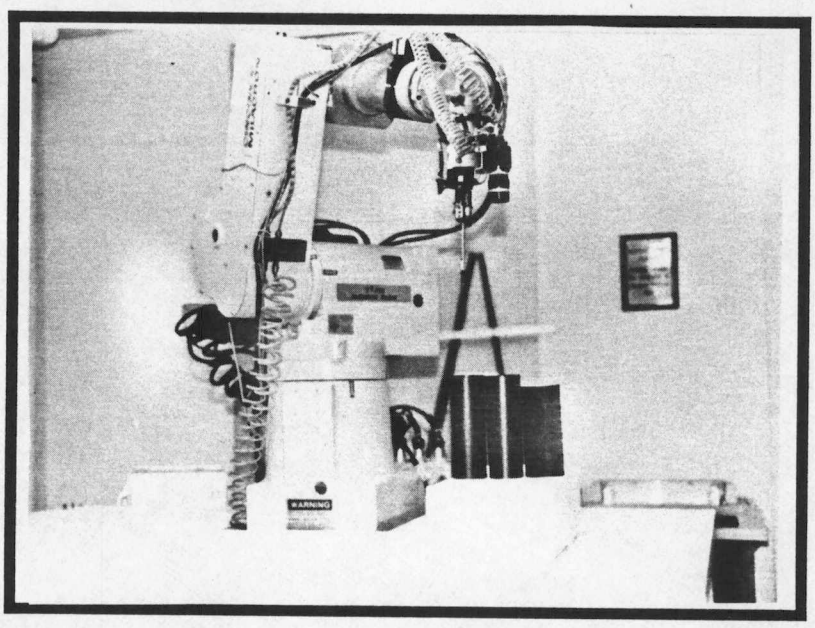


Figure 5.23: The robot acquiring the top view of a 3-D workspace.

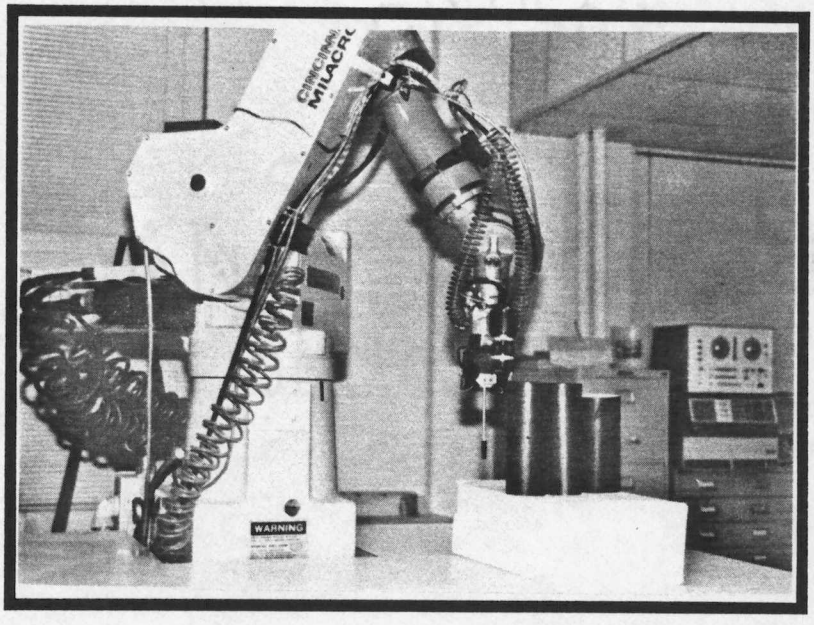


Figure 5.24: The robot at the start position of a 3-D path.

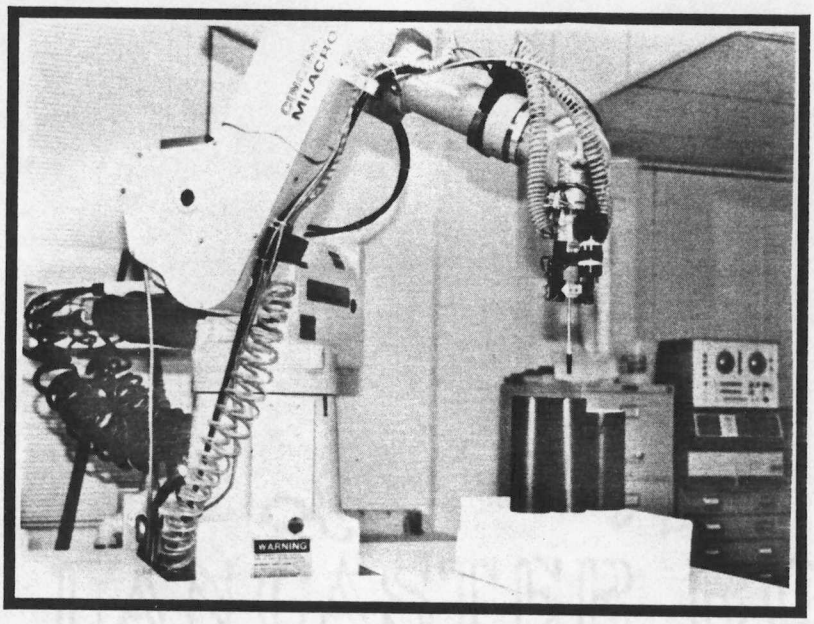


Figure 5.25: The robot at an intermediate position of a 3-D path.

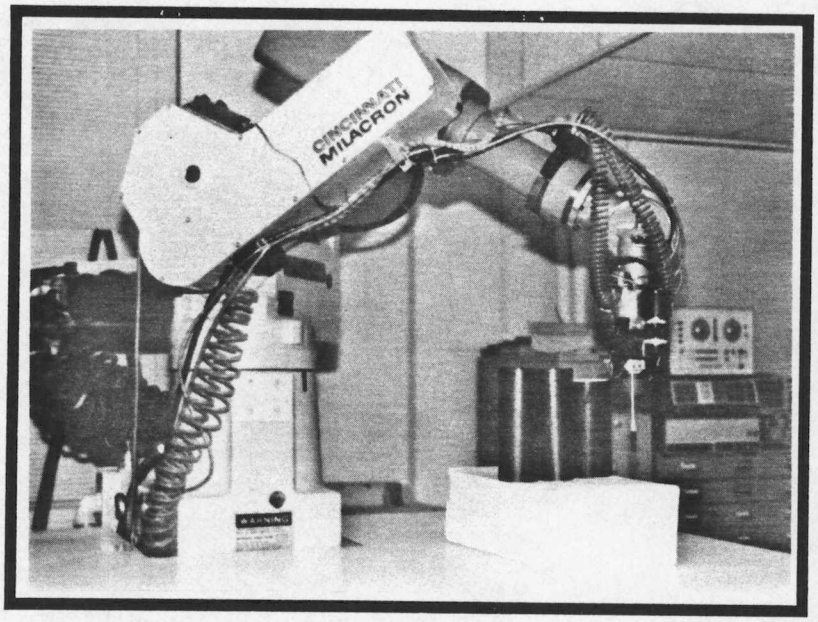


Figure 5.26: The robot at the goal position of a 3-D path.

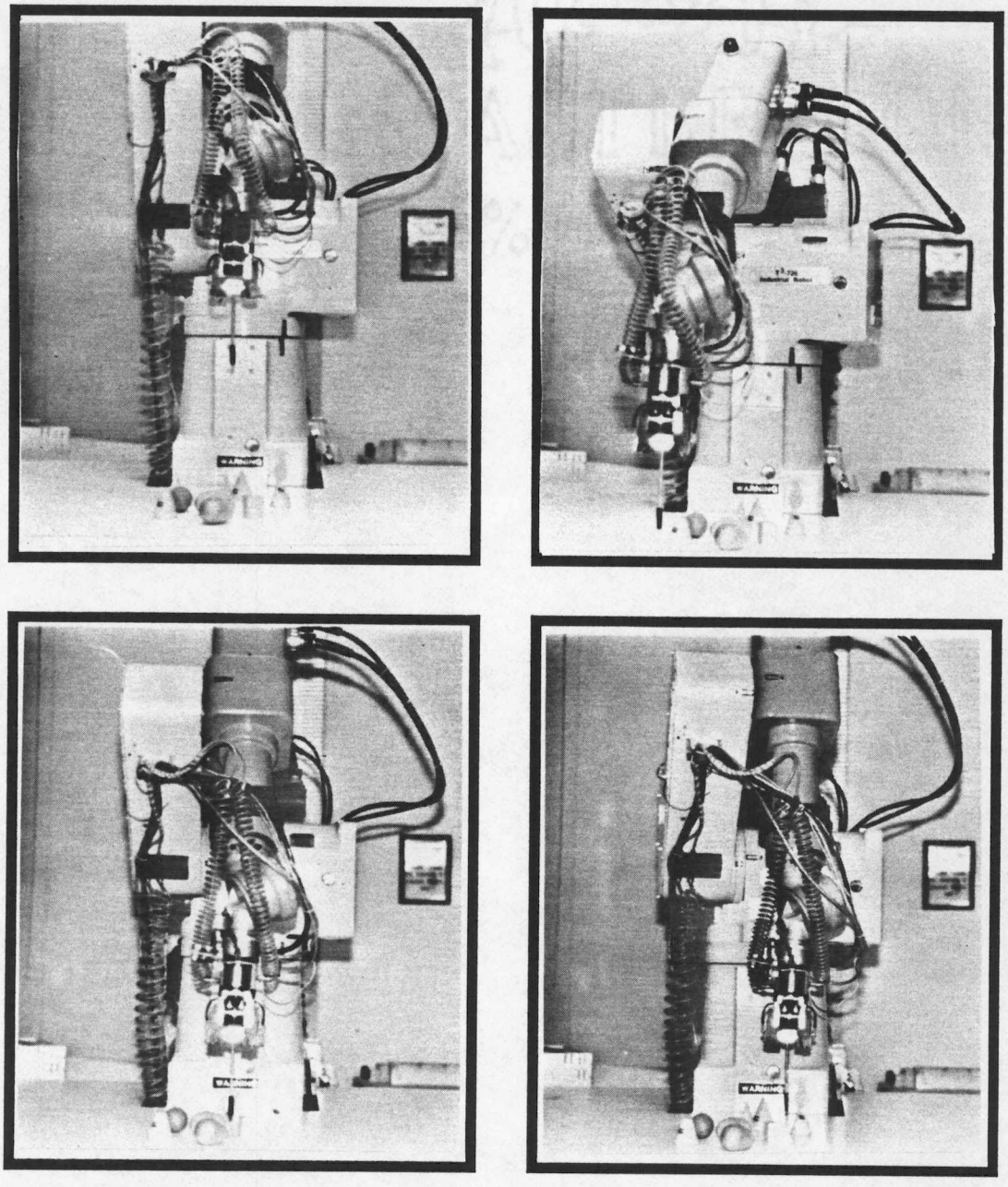


Figure 5.27: The robot acquiring an image of a 2-D workspace [Top Left], and at the start [Top Right], an intermediate [Bottom Left], and the goal [Bottom Right] positions of a 2-D path generated by the algorithm.

CHAPTER 6

CONCLUDING REMARKS AND EXTENSIONS

This thesis addressed the two major issues in path planning, namely, workspace representation, and path generation. A detailed description was given of the path planning algorithm which was developed in this study to generate safe 2-D and 3-D paths for a robot. The efforts made to develop an algorithm which extends the approach of the 2-D path planning to 3-D were discussed. The merits of this path planning approach and its scope of application were also mentioned. A justification was outlined for the choice of octree (quadtree in 2-D) as the data structure for representation and the constraints used in path generation. The path generation required the determination of the distance of a free space to the nearest obstacle block in the workspace representation. A methodology was proposed to compute this distance in 3-D space. This algorithm was implemented, and tested with both synthetic and real images. The results, illustrated through images and timing statistics, were given. An experimental system set up on a robotic workstation to verify this approach was also described. The results indicate the usefulness of this approach for the intended application.

Future research in improving the algorithm may be directed toward enhancing its speed and robustness. The following areas are worth exploring.

The constraint on the orthogonality of the three views to generate the octree

representation from multiple 2-D representations may be unrealistic. However, it is possible to generate an octree representation of a 3-D space from any three non-coplanar views. Such an octree is called a "generalized octree" [28]. Each block associated with a node in the generalized octree is a parallelepiped with its three sides aligned with the three viewing directions. The generalized octree structure allows subsequent refinement of the representation as additional views become available.

The hierarchical nature of the representation has the important advantage that the search can be made *staged*, i.e., plan at a coarser level and subsequently refine the path as needed, thus reducing the planning cost substantially [3]. Small isolated black nodes will fragment the freespace, and give rise to an undesirable increase in the depth of the tree and the number of leaf nodes, consequently increasing the cost of the search. This problem can be dealt with by pruning the tree so that certain gray nodes are made terminal nodes. These gray nodes correspond to a mixture of obstacle nodes and free nodes in the original tree. Path planning now consists of planning a global path at this coarse level and subsequently developing a path inside the regions represented by the gray nodes.

Path planning in the current implementation is achieved by shrinking the robot to a reference point while expanding the obstacles by the size of the robot. The *pseudo-obstacle* method, though simple to implement, is not robust. It blocks paths which otherwise could have been utilized. Therefore, a better method to ensure that the path is collision-free is the generation of the *linear swept volume* of the robot along its path [33]. If this volume does not collide with an obstacle, then

the path is collision-free. One way of representing this *linear swept volume* would be to use another octree. Collision detection consists of traversing in parallel, the two trees representing the workspace representation and the swept volume. The path is not collision-free if for any BLACK node in a tree the corresponding node in the other tree is BLACK or GRAY. A better and more efficient way of representing the swept volume would be to approximate the articulate parts of the robot and the swept volume of the robot by a set of primitive shapes. Since the robot is fully contained in its approximating shape, a free path for the shape is a free path for the robot. Associated with each swept-volume primitive shape is a set of curves within its volume that follow the sweep used to form the shape. Collision detection consists of individually testing the curves within the swept volume to determine if they intersect with an obstacle. A technique called *ray-tracing* [27] can be used to perform this task. This algorithm is time-efficient if the objects in the workspace are represented as octree structures.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] C. E. Thorpe, "FIDO: Vision and Navigation for a Robot Rover," Tech. Rep. CMU-CS-84-168, Dept. of Comput. Sci., Carnegie-Mellon University, Pittsburg, PA, December 1984.

[2] O. Khatib, "Real-time Obstacle Avoidance for Manipulators and Mobile Robots," *Int'l J. Robotic Research*, vol. 5, no. 1, pp. 90–98, Spring 1986.

[3] S. Kambhampati and L. S. Davis, "Multiresolution Path Planning for Mobile Robots," *IEEE J. Robotics and Automation*, vol. RA-2, no. 3, pp. 135–145, December 1986.

[4] E. Wong and K. Fu, "A Hierarchical Orthogonal Space Approach to Three-Dimensional Path Planning," *IEEE J. Robotics and Automation*, vol. RA-2, no. 1, pp. 42–53, March 1986.

[5] J. L. Crowley, "Navigation for an Intelligent Mobile Robot," *IEEE J. Robotics and Automation*, vol. RA-1, no. 1, pp. 31–41, March 1985.

[6] S. V. Rao, S. S. Iyengar, C. C. Jorgensen, and C. R. Weisbin, "Robot Navigation in Unexplored Terrain: Algorithms and Analysis," Tech. Rep. TR-TM-9732, Oak Ridge National Lab., TN, 1986.

[7] N. J. Nilsson, "A Mobile Automation: An Application of Artificial Intelligence Techniques," in *Proc. 2nd Int'l Joint Conf. Artificial Intelligence*, pp. 509–520, May 1969.

[8] T. Lozano-Perez and M. A. Wesley, "An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles," *Comm. ACM*, vol. 22, no.10, pp. 560–570, October 1979.

[9] A. M. Thompson, "The Navigation of the JPL Robot," in *Proc. 5th Int'l Joint Conf. Artificial Intelligence*, pp. 749–757, 1977.

[10] H. Moravec, "Rover Visual Obstacle Avoidance," in *Proc. 7th Int'l Joint Conf. Artificial Intelligence*, pp. 785–790, 1981.

[11] R. A. Brook, "Solving the Find-path Problem by Good Representation of Free Space," *IEEE Trans. Syst., Man, and Cyber.*, vol. SMC-13, no. 3, pp. 190–197, 1983.

[12] G. Giralt, R. Sobek, and R. Chatila, "A Multi-level Planning and Navigation for a Mobile Robot: A First Approach to Hilare," in *Proc. 6th Int'l Conf. Artificial Intelligence*, pp. 335–338, August 1979.

[13] R. Ruff and N. Ahuja, "Path Planning in a Three-Dimensional Environment," in *Proc. 7th Int'l Conf. Patt. Rec.*, pp. 188–191, July 1984.

[14] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of the Minimum Cost Paths," *IEEE Trans. Sys. Sci. Cyber.*, vol. SSC-4, no. 2, pp. 100–107, July 1968.

[15] J. Canny, "A Voronoi Method for Piano-movers Problem," in *IEEE Int'l Conf. Robotics and Automation*, pp. 530–535, March 1985.

[16] O. Khatib and J. F. Le Maitre, "Dynamic Control of Manipulators Operating in a Complex Environment," in *Proc. 3rd CISM-IFTOMM Symp. Theory Practice Robots Manipulators*, pp. 267–282, September 1978.

[17] P. H. Winston, *Artificial Intelligence.* Reading, MA: Addison-Wesley, 1984.

[18] J. Veenstra and N. Ahuja, "Octree generation from Silhouette Views of an Object," in *Proc. IEEE Conf. Comput. Vision and Patt. Rec.*, pp. 524–529, June 1985.

[19] T. Hong and M. O. Shneier, "Incrementally Constructing a Spatial Representation Using a Moving Camera," in *Proc. IEEE Conf. Comput. Vision and Patt. Rec.*, pp. 591–596, June 1985.

[20] D. H. Ballard and C. M. Brown, *Computer Vision.* Englewood Cliffs, NJ: Prentice Hall, 1982.

[21] M. O. Shneier, E. Kent, and P. Mansbach, "Representing Workspace and Model Knowledge for a Robot with Mobile Sensors," in *Proc. 7th Int'l Conf. Patt. Rec.*, pp. 199–202, July 1984.

[22] M. Herman, "Generating Detailed Scene Descriptions from Range Images," in *Proc. IEEE Conf. Robotics and Automation*, pp. 426–431, March 1985.

[23] C. L. Jackins and S. L. Tanimoto, "Oct-trees and Their Use in Representing Three-Dimensional Objects," *Comput. Graph. and Image Processing*, vol. 14, pp. 249–270, 1980.

[24] D. Meagher, "Geometric Modeling Using Octree Encoding," *Comput. Vision, Graph. and Image Processing*, vol. 19, pp. 129–147, 1982.

[25] T. Lozano-Perez, "Spatial Planning: A Configuration Space Approach," *IEEE Trans. Comput.*, vol. C-32, no. 2, pp. 108–120, February 1983.

[26] C. H. Chien and J. K. Aggarwal, "Volume/Surface Octrees for Representation of Three-Dimensional Objects," *Comput. Vision, Graph., and Image Processing*, vol. 36, pp. 100–113, 1986.

[27] A. S. Glassner, "Space Subdivision for Fast Ray Tracing," *Comput. Graphics and Applications*, vol. 4, no. 10, pp. 15–22, October 1984.

[28] C. H. Chien and J. K. Aggarwal, "Identification of 3D Objects from Muliple Silhouttes Using Quadtrees/Octrees," *Comput. Vision, Graph., and Image Processing*, vol. 14, pp. 256–273, 1986.

[29] H. Samet, "Region Representation: Quadtrees from Binary Arrays," *Comput. Graph. and Image Processing*, vol. 13, pp. 88–93, 1980.

[30] H. Samet, "Distance Transform for Images Represented by Quadtrees," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-4, no. 3, pp. 298–303, May 1982.

[31] E. Rich, *Artificial Intelligence.* New York, NY: McGraw Hill, 1983.

[32] N. J. Nilsson, *Principles of Artificial Intelligence.* Palo Alto, CA: Tioga, 1980.

[33] M. Herman, "Fast Three-Dimensional, Collision-Free Motion Planning," in *IEEE Int'l Conf. Robotics and Automation*, pp. 1056–1063, April 1986.

[34] K. S. Fu, R. C. Gonzalez, and C. S. G. Lee, *Robotics: Control, Sensing, Vision, and Intelligence.* New York, NY: McGraw Hill, 1986.

VITA

Hyder Ali was born in Madras, INDIA, on April 6, 1962. He joined the Anna University at Madras to pursue a Bachelor's degree in Electronics and Communication Engineering in October 1980. He graduated with honors in March 1985. He entered the University of Tennessee at Knoxville, Tennessee, in September 1985. He was a teaching and a research assistant from October 1986 until his graduation. He received his Master's degree in Electrical and Computer Engineering in December 1987.