1991

Environment exploration and path planning algorithms for mobile robot navigation using sonar

Alexander Zelinsky
University of Wollongong

Recommended Citation

Zelinsky, Alexander, Environment exploration and path planning algorithms for mobile robot navigation using sonar, Doctor of Philosophy thesis, Department of Computer Science, University of Wollongong, 1991. http://ro.uow.edu.au/theses/1295

Environment Exploration and Path Planning Algorithms for Mobile Robot Navigation using Sonar

A thesis submitted in fulfilment of the
requirements for the award of the degree of

Doctor of Philosophy
(Computer Science)

from

THE UNIVERSITY OF WOLLONGONG

by

Alexander Zelinsky, B.Math(Hons) W'gong

Department of Computer Science
Department of Electrical and Computer Engineering

1991

I hereby declare that I am the sole author of this thesis. I also declare that the material presented within is my own work, except where duly acknowledged, and that I am not aware of any similar work either prior to this thesis or currently being pursued.

A. Zelinsky

Abstract

The work reported in this thesis is motivated by the need to construct a navigation system for mobile robots which can operate in unknown and partially known environments, and which has the capability to progressively learn an environment. A new environment mapping procedure is described that constructs high resolution maps of an environment using ultrasonic range sensing. The ultrasonic range maps are converted into quadtrees. Quadtrees are used by the navigation system as the data structure that models the environment.

This thesis presents a new algorithm for a mobile robot to explore an unknown environment using the quadtree data structure and the distance transform path planning methodology. Past approaches to robot path planning have concentrated on finding the shortest path to a goal. A path planner should also support finding a variety of other kinds of paths to a goal. For example a path planner could support finding "conservative", "adventurous" and "safe" paths. Conservative paths favour known areas, while adventurous paths avoid known areas. Safe paths keep a safe distance from obstacles, while simultaneously keeping the path length to the goal as short as possible. It is shown that the new mobile robot exploration algorithm can generate a wide variety of path planning behaviours by a novel use of distance transforms.

Much of the research effort into path planning for mobile robots has concentrated on the problem of finding paths by translation of the robot body only. The problem of finding paths which require the rotation of the robot body have been largely ignored. This thesis presents an new algorithm for path planning with three degrees of freedom which is based upon an extension to the "safe" path planning behaviour.

Finally it is shown that the new algorithms that have been presented are computationally efficient, and have desirable features that are absent from other path planning algorithms.

Acknowledgements

My decision to undertake a Ph.D on a part time basis has turned out to be a 6 year test of endurance for my family and friends. I must thank all the people who have helped me along the way. First, I must thank my supervisors Dr. Phillip McKerrow and Prof. Chris Cook for their assistance and guidance. I am grateful for their encouragement to persevere with the work in spite of difficulties and frustrations in tasks of this nature. I must also thank Dr. Greg Doherty who waded through this document and made much appreciated suggestions.

The sonar mapping work described in this thesis was carried out at the Intelligent Robotics Research Centre, Monash University, Australia where I spent 6 months of study leave. I thank Prof. Ray Jarvis, Director of the Intelligent Robotics Research Centre, for providing me with the opportunity to work at the Centre. I am grateful for the first class mobile robot, the computing power and the excellent support staff that were provided by the Centre during the course of the work. I extend my special thanks to Jullian Byrne and Kemal Ajay for all their assistance in helping to solve the software and hardware problems that cropped up from time to time.

I must also thank my students Glen Conners, Gerrard Drury, Graham Heathcote, Harald Kolodziej, Grant Moule and Graham Shaw who ported my path planning software from a crude VT100 display to run on the user friendly Macintosh. This enabled us to find bugs in my software and algorithms and made it possible to use screen dumps from the Macintosh in the many diagrams which illustrate this thesis. I am grateful for their willingness to question what I said and wrote; through their inquisitiveness I was able to correct and refine my work.

On a personal note I would like to thank my parents, particularly my mother, for encouraging me to pursue my studies. I am also very grateful to my friends George Perin and Irene Maya Romanova, who cared for my children, thus giving me the opportunity to

work long hours at the research laboratory. I am also thankful to George and Irene for all their nurturing personal support. Without their support I could not have completed this work. I am blessed to have friends like this. I must also thank my sons Misha and Tima who have been willing to put up with my coming and going at strange hours, and have never once forgotten who I am.

Publications from this Thesis

A. Zelinsky and C.D. Cook, "Environment Mapping for Mobile Robots", Proceedings of the 2nd National Australian Robot Conference, Perth, May 1986.

A. Zelinsky, "Robot Navigation with Learning", Australian Computer Journal, Vol. 20, No.2, May 1988.

A. Zelinsky, "Navigation with Safety", Proceedings of the 1989 IEEE/RSJ International Workshop on Intelligent Robots and Systems, Tsukuba Japan, September 1989.

A. Zelinsky, "Robot Path Planning with Safety", Proceedings of the 13th Australian Computer Science Conference, Melbourne, February 1990.

A. Zelinsky, "Path Planning for Mobile Robots with 3 DOF", Proceedings of the 14th Australian Computer Science Conference, Sydney, February 1991.

A. Zelinsky, "Environment Modelling with Autonomous Mobile Robots using Sonar", Second International Advanced Robotics Workshop on Sensor Fusion and Environment Modelling, Oxford UK, September 1991.

A. Zelinsky, "Mobile Robot Map Making using Sonar", Journal of Robotic Systems, *to appear* October 1991.

A. Zelinsky, "A Mobile Robot Map Exploration Algorithm", IEEE Journal of Robotics and Automation, *accepted subject to revision* .

Table of Contents

Chapter 1
The Overview

Research into mobile robotics is driven by the goal of building machines with the capabilities of humans. The aim of this research is to create machines which will replace humans in performing tasks that we find tedious, dirty or hazardous. The "intelligence" of a robot is a measure of the sophistication of the robot, and the type of environment that it can exist in. Clearly a parts delivery robot operating on the factory floor, which navigates by following a buried wire, requires very little machine intelligence, compared with a robot whose task is to guide a motor vehicle through traffic at high speed. To achieve the latter task, the robot must simultaneously address the problems of "perception", "mobility" and "intelligence" in a changing environment, and therefore this robot can be regarded as an intelligent robot. The challenge of mobile robotics research is to develop intelligent technology that does not operate only in a limited domain, but in the "real world".

The purpose of an autonomous mobile robot is to perform tasks for a user. For the correct and efficient performance of user tasks, a mobile robot must have the capability to interact with the environment in which it is operating. An essential interaction capability is navigation. Navigating a mobile robot in a known environment requires little or no interaction with the environment. However, a robot may have to operate in partially known or completely unknown environments. In such circumstances the robot must learn and understand the structure of the environment. The robot must acquire and handle information about the existence and location of objects and areas of unoccupied space. Building a coherent world model within the limitations of current perception sensing technology is a complex task. This involves the extraction of range information from the environment, taking into account noise and the inaccuracy of sensor information. This introduces uncertainty into the environment map and makes the task of navigating a robot more challenging.

The work reported in this thesis is motivated by the need to construct a navigation system for mobile robots which can operate in unknown and partially known environments, and which has the capability to progressively learn an environment. In this thesis a navigation system will be described that has the capability of sensing the environment, and incorporating the freshly sensed data into an internal model of the environment. To efficiently model an environment, it is necessary to develop a data structure which can support the inclusion of freshly sensed data, and which supports path planning. In summary, the contribution of the research reported in this thesis furthers the development of data structures and algorithms for:

* environment mapping with sonar range data.
* path planning for mobile robots.
* path planning behaviours for mobile robots.
* path planning for mobile robots with 3 degrees of freedom.

The detailed description of the research goals of this thesis are provided in the remaining paragraphs of this chapter.

Environment Mapping

The first goal of this research was to build a practical and reliable navigation system which was capable of operating in real-time or near real-time. To achieve this aim ultrasonic range sensing (referred to as "sonar" for short) was chosen as the perception medium ahead of computer vision. Using computer vision as the perception medium requires the storage and quick processing of large quantities of data. Real-time computer vision requires expensive dedicated processing hardware, while sonar has minimal storage and processing requirements. Another reason for the choice of sonar as the perception medium was to allow concentration on the research issues of robot navigation and mapping with range data, rather than the issue of extracting range from computer vision.

One of the drawbacks of sonar is the noise and uncertainty associated with sonar range readings. In this thesis it is shown how clean and reliable data can be obtained from a noisy sonar signal and how this data can be integrated into an environment map for use in a robot navigation system. This work is described in Chapter 2.

Path Planning

A review of past mobile robot path planning research is presented in Chapter 3. Past research into path planning for mobile robots has concentrated on the problem of finding the shortest path in a known environment. However for a mobile robot operating in a partially known or a completely unknown environment, it is difficult to design a path planner which can effectively handle the unknown regions of the environment. For a path planner to be effective, the navigation system must have a mechanism to use the freshly sensed environment data. The second goal of this research was to develop a path planning algorithm which allows a mobile robot to operate in partially known and completely unknown environments, and which can handle sensor data describing the structure of the unknown parts of an environment. Such a path planning algorithm is presented in Chapter 4. The path planning results are presented using the sonar data results given in Chapter 2.

Path Planning Behaviours

A robot path planning system should not only find "optimum" paths i.e. the shortest paths to goals, but the system should also be able to generate "conservative" and "adventurous" paths. Conservative paths are paths which favour known areas, so the robot will attempt to travel in regions which it knows about. Adventurous paths on the other hand are paths which avoid known areas, causing the robot to operate in regions which have not previously been visited. A robot navigation system should also possess a "learn all" behaviour, which allows the robot to systematically map all the unknown regions of an environment. A useful path planning behaviour that a robot can possess once an environment is completely known, is a behaviour which allows the robot to efficiently "visit all" the free space in an environment. Such a path planning behaviour is relevant for flooring cleaning and security surveillance robots.

The biggest challenge in building a competent path planner is to reconcile the conflicting requirements of finding the "best" path from some start location to a goal location. The best path is not necessarily the shortest path to the goal. A mobile robot should keep the length of the path as short as possible, while simultaneously keeping a safe distance from obstacles and avoiding unknown regions of the environment.

The third goal of my thesis was to construct a robot path planner which is capable of exihibiting different types of path planning behaviours. In Chapter 5 the concept of behaviours for mobile robot path planning is introduced. This chapter shows how the path planner described in Chapter 4 can be extended to induce the robot navigation control system to exhibit the path planning behaviours of finding "best", "adventurous", "conservative", "learn all" and "visit all" paths. Experimental results using the sonar data collected in Chapter 2 are reported.

In Chapter 6 a comparison of the new robot path planning algorithms with another similar class of path planning algorithms is presented.

Path Planning with 3 Degrees of Freedom

Much of the research effort into path planning for mobile robots has concentrated on the problem of finding paths from a start position to a goal position by translation of the robot body only. The problem of finding paths which require the rotation of the robot body have been largely ignored. The fourth and last goal of this research was to develop a path planning algorithm for mobile robots which have 3 degrees of freedom (DOF) of movement. In Chapter 7 of this thesis an algorithm is presented which is based upon the extension of ideas presented in earlier chapters. It is shown that this new algorithm is computationally superior to other 3 DOF path planners, and has desirable path planning features that are absent from other 3 DOF path planners.

Finally, in Chapter 8 the conclusions that have been drawn from this research are presented, and mobile robot navigation problems which deserve further research attention are discussed.

Chapter 2
Environment Mapping using Sonar

2.1 Introduction

This chapter describes a new method of producing high resolution maps of an indoor environment with an autonomous mobile robot equipped with sonar range finding sensors, which is based upon investigating obstacles in the near vicinity of a mobile robot. The mobile robot examines the straight line segments extracted from the sonar range data describing obstacles near the robot. The mobile robot then moves parallel to the straight line sonar segments, in close proximity to the obstacles, continually applying the "sonar mapping test". The sonar mapping test exploits the physical constraints of sonar data, and eliminates noisy data. This test determines whether or not a sonar line segment is a true obstacle edge or a false reflection. Low resolution sonar sensors can be used with the described method. This environment mapping procedure can be integrated with most path planning algorithms and different types of range finding sensors.

The experimentation in this work was carried out on a Model T Denning Mobile Robot (Figure 2.1), equipped with a twenty four element Polaroid Corp. Ultrasonic Range Finder sensor array [Polaroid 82]. The control system of the Model T uses three microprocessors. A supervisory 68008 microcomputer communicates with the outside world, the drive and steer system, and the sonar system. The robot communicates with the outside world using a serial link to a VAX minicomputer. The three wheel synchronous drive and steer system in the Model T is controlled by a Z80 microcomputer. The sonar array built by Denning Mobile Robotics is arranged in a ring configuration, with each of the sonar sensors spaced 15 degrees apart. The ring is controlled by a Z80 microprocessor which selects sensors, activates the ranging and returns the corresponding range data. The sonar data is sent to the supervisory 68008 microcomputer, which passes the data onto the VAX minicomputer. The VAX interprets the sonar data and performs the navigation functions of mapping and path planning.

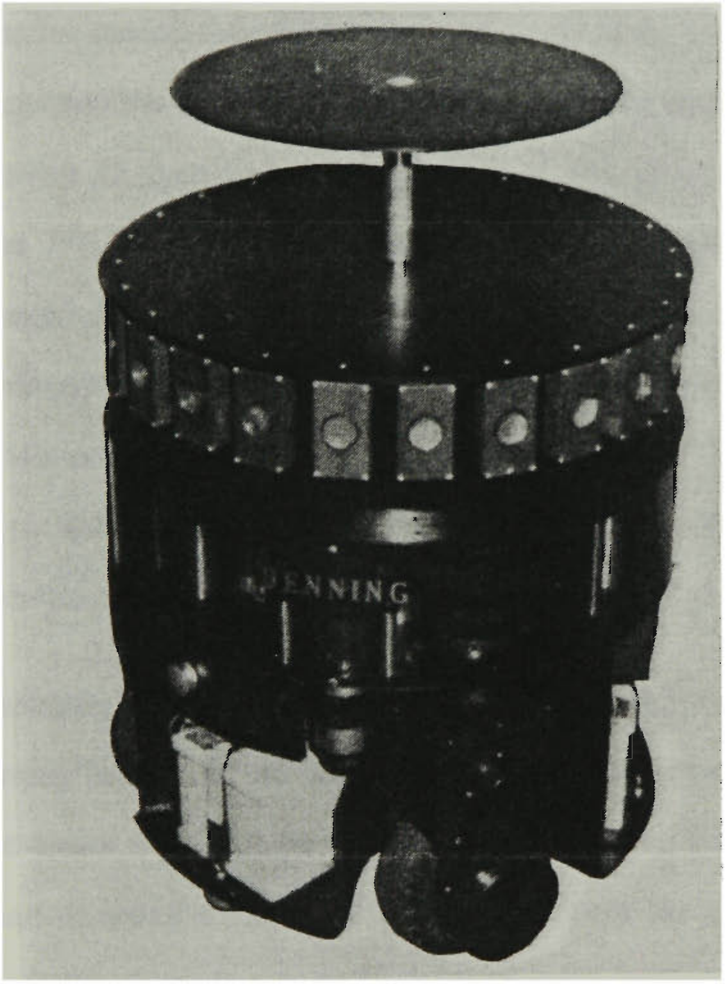


Figure 2.1

Denning Corp. Model T Mobile Robot Research Vehicle.

The remainder of this chapter is organised in the following manner. Section 2.2 reviews the research which has been undertaken in the past into environment mapping for mobile robots. Section 2.3 reviews the reported uses of sonar in mobile robotics, and the problems associated with sonar sensing. In Section 2.4 a new approach to environment mapping using sonar is presented. Section 2.5 presents the experimental results obtained from an implementation of the new approach. Finally in Section 2.6 the conclusions which have been drawn from this research are presented.

2.2 Environment Mapping

Research efforts into environment mapping with mobile robots follow a variety of approaches. These can be broadly classified into two groupings; "adaptive" models and "rigid" models. Adaptive models reflect the nature and clutter of the environment. Typical adaptive models represent the environment as a network of free space regions [Brooks 84, Chatila 82, Crowley 85, Iyengar *et. al.* 86, Rao *et. al.* 86], or as a graph of obstacle vertices [Thompson 77]. Environment mapping methods which use adaptive models require accurate sensor information. Rigid models impose a structure, such as a grid, onto the environment without any regard to the nature and clutter of the environment [Elfes 87, Jarvis *et. al.* 86, Moravec 80, Thorpe 84]. Adaptive model environment mapping methods offer elegant and efficient solutions, but in practice are difficult to implement. The converse can be said of rigid model environment mapping methods.

Environment mapping by a mobile robot can be accomplished either by operating in "mapping" or "learning" modes. When a robot is operating in mapping mode [Crowley 85] it traverses the entire environment in a systematic manner, while scanning with onboard sensors and updating a map. The map is then used for all subsequent path planning exercises. Difficulties arise with this method if the environment is allowed to alter after the mapping has been completed. Work on environment mapping using the mapping mode is reported in Chapter 5.

The other mode of learning is to sense the environment while executing paths which have been generated by a path planner. As the robot encounters obstacles en route to a goal, the mapping process updates the model of the environment, and the path planner plans a new path to the goal which avoids the obstacles [Elfes 87, Iyengar *et. al.* 86, Jarvis *et. al.* 86, Rao *et. al.* 86, Thorpe 84]. The paths generated by these methods will initially be negotiable paths from the start to the goal location. The lengths of these paths are not optimum, but as knowledge of the environment increases better paths are generated until eventually global optimality is attained. Research into using the learning mode to generate a map of the environment is described in Chapter 4.

The environment mapping methods presented by [Iyengar *et. al.* 86, Rao *et. al.* 86] have problems since they uses heuristics to plan local paths around obstacles. These methods assume that a robot can always recognise the line of sight distances to obstacles, and that obstacle edges can always be precisely detected. Such restrictions make it difficult to implement this mapping algorithm using current sensor technology. Since line of sight sensing cannot be guaranteed, obstacles in an environment can be arranged in a configuration which will cause a heuristic path planner to fail [Cahn *et. al.* 75, Chattergy 85].

Since one of the goals of this research was to produce a practical and reliable mobile robot navigation system, a rigid model using grids was selected as the representation of the environment map. However this rigid model can be easily transformed into an adaptive model for use in path planning activities, as shown later in Chapter 4.

The environment mapping method presented in this chapter can used to operate in both "mapping" and "learning" modes to obtain information about the environment. This map making method assumes that the location of the robot is known at all times. In the implementation of this mapping method, dead-reckoning was used satisfactorily. Finally this environment mapping method does not have the short comings of using heuristics to plan paths, nor does it assume that the robot has the ability to measure the exact shape of obstacles and the line of sight distances to obstacles.

2.3 Sonar and Mobile Robots

The application of sonar sensors in robotics is increasingly attracting interest and research. This is partly due to the low cost of the sensor and the ease with which the sonar data can be processed to directly provide range information. Sonar has been used by mobile robots for navigation [Brooks 86, Chatila 82, Chatila *et. al.* 85, Chattergy 85, Crowley 85], determination of position [Crowley 85, Drumheller 87, Durrant-Whyte *et. al.* 89, Miller 84, Miller 85] and mapping purposes [Crowley 85, Elfes 87, Flynn 85].

The Hilare mobile robot project [Bauzil *et. al.* 81, Chatila 82, Chatila *et. al.* 85, Chattergy 85] uses sonar sensors as proximity indicators for close up obstacle detection and for parallel wall following. [Chattergy 85] uses sonar with a heuristic navigation system to detect and follow obstacle boundaries. Both these approaches used sonar for collision avoidance with stationary obstacles only. The MIT AI Lab mobile robot [Brooks 86] uses sonar for collision avoidance with both stationary and moving obstacles.

Sonar has been used to determine the position of a mobile robot in an environment [Durrant-Whyte *et. al.* 89, Drumheller 87, Miller 84, 85]. [Miller 84, 85] assumed that the environment was known, and that an accurate map of the environment was available. The map was searched to determine the location of the robot. The position of the robot found in the map search had to confirm the set of sonar readings collected from the environment. This approach had limitations since it did not take into account the uncertainty and noise of sonar data.

[Drumheller 87] used a similar approach to Miller. This approach also required an accurate map of the environment, but it could handle noisy data. The method was implemented using the Polaroid sonar sensor. This approach coped with the uncertainties of sonar by modelling the sonar sensor and data to account for false reflection and beam spread errors.

[Durrant-Whyte *et. al.* 89] used a different approach to position estimation. This approach used an extended Kalman filter to integrate sonar readings into a known (precisely taught) environment map, and to derive an estimate of the mobile robot location.

False reflections can cause extremely large errors in sonar readings. False reflections are caused by the long wavelength of sound. A sonar beam aimed at a target object surface may not reflect an echo directly to the sensor. Instead an echo may or may not be detected when the sonar beam bounces off some objects other than the target object. Hence the sonar sensor measures a distance to a target which is much longer than the actual distance to the target. False reflections of a sonar beam occur whenever the angle of incidence is greater than the critical angle of reflection, causing the beam to be reflected away from the sensor, and giving the effect of the sonar beam penetrating the obstacle. Refer to Figure 2.2 for an illustration of false reflection of a sonar beam. False reflections occur quite often, and for this reason it is almost impossible to obtain a reasonably accurate "sonar profile" of the environment surrounding the robot.

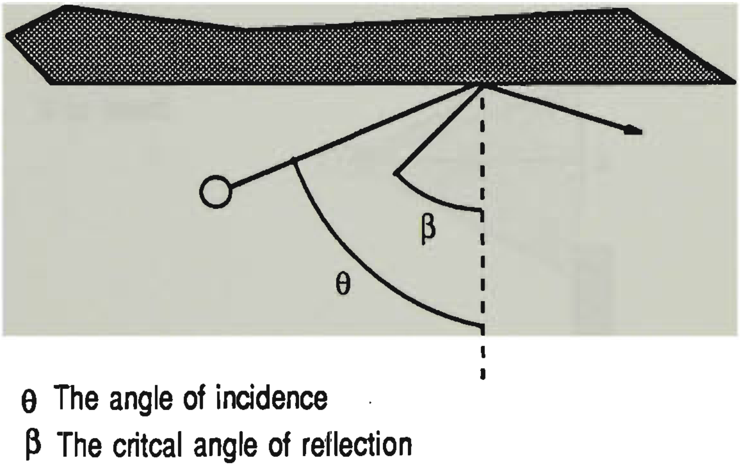


Figure 2.2

The problem of false reflections of a sonar beam.

Sonar beam spread creates a number of problems. Objects can be perceived to be much wider than they really are. This effect is exacerbated with larger distances between sensor and object. Similarly openings between obstacles, such as open doors, may be perceived to be closed. Beam spread can also cause ranging errors. A range measure is not necessarily the distance in the direction the sensor is facing, since the width of the beam may cause an echo from one edge to be returned before the echo from the sensor direction. These ranging errors have the effect of producing a blurred image of the surroundings, particularly corners. Refer to Figure 2.3 for an illustration summary of beam spread problems.

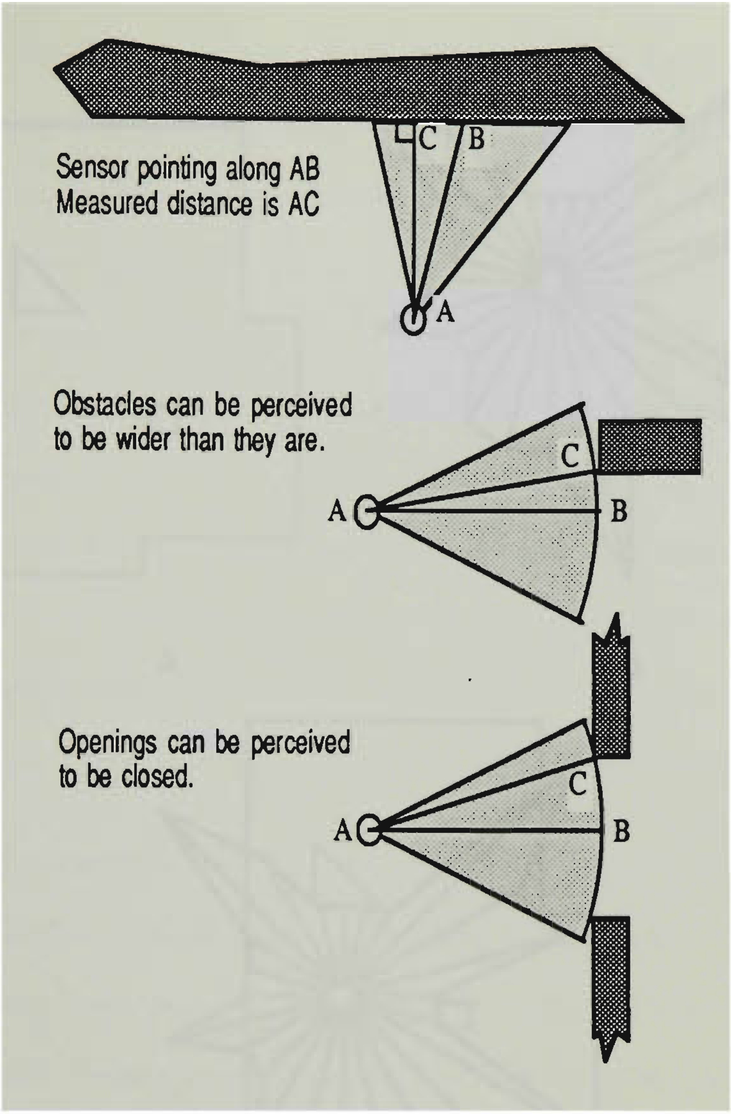

C
B
Sensor pointing along AB
Measured distance is AC
A
Obstacles can be perceived
to be wider than they are.
C
A
B
Openings can be perceived
to be closed.
C
A
B

Figure 2.3

The problem of Sonar beam spread.

Due to false reflections and beam spread a sonar profile may bear little resemblance to the actual environment, as shown in the example in Figure 2.4. Figure 2.4 (A) shows the plan of an indoor environment, which contains two obstacles. Figure 2.4 (B) shows a sonar profile obtained of the environment shown in Figure 2.4 (A). It is obvious that there is little resemblance between the actual environment and the sonar profile. In fact it is difficult to say from what position in the environment the sonar profile was obtained. Figure 2.4 (C) shows where the sonar scan was done in the environment.

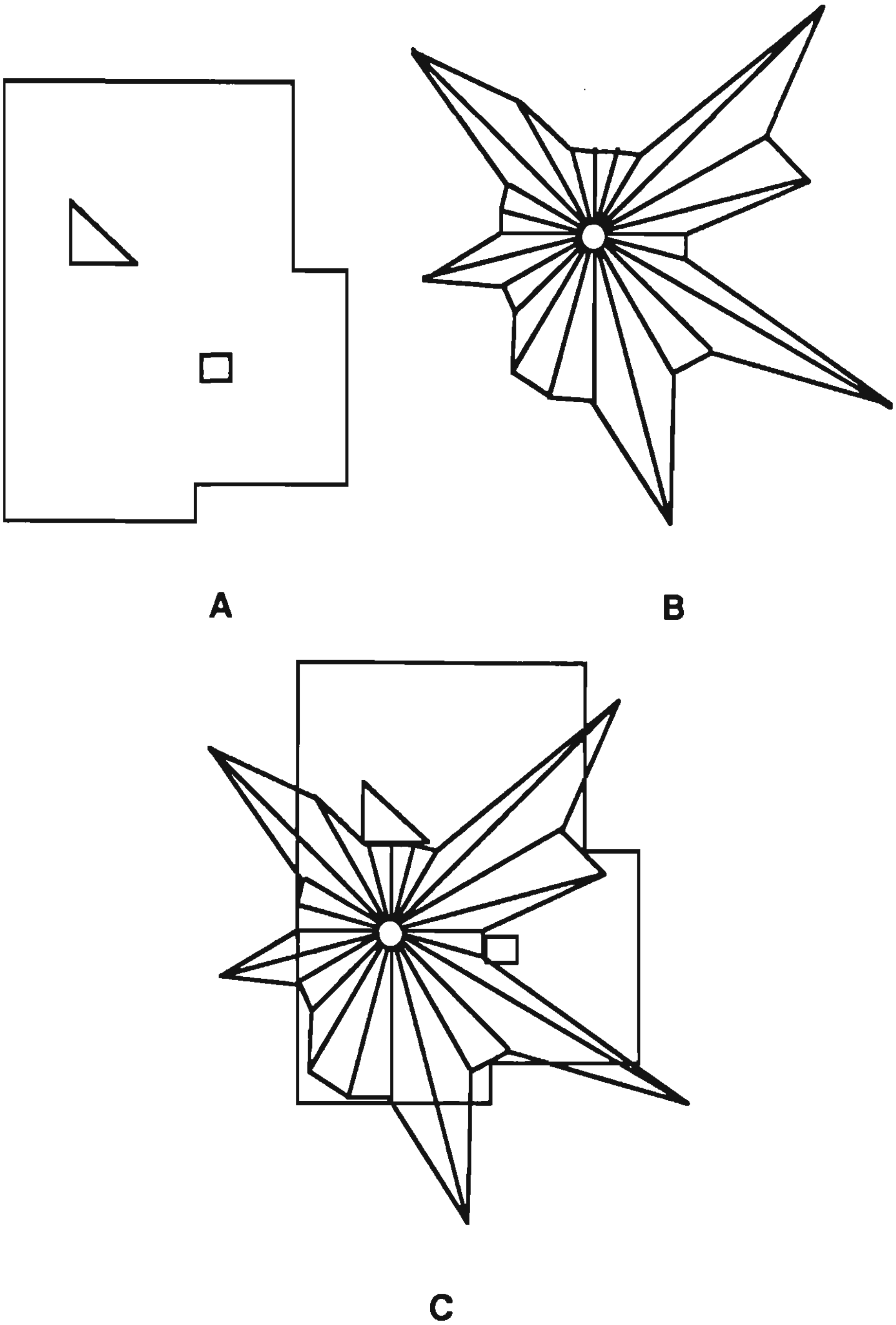


Figure 2.4

The combination of false reflections and sonar beam spread causes problems.

The sonar beam spread problem has been countered in several different ways. [Elfes 87] treats sonar beam spread with a probabilistic approach. Elfes regards a sonar reading as an assertion about two 3D spaces, one that is "probably empty" and another that is "somewhere occupied". The definition of the probability density functions is based on beam geometry and the spatial sensitivity pattern of the sensor. The functions are parameterised by the spread of the beam and the range of the sample. Twenty four sonar sensors are mounted in a ring around the robot. The probability density functions derived from the range readings for each sensor are combined and projected onto a two dimensional grid. In this probability map, the value stored in each grid cell indicates if the cell is empty, occupied or unknown. By combining information from many readings as the robot moves through an environment, areas known to be empty or occupied are expanded, and the uncertainties associated with the region are decreased. Consequently the shape and location of obstacles in the environment becomes known with increasing accuracy. This approach tackles the problems associated with sonar sensing by building the environment map with multiple sonar views taken from different locations. However it does not provide a complete solution to false reflections, since no mechanism is provided to cancel the effect on the probability functions of the false reflection sonar readings that were projected onto the probability map at an earlier stage.

[Crowley 85] countered the beam spread problem by narrowing the width of the beam. He generated a narrower beam by focusing a wide sonar beam with a parabolic horn. The narrowed beam was rotated by a stepper motor and sonar readings were taken at regular intervals. From the sonar readings, a line based model of the surrounding environment was constructed using recursive line fitting. The newly constructed local model was then matched with a map which had either been supplied to the robot or learnt by the robot on previous mapping experiments, to determine the robot's location and to update the map if necessary . This approach does not take into account the uncertainty and noise that false sonar reflections can introduce into a line based environment model.

[Flynn 85] attempted to remove the uncertainty and noisiness of sonar data, by sensor fusion. Sonar and infrared sensors were used together, each compensating for the deficiencies in the other. Data from both sensors was fused to generate a more accurate representation than could be achieved by either sensor alone. This method had limited success in dealing with beam spread and false reflection problems, because of the deficiencies of the infrared sensor. However sensor fusion of sonar with infrared was able to detect openings, such as doorways, that would otherwise go undetected if only sonar sensing was used. Flynn gives a treatment of sonar error due to atmospheric effects such as the change in the speed of sound caused by temperature and humidity changes. The timing circuitry of the Polaroid sonar sensor is identified by Flynn as another source of range precision error.

[Drumheller 87] modelled sonar beam spread by assuming that the location of an end point of any sonar reading may be in error by as much as E, where E was a constant that bounded the unpredictable beam spread errors. The value of E was determined experimentally. Drumheller introduced a new concept called the "sonar barrier test". The sonar barrier test was used to eliminate the noise introduced to the sonar profile by false reflections. The sonar barrier test checked that the sonar profile for a proposed location of the robot was consistent with the fact that sonar beams do not penetrate known solid objects. The sonar barrier test was very effective at eliminating localisations of the robot which do verify the sonar profile, but are in fact incorrect. Drumheller showed that false reflections of sonar beams could be handled effectively by the sonar barrier test.

2.4 The New Approach

In summary, past research has shown that to devise an effective environment mapping procedure using sonar, the procedure must handle the sonar sensing problems of false reflections and beam spread. The approaches described in the previous section had problems dealing with both the sonar sensing problems. In this section a new approach to environment mapping using sonar is presented. This approach handles the beam spread problem by sampling data describing obstacles at a closer range. The false reflections problem is dealt with by applying the sonar barrier test. The sonar barrier test is used in a context that is different from the original intentions of its designer [Drumheller 87]. Instead of determining a robot localisation based on the fact that a sonar profile cannot penetrate known solid objects, the test is used to determine the shape of unknown objects given the fact that a sonar profile may appear to pass through objects. To reflect the new context of use of the sonar barrier test, the test will be referred to as the "sonar mapping test"

Producing an accurate map of an indoor environment from a sonar profile sampled in one location is virtually impossible, as shown in Figure 2.4. However the sonar profile does indicate the approximate location and size of obstacles and probable areas of free space. The task of the environment mapping process is to determine whether or not the data contained in a particular sonar profile is correct or not. Testing the correctness of the sonar profile is done by moving the robot into closer proximity to the obstacles identified in the sonar profile, and applying the sonar mapping test. The conclusions drawn from investigating and testing the sonar profile data are incorporated into the environmental map.

Most path planners approximate the robot with a cylinder and then shrink the cylinder to a point and expand all the objects in the environment by the radius of the cylinder. This strategy is useful in known environments. But in unknown environments this requires unnecessary extra processing, and important information is discarded, namely the fact that the volume of space that the robot occupies is free space. This can provide considerable additional knowledge about the environment. Also the volumes of the paths swept by the robot are definitely free space in a static environment and many of them are likely to be free space in a dynamic environment. The work done by [Flynn 85] with sensor fusion showed that sonar data fused with infrared data produced improved maps. The new mapping algorithm presented in this section fuses the free space volumes swept by a moving robot with the range data collected by sonar sensors to build a detailed map of the environment.

The remainder of this section is organised in the following manner. Section 2.4.1 describes how the raw sonar data which has been collected is preprocessed to remove incorrect readings. Section 2.4.2 explains how lines are fitted to the clean sonar data, and how the most reliable segments are extracted from the newly constructed lines. In Section 2.4.3 a description is presented of how the sonar mapping test is used to validate the sonar readings which were collected in the first sonar scan of the environment. Finally in

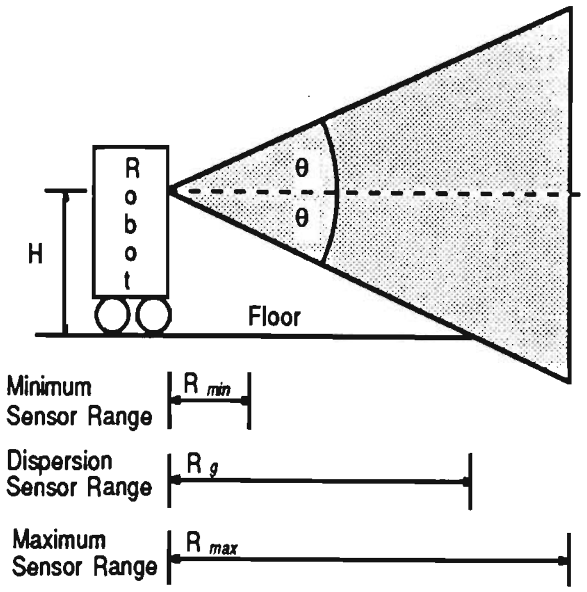
Section 2.4.4 an algorithm is presented which describes how the immediate environment surrounding a mobile robot can be systematically mapped.

2.4.1 Sonar Preprocessing

To obtain reliable sonar range data the sensing system preprocesses all the incoming sonar data to remove the incorrect range readings. To achieve the aim of clean and reliable sonar range data the preprocessor executes the following steps:

***Calibrating*:** Variations in the speed of sound caused by changes in temperature and humidity give inaccurate sonar readings. Such errors need to be compensated. Compensation can be achieved by equipping the mobile robot with onboard sensors for temperature and humidity, allowing the errors to be corrected onboard the robot before the sonar data is used for further processing. This is the approach used by [Flynn 85]. Another way to compensate for the effect of temperature and humidity changes is to calibrate the sonar sensors. This is done by obtaining the sonar range measurement to a target at a known distance and normalising the measured distance with respect to the known distance. This calibration constant is used to compensate all subsequent sonar range data readings for the effects of atmospheric changes. Since the work described in this thesis was carried out in an air conditioned indoor environment, where the fluctuations in temperature and humidity are small, the calibration approach was used.

***Thresholding*:** As shown in Figure 2.5. range data which falls outside the distance interval $[R_{min},R_{u}]$ is discarded, where R_{min} is the minimum sensor range, and R_u is the the maximum useful range. The maximum useful range R_u is defined as the $\min(R_g,R_{max})$, where R_{max} is the maximum sensor range, and R_g is defined as $H/\mathrm{Sin}\theta$. H is the height the sonar sensors are mounted above the floor, and 2θ is the angular dispersion of the sonar beam. Using range thresholding removes all errors caused by faulty sensors, and some errors caused by false reflections. Sensors which are faulty return range measures which are lower than the threshold range, while false reflections will result in range measures which are higher than the threshold range.



The range interval of the sonar sensor is:

$[R_{min}, \min(R_g, R_{max})]$

where $R_g = H / \sin\theta$

θ = The angular dispersion of the sonar beam

H = The height of the sonar sensor above the foor

Figure 2.5

The useful range interval of a sonar sensor

***Averaging*:** A set of C_n range readings from the same sonar sensor, sampled from the same robot position are usually dispersed. The dispersion of the data is due to false reflections and the varying sensitivity of sonar transducers. It was found experimentally that the sonar range data may be clustered around two different mean values, rather than just one mean value. This phenomenon occurred when a sensor was directed at two obstacles which were placed at staggered distances. The sensor detected the closest obstacle most of the time. However sometimes a false reflection occurred on the sonar beam which was directed at the closest obstacle. This resulted in the obstacle which was furthest away from the sensor being detected. To reduce the dispersion of the sonar data, an analysis of the data is done to identify the clustered sets of range readings. A cluster is considered to be a contiguous set of range readings which accounts for a significant proportion C_p of the collected sonar range readings. The value of C_p is determined experimentally. This is discussed in the Results section. The mapping algorithm discussed here requires knowledge of the closest obstacle to the robot. To accomplish this task, the cluster of sonar range readings with the shortest distance measures is averaged. This averaged value is used in all subsequent processing of the mapping algorithm. Refer to Figure 2.6 for an illustration of averaging.

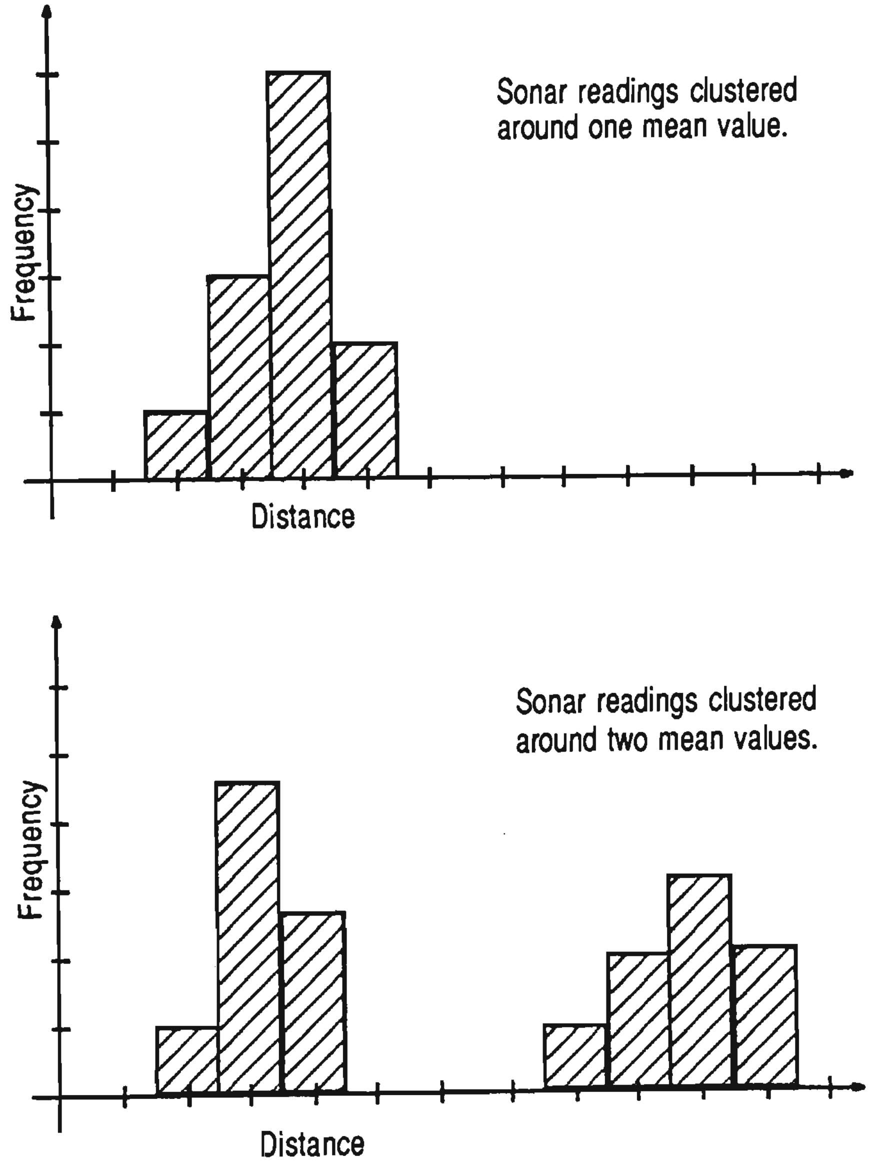


Figure 2.6

Sonar data is not always clustered around one mean value.

2.4.2 Line Fitting and Extracting Sonar Edge Segments

The preprocessed sonar readings obtained from the sonar sensors which surround the robot form a sonar profile of the environment. This sonar profile can be regarded as a robot centred polar coordinate map, since the sonar data represents depth readings from sensors positioned at known angles. The polar coordinate sonar profile is converted into a cartesian world coordinate profile. The cartesian coordinate sonar profile is passed to a recursive line fitting procedure, which fits straight lines to the data. Recursive line fitting of sonar data is not new and has previously been done by [Crowley 85] and [Drumheller 87] using the algorithm described in [Duda *et. al.* 73]. The algorithm is illustrated in Figure 2.7. A straight line is initially computed between the two end points of the collection of sonar range points. The sonar range points in the collection are tested to determine the point where the perpendicular distance to the approximating straight line is greatest. If this largest perpendicular distance is below a tolerance T, then the computed line is accepted as representing the collection of sonar range points. Otherwise the collection of points is divided into two collections at the point where the perpendicular distance was greatest. The line fitting procedure is then invoked recursively for each of the two groups.

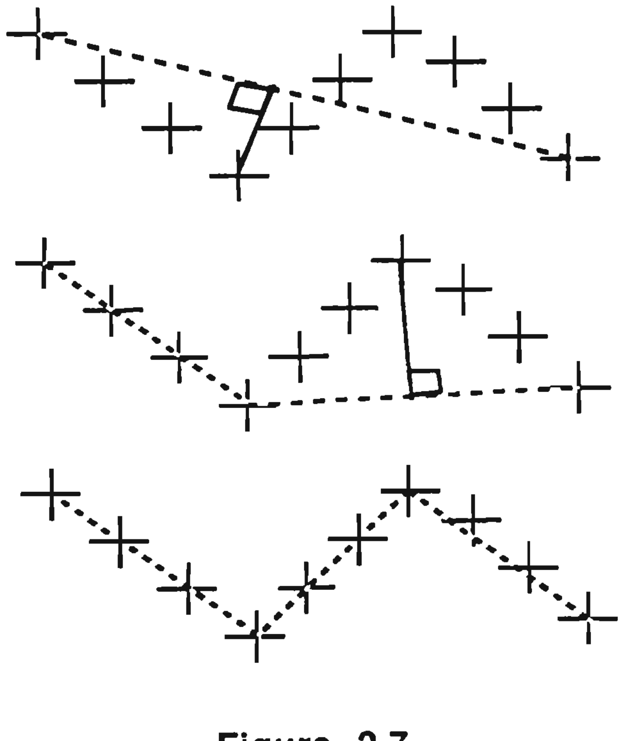


Figure 2.7

Stages of recursive line fitting

Straight line segments extracted from a sonar profile are called "sonar edges". Sonar edges represent the best approximation of the sonar images of the surfaces of obstacles. Sonar edge profiles of the environment include noise and uncertain data. In order to minimise the effect of noise, the sonar edges that are used to approximate obstacle edges are those which contain a contiguous set of N sonar range points i.e. sonar range readings that occur consecutively in the sonar profile. An example of extracting sonar edges is shown in Figure 2.8. Figure 2.8 (A) shows a small room and the accompanying sonar profile generated by recursive line fitting. The bold segments extracted from the sonar profile are the sonar edge segments which best approximate surface edges. In this example the number of contiguous readings N in a sonar edge has been set to 4. Figure 2.8 (B) shows the extracted sonar edges.

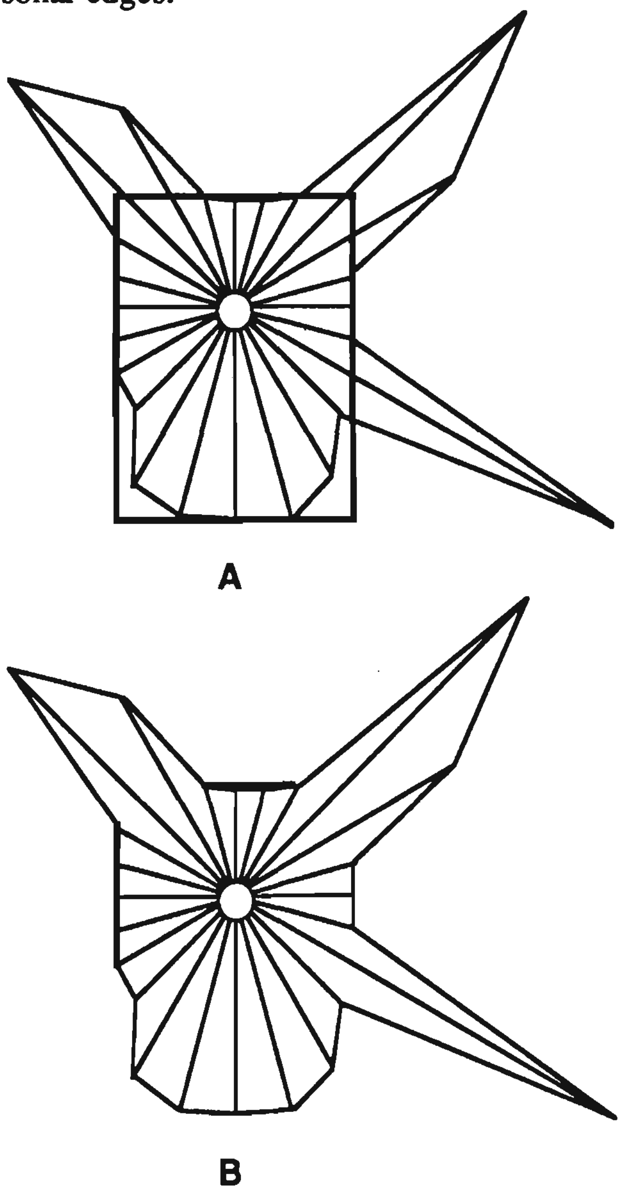


Figure 2.8

Extracting sonar edge segments.

2.4.3 Using the Sonar Mapping Test to Process Noisy Sonar Data

After a mobile robot equipped with sonar sensing has scanned an indoor environment, the following is known; the volume of space V_r, occupied by the mobile robot, and the two concentric volumes of space derived from the sonar profile which enclose V_r. Refer to Figure 2.9 for a diagram of the three volumes of space. The space occupied by the robot is "definitely free", the inner concentric volume space V_e, described by the sonar profile, is "probably empty". The remaining volume of space V_o, described by the sonar profile is "somewhere occupied".

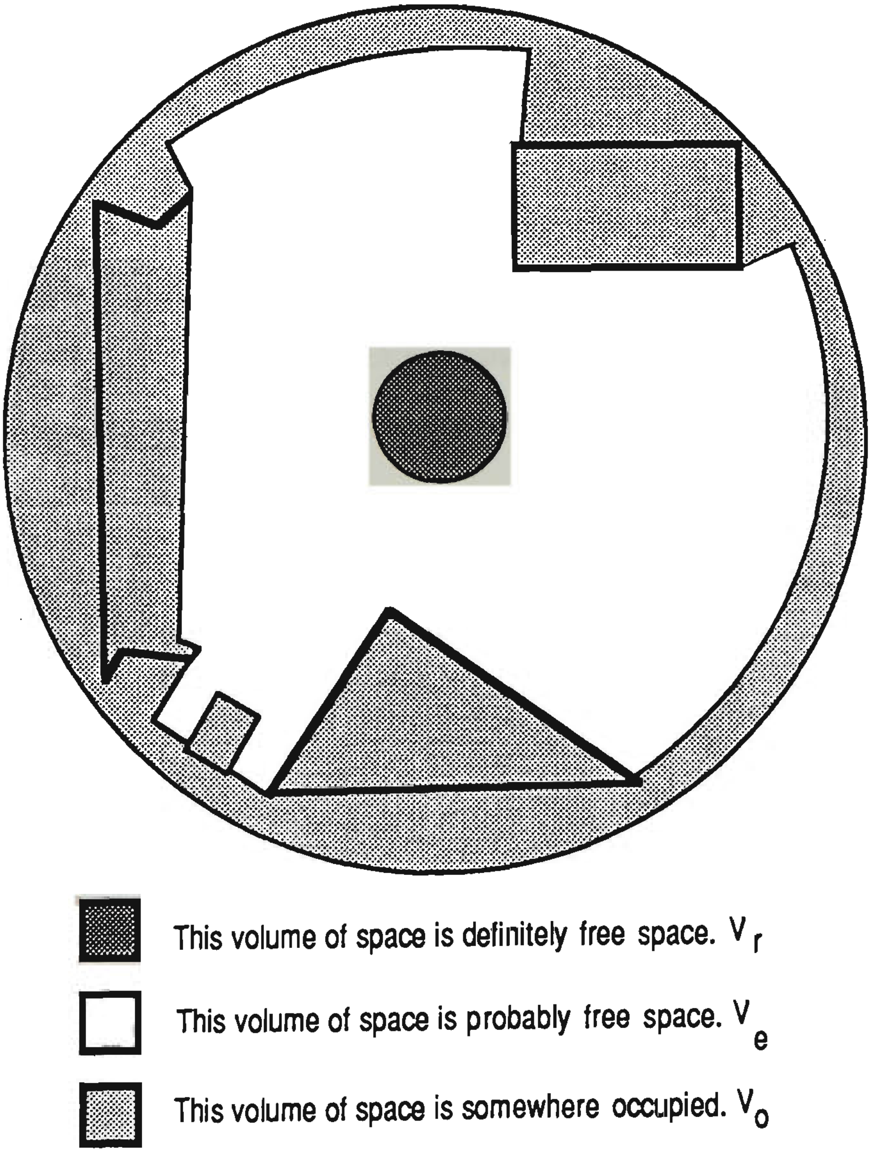

▩ This volume of space is definitely free space. V_r
□ This volume of space is probably free space. V_e
▨ This volume of space is somewhere occupied. V_o

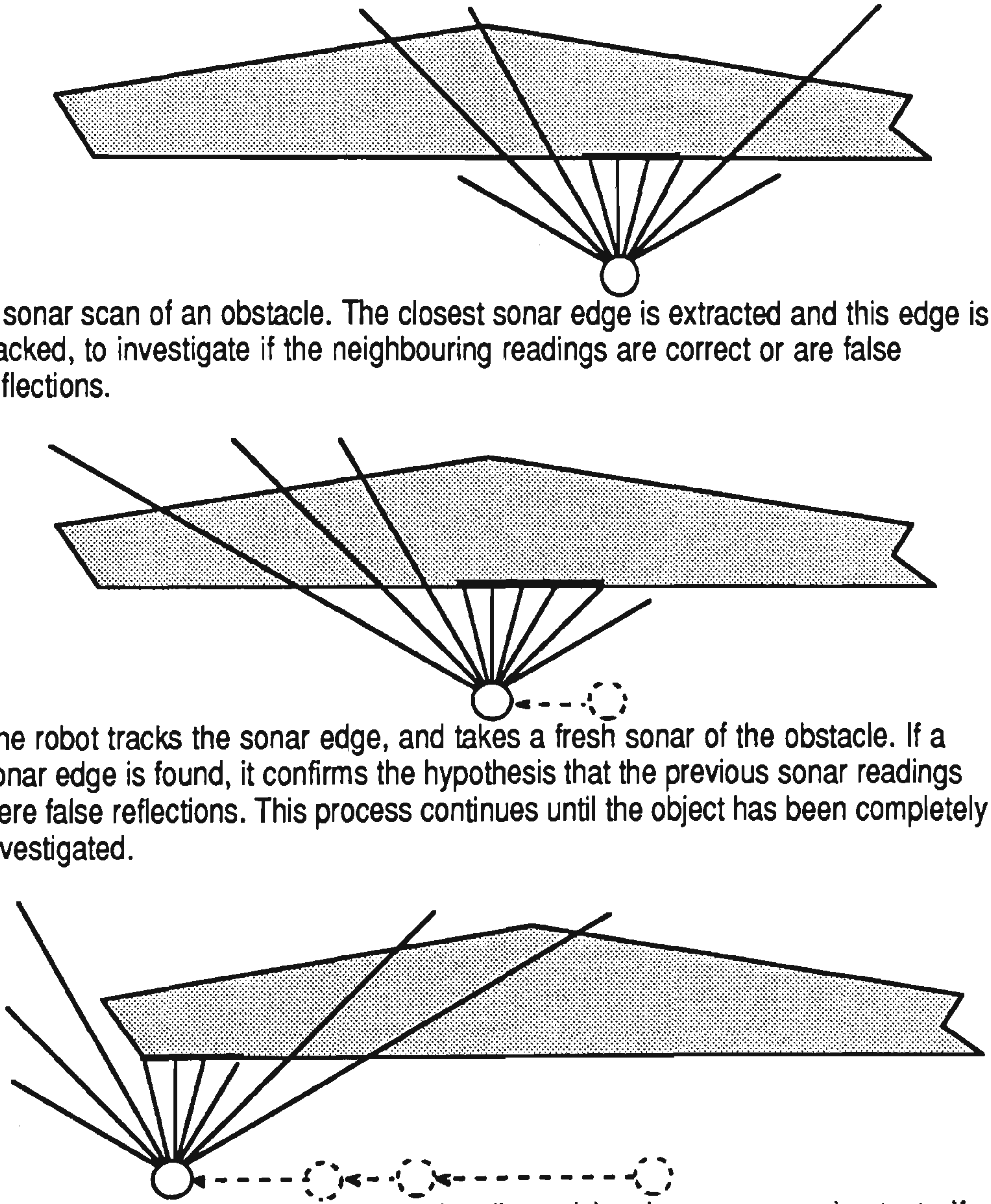
Figure 2.9

Volumes of Space in an indoor environment

The V_e volume space is verified to be empty if a mobile robot can travel through this space without collision. To verify the volume space V_o, the location and the size of obstacles in this space must be determined accurately. This is done by moving the robot into closer proximity of regions of space to be verified, and examining their sonar profiles.

Sampling the sonar profile of obstacles from a short distance D has the effect of greatly reducing the noise in the sonar data due to beam spread. This allows the surfaces of obstacles to be detected with high accuracy, which is particularly useful when trying to accurately map an environment which contains small objects and objects of irregular dimensions i.e. many edges and corners. There is a trade off between the accuracy of object edge detection and the discomfort of approaching obstacles too closely. The closer objects are approached to obtain sonar profiles, the greater is the accuracy of the detected sonar edges. However there is also a greater chance of collision with the obstacle. The value of D is determined experimentally; this is discussed in the Results section.

Eliminating beam spread from sonar data does not remove all the noise, as there is still the problem of false reflections. False reflections are eliminated by applying the sonar mapping test. The sonar mapping test is used to determine the shape of unknown objects given the fact that a sonar profile may appear to pass through objects. False reflections can be eliminated in the following manner (refer to Figure 2.10 for an example). Locate the robot in close proximity to an object whose shape is unknown, and carry out a sonar scan. From the sonar profile extract the sonar edge closest to the robot. This sonar edge is not blurred due to the minimal effect of beam spread. It was experimentally found that sonar edges close to the robot were measured accurately if the beam axis of the sonar sensors was positioned perpendicularly or nearly perpendicularly to the sonar edges. The closest sonar edge becomes the tracking edge used by the sonar mapping test. The tracking edge is extended in the direction where the edges of the obstacle being investigated are unknown. Extending the tracking edge will cause the tracking edge to either cut across or pass behind the distance measures of neighbouring sonar sensors.



A sonar scan of an obstacle. The closest sonar edge is extracted and this edge is tracked, to investigate if the neighbouring readings are correct or are false reflections.

The robot tracks the sonar edge, and takes a fresh sonar of the obstacle. If a sonar edge is found, it confirms the hypothesis that the previous sonar readings were false reflections. This process continues until the object has been completely investigated.

The robot tracks the sonar edge continually applying the sonar mapping test. If the sonar mapping test confirms that the previous sonar readings were correct, the robot then ceases tracking the sonar edge. The robot then searches for a new obstacle surface edge to investigate, and determine whether or not the obstacle has a convex corner.

Figure 2.10

Sonar Mapping Test

If the tracking edge cuts across the range measures of neighbouring sensors, then one of two possibilities has occurred; the range measures are correct and they represent a discontinuity in the obstacle edge being tracked i.e a gap of free space, or the range measures are incorrect due to false reflection errors. A hypothesis is made that the range readings are incorrect, and that the sonar beams are seemingly penetrating solid objects. To test the sonar mapping hypothesis, the robot is moved along the extended tracking edge until it is positioned in a location which is perpendicular to the suspect sonar readings. A fresh sonar scan is taken of the sonar mapping test region. If the nearest sonar edge has an orientation which is close to that of the track edge, and mates neatly with the track edge, then the hypothesis of the sonar mapping test is true. Otherwise the hypothesis is false, and a gap between obstacles has been found. The robot decides, based on the sonar profile, whether or not it can pass through the gap.

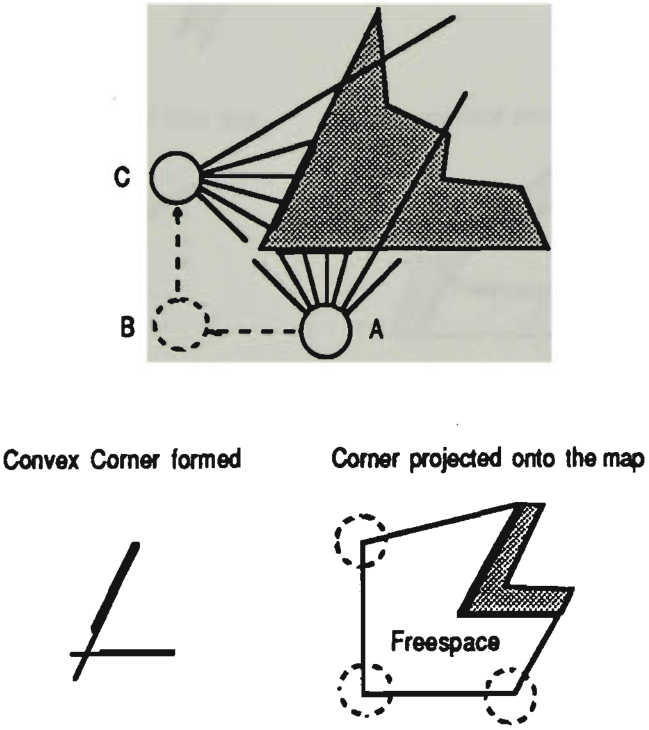


Figure 2.11

Investigating and mapping Convex Corners

If the robot can navigate through the gap, it does so and then proceeds to do a sonar scan to find the closest sonar edge of the obstacle which it is currently tracking. The closest edge and the previous tracking edge are projected to form a convex corner. Figure 2.11 shows an example of a robot mapping a convex corner. In this example the robot investigates and maps a surface edge of an obstacle by continually applying the sonar mapping test, until the test confirms that the tracking edge has terminated. This occurs when the robot moves from position A to position B. The robot investigates the gap of free space, and searches for a new edge to track, as shown at location C. Once a new sonar edge to track has been extracted, this edge and the previous tracking edge are projected together to form a convex corner. This corner together with the associated free space are then projected onto the environment grid map.

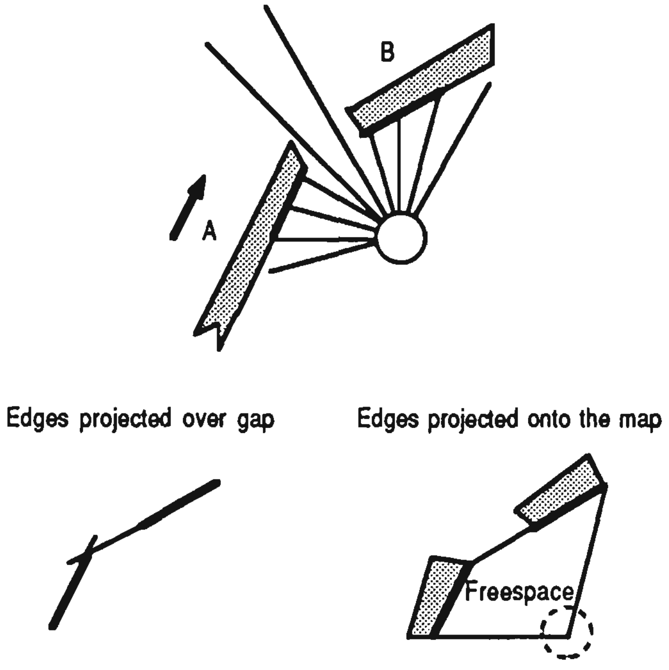


Figure 2.12
Investigating and mapping Impassable Gaps

However if the robot cannot pass through a gap, the robot performs a sonar scan to find the closest sonar edge of an obstacle that is currently *not* being tracked. The closest sonar edge and the current tracking edge are projected with free space between the two edges onto the environment map. Figure 2.12 shows an example of a robot mapping an impassable gap. This example shows a situation where the sonar mapping test confirms that the tracking edge A has terminated and that a gap exists, through which the robot is not able to pass. In this situation the robot finds the sonar edge which is preventing the robot passage, which in this case is sonar edge B. The freshly extracted sonar edge and the current tracking edge are extended to cover the gap. The new sonar edge and the current tracking edge, together with the associated free space, are then projected onto the environment grid map. The freshly sensed sensed edge becomes the new tracking edge.

When the tracking edge passes behind the distance measures of neighbouring sensors, this indicates that following the tracking edge too far could cause collision with another obstacle. It was found experimentally that the sonar profile for potential object collision edges was not subject to false reflections. To minimise beam spread errors the robot moves toward a potential collision edge, until the robot is D distance away from the edge. Once the robot is in close proximity to the potential collision edge a fresh sonar scan is done. The freshly sensed potential collision and tracking edges are extended to form a concave corner of the object being tracked. This concave corner is projected onto the environment map. Figure 2.13 shows an example of mapping a concave corner. This example shows a situation where it is not possible to apply the sonar mapping test. In this case a collision will result with surface edge B if the robot continues to track the sonar edge A. To avoid the collision, the robot extracts the potential collision sonar edge and extends this edge to form a concave corner with the current tracking edge. The corner together with the associated free space are then projected onto the environment grid map. The potential collision sonar edge becomes the new tracking edge.

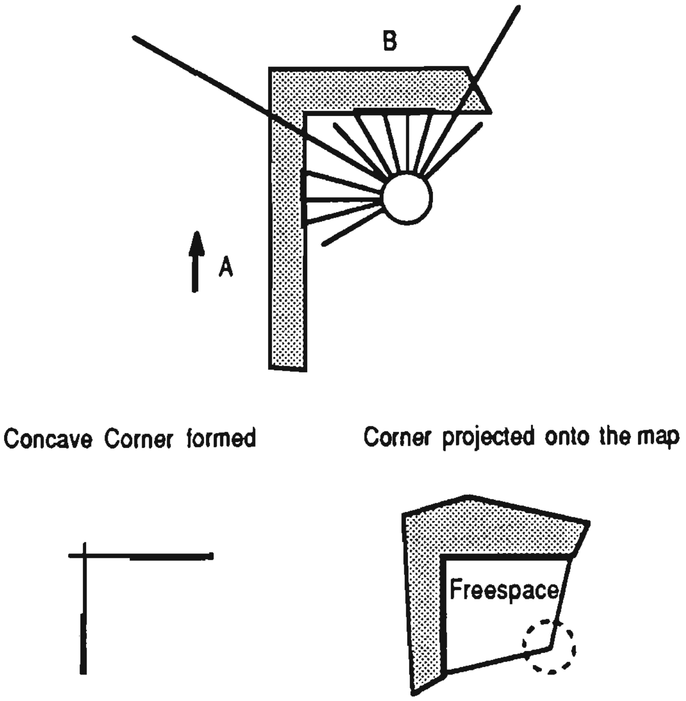


Figure 2.13

Investigating and mapping Concave Corners

2.4.4 The Environment Mapping Algorithm

The algorithm for a mobile robot equipped with sonar sensors mapping an unknown environment begins with the robot "waking up" in a unknown world. After waking up, the robot scans the environment with its sonar sensors, to ensure that it is further than R_{min} (the minimum sensor range) from any obstacles. Sonar readings that are equal to R_{min} indicate that the robot is at most a distance of R_{min} from neighbouring obstacles; the distance could be less. To ensure that the robot is at a distance greater than R_{min} from obstacles, the robot moves away from obstacles that are located R_{min} from itself. The wake up procedure can also be used to "herd" the robot to a desired location. This is done by surrounding the robot with obstacles such as humans, thus forcing the robot to move in a desired direction. If the robot is completely surrounded and can not move, it goes to "sleep" and then wakes up after a timeout T_{out}, when it will again attempt to move away from the obstacles that it perceives are too close.

Upon completion of the wake up procedure, the robot takes a fresh sonar scan of the environment. The sonar scan is examined for obstacles within a radius of D_{max} of the robot. The distance D_{max} represents the radius of the cylindrical volume around the robot which needs to be investigated and learnt. Sonar beam spread at large distances can give the effect that openings through which the robot can pass e.g. doorways, are perceived to be closed. The distance D_{max} is set to a value such that openings through which the robot can pass do not appear to be closed. The obstacles within D_{max} of the robot are isolated to contiguous sectors of the sonar profile, i.e. consecutive sonar readings which are all less than D_{max}. The robot systematically examines, in clockwise order, the obstacle sectors in the sonar profile in order to accurately map the surfaces of obstacles in the environment. In each sector the robot moves into close proximity to the obstacle in the sector, and the sonar mapping test is applied to a fresh sonar scan. Refer to Figure 2.14 for an illustration of finding obstacles within D_{max} of the robot. In this example distances that are within D_{max} of the robot are shown in the shaded circular area. Range readings within D_{max} are isolated into contiguous sectors. Should two sonar sectors be separated by a single sonar reading, a simple heuristic is used to decide whether the separating sonar reading is a false reflection and that the two sonar sectors can be merged. If the separating sonar reading is sufficiently greater than its neighbouring sonar readings, then the separating reading is considered to be a false reflection. In this case the separating reading is replaced with the average of its neighbouring sonar readings. In the this example the false reflections have been removed and sonar sectors have been merged. The sectors are numbered in the order they will be investigated by the robot.

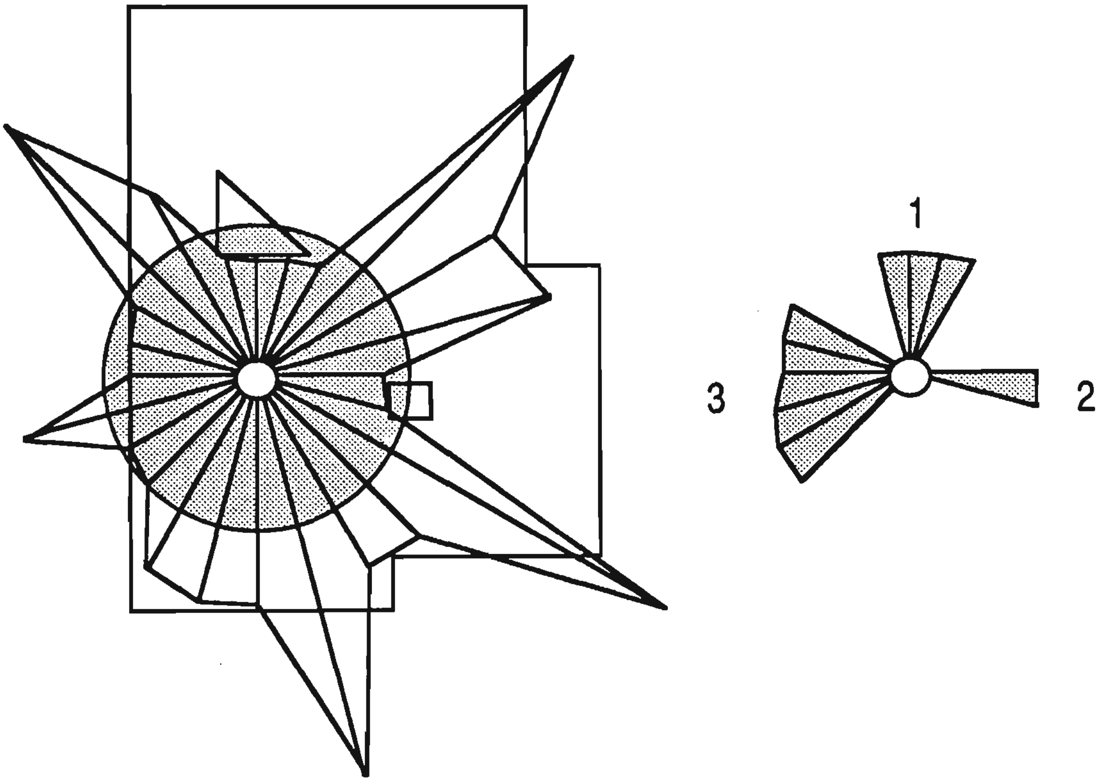


Figure 2.14

Deciding which obstacles to Investigate and map

The mobile robot moves along the axis of an obstacle sector, to a position that is distance D from the obstacle being investigated. A fresh sonar scan is taken of the environment, and a recursive line fit is applied to the sonar data belonging to the obstacle sector under investigation. The closest sonar edge is extracted from the straight line sonar profile. This sonar edge becomes the tracking edge which the robot follows to obtain the sonar profile of the obstacle. The tracking of the sonar edge, and its incorporation into the environment map, using the sonar mapping test is described in detail in Section 2.4.3.

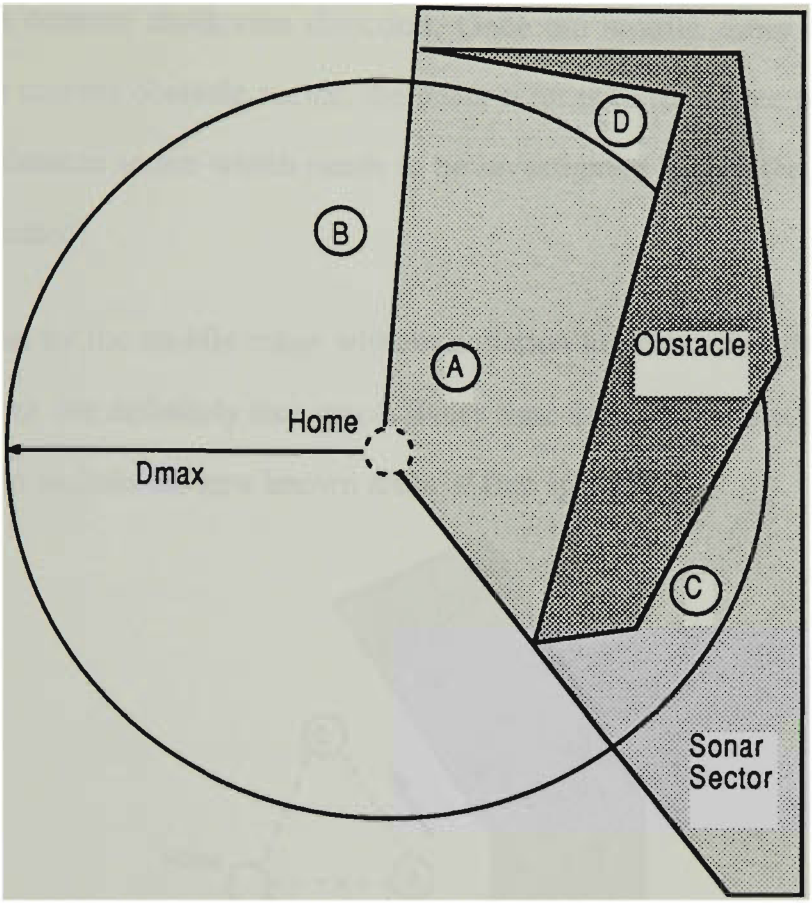


Figure 2.15

Deciding which obstacles to Investigate and map

Initially the robot tracks the closest sonar edge in a clockwise direction, until one of three possible conditions occurs: the robot moves outside of the sonar sector being mapped, the robot is no longer within the distance D_{max} of the home position, or the robot moves to a position which is no longer in line of sight to the home position i.e. moves behind an obstacle. The first two conditions are verified by encoders on the robot's wheels and the third condition is verified by sonar. Refer to Figure 2.15 for examples of the three conditions to terminate tracking. In the example position A satisfies all three tests. Position B fails the inside sonar sector test. Position C fails the home visible test. Position D fails the test of being within D_{max} of home. When one of the three above conditions occurs, the mobile robot finishes tracking and mapping the current sonar edge, and backtracks to the position where it began investigating the obstacle. Once the robot has returned to its initial tracking position, it then proceeds to investigate the remaining unmapped portion of the obstacle sector. This is done by tracking the closest sonar edge in a counter clockwise direction. Once the mobile robot has completely investigated the current obstacle sector, the robot returns to the home position. At this point the next obstacle sector which needs to be investigated is selected, and the whole procedure is repeated.

Every motion by the mobile robot without collision indicates that the path volumes swept by the robot are definitely free space. Every time the robot moves the environment map is updated to include the new known areas of free space.

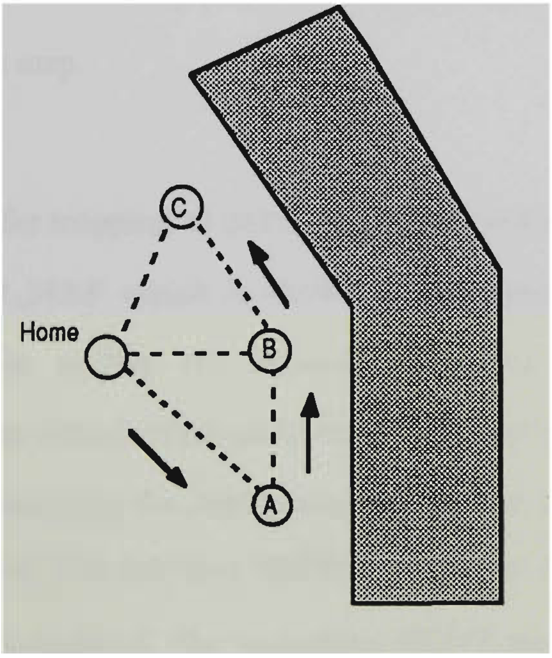


Figure 2.16

Updating free space while the home location is visible

During the tracking of sonar edges in obstacle sectors, termination of the tracking is based upon the home position being in line of sight of the current robot position. Should the home position be visible, then the triangular region enclosed by the home position, the previous robot position and the current robot position is updated as free space on the environment map. Refer to Figure 2.16 for an illustration of updating free space when the home location is visible. In this example the robot movements are shown with arrows. After the robot moves from position A to Position B the sector enclosed between A, B and Home is updated as free space. Similarly after the robot reaches Position C, the free space sector enclosed between B, C and Home is updated as free space.

Once the mobile robot has investigated all the obstacles within a radius of D_{max} of the robot, all the sectors which were perceived to be free space are projected onto the environment map.

It should be noted that free space regions generated from sonar data are not written over the information previously recorded about the region if the same region has been labelled as an obstacle region. This action is necessary to prevent false reflections of sonar beams, which were not screened out by preprocessing and the sonar mapping test, from corrupting the environment map.

2.5 Results

The general algorithm for mapping an unknown environment using sonar is described in the procedure BUILD_MAP which is shown in Algorithm 2.1. The procedure WAKEUP performs the action of waking the robot up. The procedure FIND_OBSTACLE extracts obstacles that are in close proximity to the robot from a sonar profile. It does this by identifying the contiguous sonar sector readings that are within D_{max} distance of the robot. The function MIDDLE finds the middle area of obstacle region currently being investigated. The procedure MOVE moves the robot from its present location to a new location, and also updates the environment map with the free space volumes swept by the robot during the execution of the move motion. Once the robot has moved sufficiently close to an obstacle under investigation, it maps the obstacle and updates the environment map. Obstacle mapping is performed by the procedure MAPPER and the details of this procedure were discussed in Section 2.4.3. The BACKTRACK procedure retraces the robot path from the current location back to the point where it began investigating an obstacle. The UPDATE_FREE procedure updates the environment map with the areas which are not occupied by obstacles within D_{max} of the robot i.e. free space areas. Note that all the algorithms described require parameter passing to subroutines in one of two ways: by reference which allows the value to change (underlined), or by value which does not allow the variable to change (plain).

```
procedure BUILD_MAP ( Map )
    perform WAKEUP ( Sonar, Current )
    perform FIND_OBSTACLE ( Sonar, ObstacleSectors )
    Home := Current
    dowhile ( There are more ObstacleSectors )
        Aim := MIDDLE ( ObstacleSector )
        perform MOVE ( Current, Aim, Map )
        perform MAPPER ( Clockwise, Home, ObstacleSector, Current, Map, History )
        perform BACKTRACK ( Current, Aim, History )
        History := Nil
        perform MAPPER ( AntiClockwise, Home, ObstacleSector, Current, Map, History )
        perform BACKTRACK ( Current, Aim, History )
        History := Nil
        perform MOVE ( Current, Home, Map )
    enddo
    perform UPDATE_FREE ( Sonar, Map )
end procedure
```

Algorithm 2.1

Mapping an unknown environment

The sonar generated environment maps were obtained from four different setups of a Robotics Laboratory at Monash University. The layout of the laboratory was varied by moving and rearranging the laboratory furniture. In order to be able to quickly change the layout of the laboratory and to test different obstacle configurations, cardboard boxes of various shapes and sizes were used to build different room layouts. Figures 2.17, 2.18, 2.19 and 2.20 illustrate the different laboratory layouts and the corresponding sonar maps. Each result is represented by two diagrams. Diagram (A) shows a configuration of obstacles in an indoor environment which must be mapped. Diagram (B) shows the robot produced sonar map of the environment described in Diagram (A). In the sonar map diagram the dark shaded areas are obstacles, the lightly shaded areas are unknown regions which have not been mapped, and the unshaded areas are free space. The sonar maps also show the paths that were executed by the robot during map making. These execution paths are shown as directed arrows, and they are labelled in the order in which they were executed i.e path AB, followed by path CD etc.

To produce accurate maps a robot must be able to estimate its position and orientation in an environment. In all the experiments the robot was accurately located to a home position initially. From this home position the robot investigated its surroundings. Estimation of the robot position during the course of mapping the surroundings of the home position was done by dead reckoning. Once the robot completed mapping, it returned to the home position, where its estimates of position and orientation were corrected by comparison with a global source i.e. beacons etc. In the future it is proposed to use a beacon system to correct the robot location, but in these experiments the errors in location were corrected manually. It was found during the course of experimentation that the drift in the robot position was small, less than 0.25 ft. when the robot had returned to its home position. Thus the mapping errors due to dead reckoning errors were negligible.

The environment map grid size was set at 0.5 ft. The motivation for this choice of grid size was that the floor of the experimental laboratory was made up of 1.0 ft. square tiles. Thus it was easy to verify the correctness and the accuracy of the robot generated maps.

In terms of the variables previously mentioned in the text, the algorithm parameters used in these experiments were:

R_{min}	= 0.9 ft.	(The minimum range of the sonar sensor)
R_{max}	= 25.6 ft.	(The maximum range of the sonar sensor)
H	= 2.32 ft.	(The height of the sonar range sensor above the floor)
θ	= 15^{o}	(The angular dispersion of the sonar beam spread)
R_g	= H/Sinq = 9.0 ft.	(The practical sensor range)
R_u	= $\min(R_g, R_{max})$ = 9.0 ft.	(The useful range of the sonar sensor)
C_p	= 0.35	(The proportion of sonar readings which account for a cluster)
C_n	= 10	(The number of sonar readings taken prior to cluster analysis)

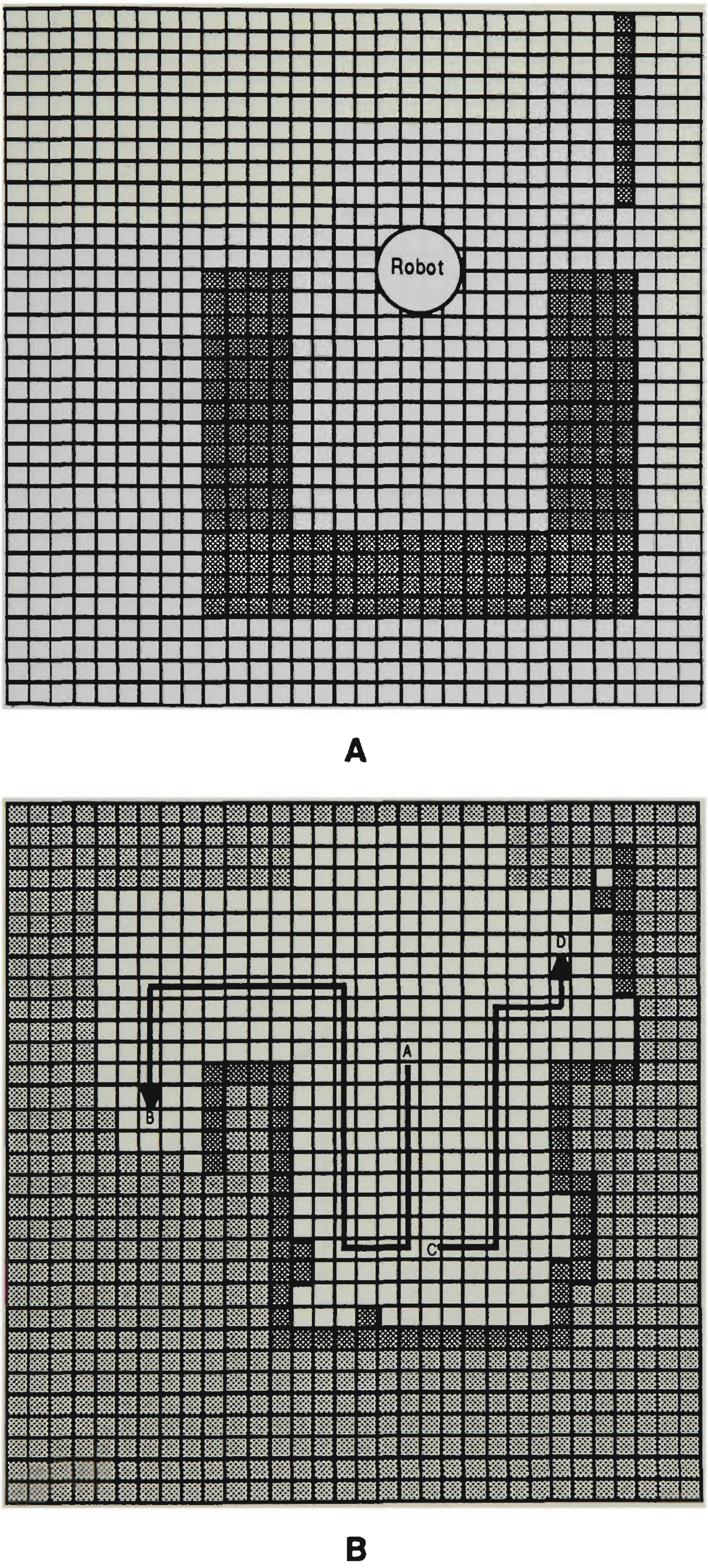
N	=	3	(The minimum number of sonar readings in a sonar edge)
T	=	0.2 ft.	(The maximum allowable perpendicular deviation of a sonar reading)
D	=	1.0 ft.	(The distance from obstacles for sonar range scans)
D_{max}	=	4.0 ft.	(The maximum radius in which obstacles are mapped)
T_{out}	=	30 sec.	(The time the robot sleeps before waking up, to try to move)

This set of parameter values was used for all the reported results. This set may not be optimum, since they were selected based upon trial and error experimentation. Since theoretically predict the behaviour of a low resolution sonar rangefinder such as the Polaroid Corp. Ultrasonic Rangefinder is very difficult, an effective way of determining operating parameters for the mapping algorithm is to run extensive tests of the system in the environment where it is to be used, and to adjust the various system parameters until the system is "tuned".

The program for these experiments was written in PASCAL on a VAX-11/750 minicomputer. The execution times of the mapping program for the four different laboratory layouts are given in Figures 2.17 - 2.20 and are as follows:

Figure 2.17	-	5.50 minutes
Figure 2.18	-	2.39 minutes
Figure 2.19	-	8.29 minutes
Figure 2.20	-	4.11 minutes

The execution times reduced by 30% when I/O functions such as terminal status displays, debug information etc. were disabled.



A

B

Figure 2.17

Sonar mapping an indoor environment with a robot.

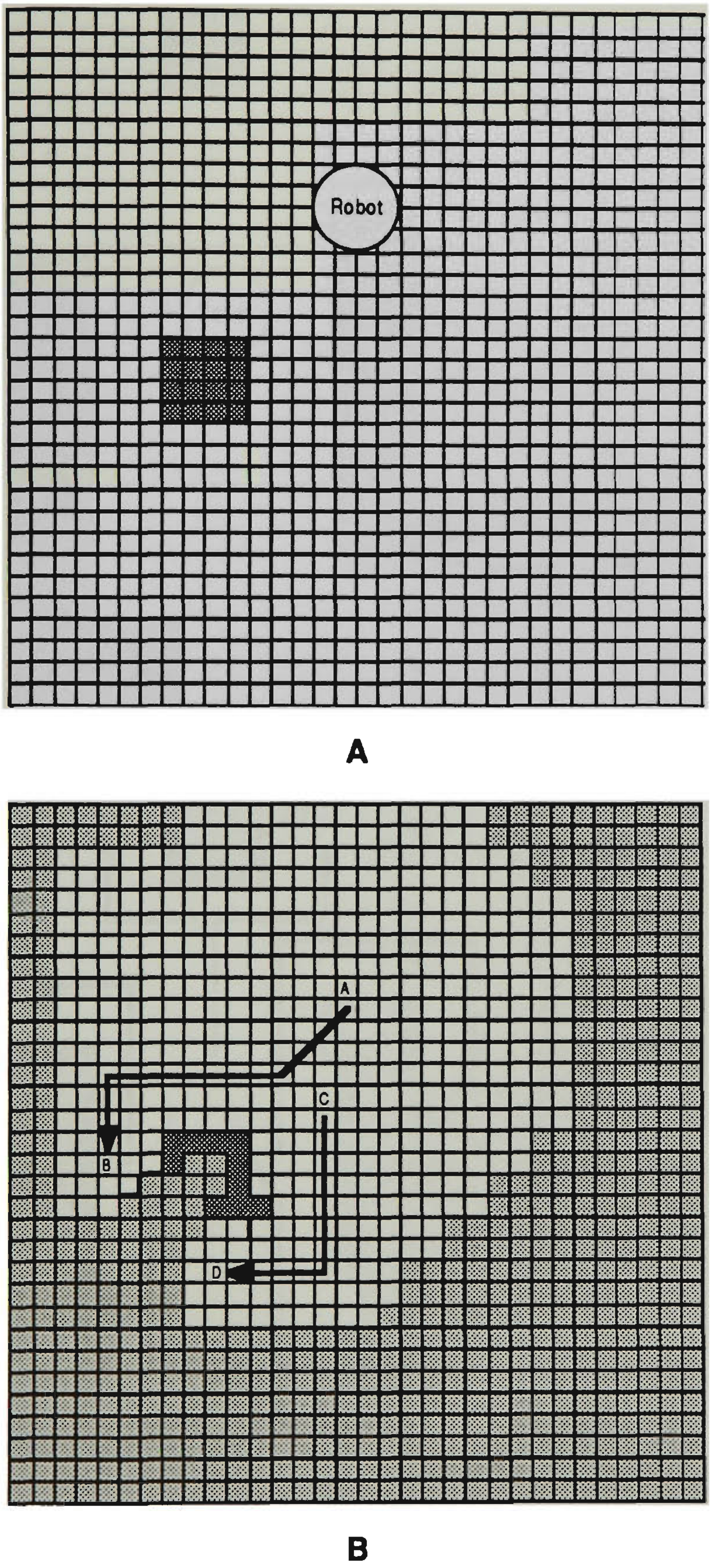
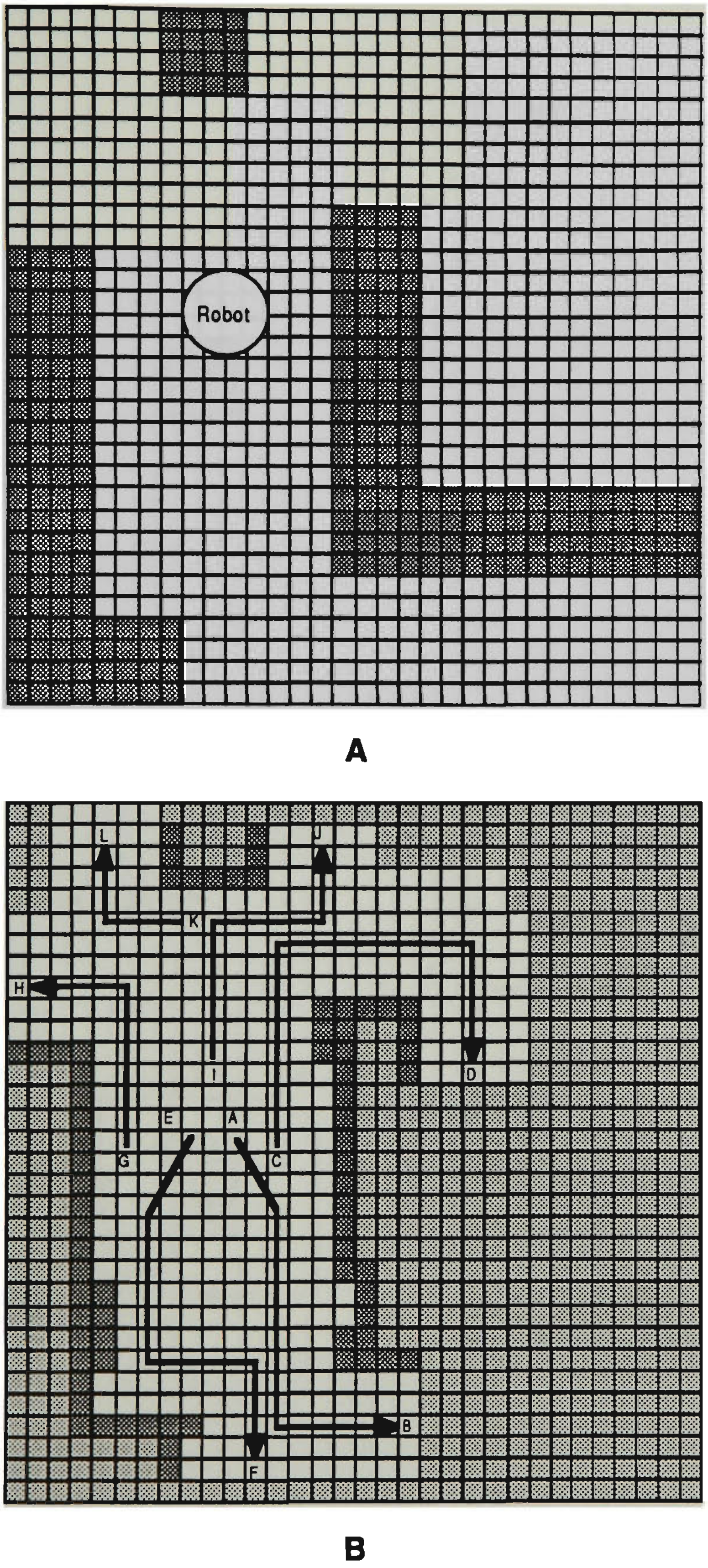


Figure 2.18

Sonar mapping an indoor environment with a robot.

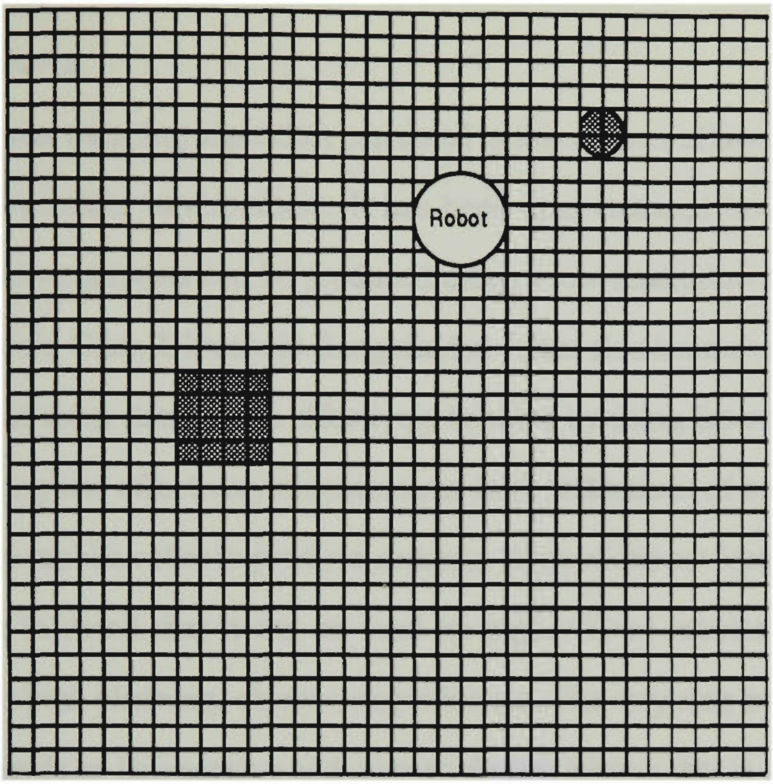


A

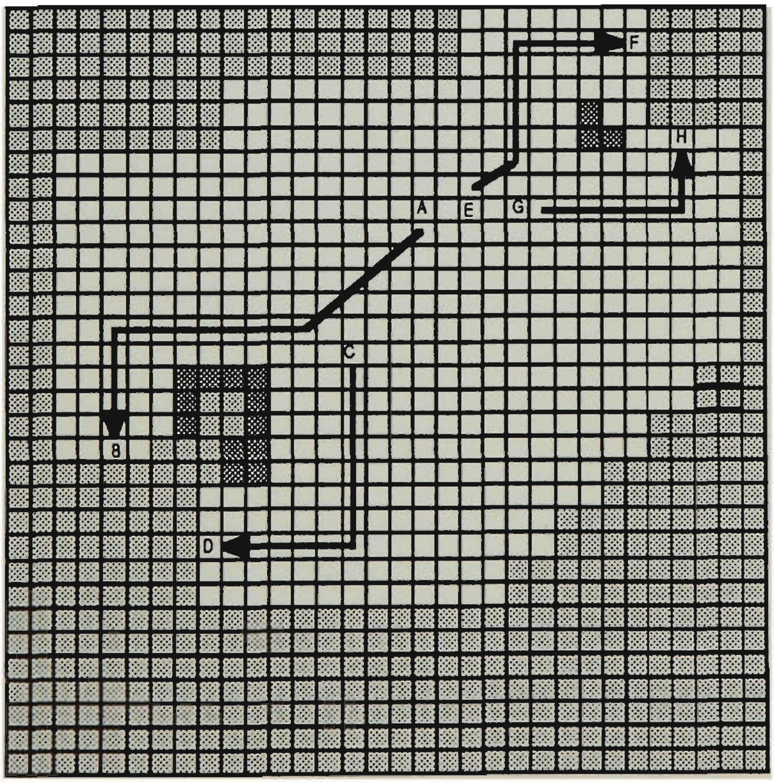
B

Figure 2.19

Sonar mapping an indoor environment with a robot.



A



B

Figure 2.20

Sonar mapping an indoor environment with a robot.

2.6 Conclusions

The environment mapping algorithm presented in this chapter allows a mobile robot to map an unknown indoor environment. It has been demonstrated how high resolution maps of indoor environments can be produced using a low resolution sonar rangefinder, such as the Polaroid Corp. Ultrasonic Rangefinder. It has also been shown that the noise and uncertainty of sonar data can be effectively handled by using the sonar mapping test. The sonar mapping test effectively discriminates false reflections of sonar sound waves, thus allowing the mobile robot to produce accurate maps of the environment. The map is sufficiently rich in detail that it can be used by higher level mobile robot navigation functions such as path planning, object recognition etc. The mapping technique described in this chapter yields an inexpensive and reasonably fast system that is suitable for indoor environments.

Chapter 3
Review of Path Planning

3.1 Introduction

The problem of finding optimum paths for robot manipulators and autonomous mobile robots through environments cluttered with obstacles has attracted much research interest. A great deal of this research has concentrated on situations where the environment in which the robot operates is completely known and supplied to a path planner. For a robot navigating in a partially known or completely unknown environment these path planning techniques are often not directly applicable or extendible. For example difficulties arise in deciding how to treat the unexplored regions of the environment. A typical approach is to treat unexplored regions as obstacles, and only proceed into the unexplored regions if the goal lies there. In this chapter a review will be presented of path planning algorithms which allow a mobile robot to function in known, unknown and partially known environments. On completion of the review, a list of desirable features for a mobile robot path planning algorithm will be given. A discussion will also be presented of the type of data structure that is necessary to support a superior path planning algorithm.

Before reviewing past research into path planning, it is useful to think about the path planning problem from the human stand point. In other words, to look at how humans tackle the task of path planning. Examining the path planning problem from this view point could provide an insight into useful strategies and features that a robot path planner should possess. These strategies and features should be kept in mind while evaluating robot path planning algorithms.

The remainder of this chapter is organised in the following manner. Section 3.2 discusses the philosophy of robot navigation used in this thesis. Section 3.3 reviews past robot path planning research. Section 3.4 discusses the various data structures that can be used to model a robot's environment to support path planning. Finally in Section 3.5 the conclusions that have drawn from the review of mobile robot path planning research are presented. In this section the data structure that has been chosen to model the environment is described, together with an outline of the features that the new path planning algorithms, which have been developed in this research, use to overcome the shortcomings of previous approaches.

3.2 Philosophy of Mobile Robot Navigation

Before developing navigation algorithms for a mobile robot, it is useful to think about how humans perform the task of navigation. Consider a person in an unfamiliar city who asks for directions to a particular destination. The reply may be that the destination is a certain distance in a certain direction. The person will then proceed to the goal using the most direct route. If the person is fortunate the goal is achieved without deviation from the planned path. However this is rarely the case, since both stationary and moving obstacles will be encountered. Should an obstacle be met during the course of the journey to the goal the person classifies the obstacle as either stationary or moving. If the obstacle is stationary the person notes the location and features of the obstacle, whilst for moving obstacles the person notes only the features of the obstacle. At this point the person revises the original plan using the new information. If the obstacle which is blocking the person's passage to the goal is stationary, he or she revises the plan to go around the obstacle, based on his or her knowledge of the environment. For the case of the moving obstacle, assuming the person is patient, he or she will wait until the moving obstacle is out of the way. Such a strategy is a safe one, particularly in potentially dangerous situations, such as crossing busy streets. If a moving obstacle comes to rest for a significant period of time, then it too can be treated as a stationary obstacle. This process of planning and executing the plans continues until the person either reaches the goal or deduces that the goal is not reachable. A goal is unreachable if it can not be reached from any route. For example the goal is unreachable when all the routes have been cut due to flooding.

Should the person later need to travel to the same goal, then one of three approaches can be used. The person can proceed to the goal using the knowledge of the environment learnt on previous journeys. This approach will produce better paths than were previously executed, and is favoured if the person is in a hurry or is in a conservative mood. With the second approach the person can choose to be adventurous and can spend time exploring alternative paths to the goal, and in doing so learn more about the environment. A third approach can be taken if a significant period of time has transpired since the previous journey. In this case the person may have forgotten some information about the structure of the environment. The way to navigate to the goal is to follow the strategy of navigation in an unknown environment. This means the robot must relearn the environment.

On the other hand, the person could be an inquisitive tourist, and may wish to systematically visit all locations of interest in the city. The person will devise a plan which will minimise the distance which must be traversed to visit all the locations of interest. However it should be noted that most probably this plan will not include a solution based on the "travelling salesman" problem. Instead a path is generated which visits all locations but in the process performs some backtracking. This path is easy to compute.

Another observation that can be made about people navigating from one location to another is that they do not always select the shortest paths to a goal location. A path which minimises the distance to the goal, but also takes into account the discomfort of clipping the corners of obstacles, and the cost of venturing into unknown areas may be selected. People tend to choose to walk in uncluttered areas, and only come into close proximity to obstacles, or venture into the unknown, if necessary.

Fortunately, humans are approximately cylindrical in shape, so there is no need to compute complex paths in crowded situations. However an awkward object such as a ladder is being carried from one location to another, what navigation strategy should be used? Obviously people do not sit down and compute the best solution path. A path which has maximal clearance from obstacles is likely to be chosen even though this path may be longer than the shortest path. Such a path has the advantage of minimising the chances of collision with obstacles when making turns. People only try to squeeze through tight corners, only when it is necessary. In such cases a best first strategy is used, where the ladder is moved around the corner, starting from a position which has the greatest clearance. As the ladder is moved, a check for potential collisions with the walls is continually performed, and the motion path of the ladder is modified accordingly. A situation may arise where the ladder gets stuck and another approach must be tried. The final solution path is probably not an optimum path; however it is a negotiable path which can be found with the minimum mental effort.

These observations are pertinent to mobile robot navigation. A robot must have the capability to operate in an environment about which it possesses incomplete knowledge, and the robot must have a mechanism to acquire new knowledge about the environment, and to add this new information to the knowledge the robot has previously learnt.

It may often be desirable for a robot to follow the shortest path to a goal. However the safety of the robot becomes important when there are uncertainties in the environment information, such as the exact shape and position of obstacles. Due to the limitations of current sensor technology there will often be uncertainty. This problem is compounded by the uncertainty in the control of a robot i.e. the precise position of a robot is not always known by the robot's control system. Therefore the capability to plan paths with consideration of safety of the robot is essential for a mobile robot navigation system.

The observed human strategy of moving objects such as ladders can assist in path planning for non cylindrical robots with 3 degrees of freedom (DOF). This avoids the need to search the entire solution space, which is computationally very expensive. Such approaches will be discussed in the next section. Only the areas of the environment that have the least chance of collision, and thus have the greatest chance of success are searched.

A robot should have the capability of executing different kinds of path planning behaviours, such as adventurous, conservative and safe paths, rather than just the shortest path. Exploring new areas is computationally expensive, so the capability to decide whether or not this computation should be undertaken is very useful. Similarly a robot having the capability of visiting all the locations in an environment has a purpose, particularly for floor cleaning or security robots.

In Chapter 4 of this thesis a new algorithm will be presented which imitates the human navigation skill of reaching a goal in an unknown environment. This algorithm is based on a single concept which is easily extended to accommodate the path planning behaviours discussed earlier in this section. Research work concerning different robot path planning behaviours is presented in Chapter 5. In Chapter 7 an algorithm for robot path planning with 3 DOF is presented which takes into account the observations made in this section.

3.3 Review of Mobile Robot Navigation Research

This section outlines several approaches to mobile robot path planning and the drawbacks of each approach. The abstract representation of the environment used by each path planner is discussed, together with what information is made explicit by each environment model and what information is thrown away by the choice of abstract representation. The diverse approaches to path planning are classified into four groupings; vertex graph, free space, superimposed grid and potential field methods. These are presented in Sections 3.3.1 to 3.3.4 respectively. The problem of path planning for mobile robots with 3 DOF is treated separately in Section 3.3.5. Finally in Section 3.3.6 the consideration of robot safety during path planning is discussed. This topic is given a separate treatment, since all path planners touch on the issue of robot safety in some way. Each path planner has its own approach to the problem, ranging from ignoring the issue to having a specific mechanism to deal with the matter. This section reviews the approaches to robot safety used by the various types of path planners.

3.3.1 Vertex Graphs

In vertex graph path planning [Keirsey 84, Lozano-Perez *et. al.* 79, Lozano-Perez 83, Moravec 80, Thompson 77] the obstacles in the environment map are expanded by the radius of the robot and the robot is conceptually shrunk to a point. The problem of finding a path for the whole robot through the obstacle strewn environment is exactly the same as finding a path for a point through the expanded obstacles. Conceptually this method is equivalent to placing a string at the initial and goal positions and drawing it taut. A graph is constructed by joining the "line of sight" vertices. The graph is searched by a standard AI search technique such as breadth first or A* [Dijkstra 59, Hart *et. al.* 68, Tarjan 81] to find the shortest path from the initial position to the goal. Modifications of the A* algorithm [Keirsey 84, Thompson 77] are attempts to avoid building parts of the graph by heuristic pruning. Refer to Figure 3.1 for an example of vertex graph path planning. In this figure the robot shown as a black square, and is assumed to be cylindrical. The obstacles in the environment are expanded by the radius of the cylindrical robot. All the line of sight vertices are connected. The broken lines indicate the possible paths between the start and goal positions. The solid line indicates the solution path.

Vertex graph methods in their endeavour to find the shortest path through the graph clip the corners of obstacles and run down the edges of obstacles. This is called the "too close" problem [Thorpe 84]. In practice following such paths may result in collisions with obstacles due to the inaccuracies of a mobile robot. This problem can be countered by expanding the obstacles an extra amount to avoid such collisions. However the penalty for this strategy is that possible solution paths are blocked. Another source of error is introduced by approximating the robot to be cylindrical, which could exclude potential solution paths.

A problem with vertex graph path planners is that they have no mechanism to handle the "too close" problem. Also it is assumed that the shortest path is always the best path. In addition vertex graph planners have problems with partially known environments. One way to avoid the problem is to treat the unknown regions as obstacles [Thompson 77]. In the solution put forward by [Keirsey 84], the search graph is built as the environment is explored. However this method cannot find the shortest path to the goal which includes the traversal of unknown regions; instead it can only search for the shortest path from the available knowledge of the environment. The robot will only venture into unknown areas if no solution path exists in its available knowledge.

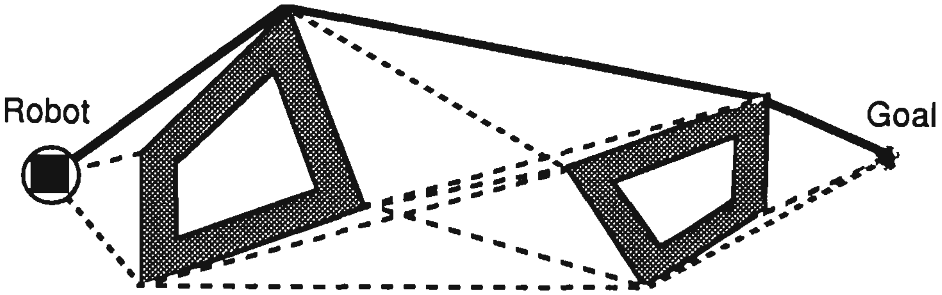


Figure 3.1
Vertex Graph Path Planning.

3.3.2 Free Space

Free space path planners deal with the free space available for a robot to navigate in rather than dealing with the obstacles to avoid. One approach is to model the free space as convex polygons [Chatila 82, Crowley 85]. The approach adopted by [Brooks 83] is to model the free space as generalised cones. [Kuan *et. al.* 85] use a hybrid of convex polygons and generalised cones called mixed space. An alternate method is to use Voronoi diagrams [O'Rourke 84, Miller 85] or modified Voronoi diagrams [Ilari *et. al.* 90]. Effectively with all these methods a path is steered down the middle of "corridors" of free space. The individual free space areas which are passable to the robot (i.e. wide enough) are included in a graph. The graph forms a network of possible paths. This graph is searched for the shortest path using the same methods as discussed in Section 3.3.1. The drawback of free search methods is the strategy of moving down the middle of corridors, since this approach may deviate significantly from the shortest solution path. This is called the "too far" problem [Thorpe 84]. Refer to Figure 3.2 for an example of free space path planning. In this figure the robot is shown as a black square and is assumed to be cylindrical. The obstacles in the environment are expanded by the radius of the cylindrical robot. All the free space is decomposed into interconnected regions. The broken lines indicate the free space regions. The heavy dotted lines depict possible paths between the start and goal positions. The solid line indicates the solution path

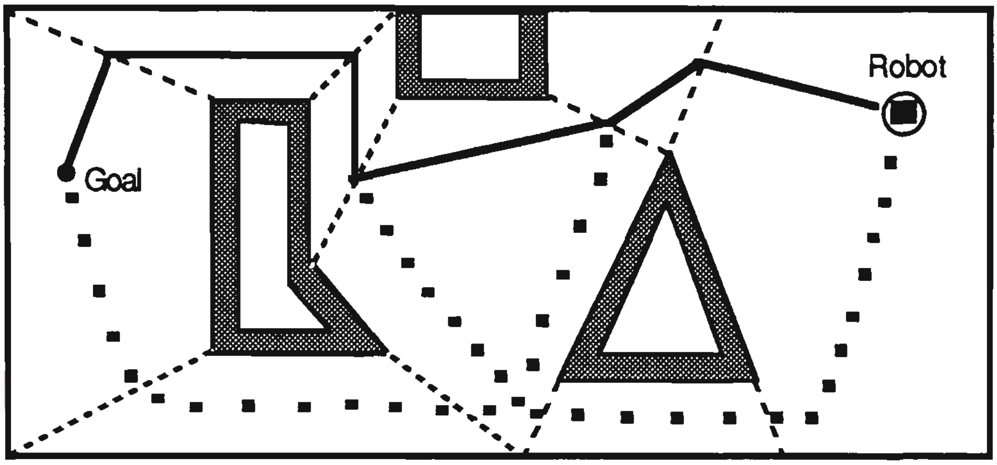


Figure 3.2

Free Space Path Planning.

3.3.3 Superimposed Grid

Grid path planning methods superimpose onto the environment a regular grid [Jarvis *et. al.* 86, Thorpe 84]. Each grid point can be 4 or 8 connected to its neighbours, thus forming a graph. Each node or grid cell contains information about whether the node is inside or outside an obstacle. The graph is searched for the shortest path using the techniques described in Section 3.3.1. Refer to Figure 3.3 for an example of superimposed grid path planning. In this figure the robot is shown as a black square and is assumed to be cylindrical. The obstacles in the environment are expanded by the radius of the cylindrical robot. The solid lines indicate the solution path to the goals.

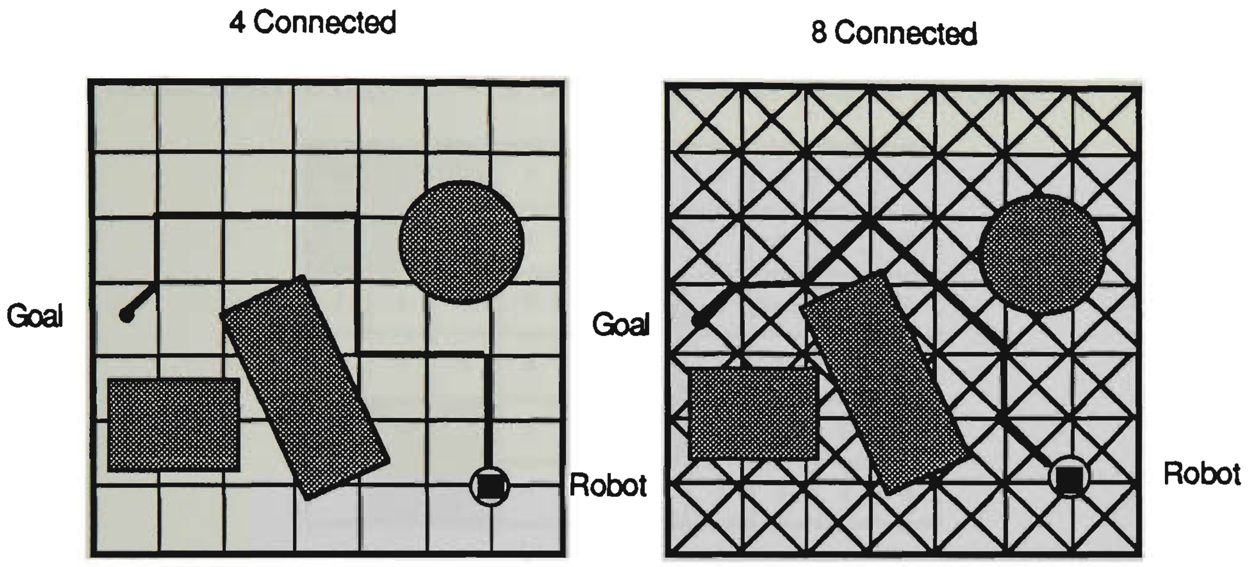
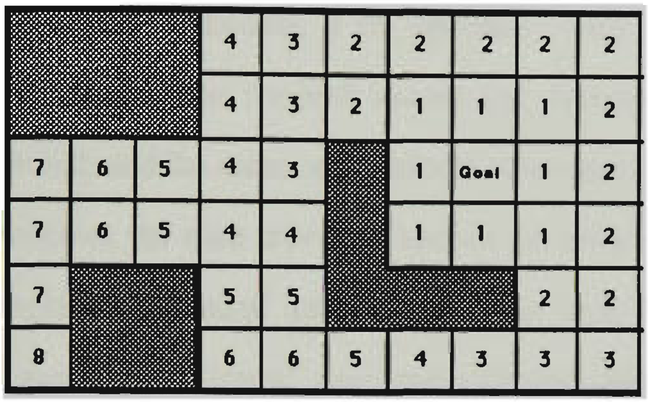


Figure 3.3

Superimposed Grid Path Planning.

The superimposed grid approach has its limitations; for example the solution path can suffer the "too close" problem, or it can zigzag, or due to a large grid size not be the true shortest path [Jarvis *et. al.* 86]. The path planner described by [Thorpe 84] remedies some of the drawbacks of the superimposed grid, by the use of cost functions, whose values depend on how close the node is to obstacles. This avoids the "too close" problem and the approach uses a concept called "path relaxation" which is applied to the solution path. It has the effect of easing the zigzags and gives a better approximation of the true shortest path. In generating solution paths, this path planner takes into account the cost of exploring unmapped regions.

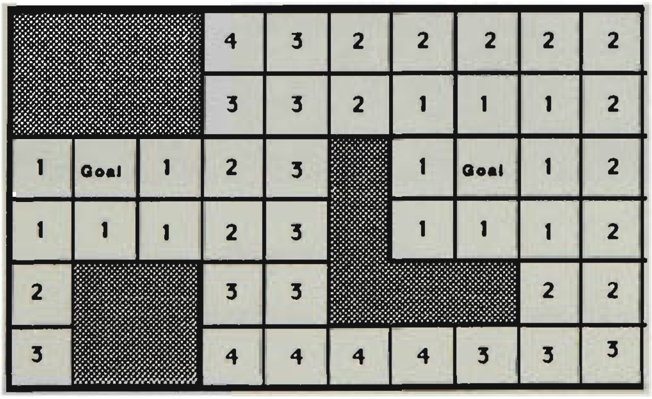
A novel approach to path planning for mobile robots using distance transforms was first presented by [Jarvis *et. al.* 86]. This approach considers the task of path planning to be finding paths from the goal location back to the start location. This path planner procedure propagates distances through free space grid cells from the goal cell (cells are assumed to be 8 connected). The distance wave front flows around obstacles and eventually through all free space in the environment. For any starting point within the environment representing the initial position of the mobile robot, the shortest path to the goal is traced by walking down hill via the steepest descent path. If there is no downhill path, and the start cell is on a plateau then it can be concluded that there is no path from the start cell to the goal cell i.e. the goal is unreachable. Initially all the cells are initialised to high values. Refer to Figure 3.4 for an example of the distance transform.



▨	▨	▨	4	3	2	2	2	2	2
▨	▨	▨	4	3	2	1	1	1	2
7	6	5	4	3	▨	1	Goal	1	2
7	6	5	4	4	▨	1	1	1	2
7	▨	▨	5	5	▨	▨	▨	2	2
8	▨	▨	6	6	5	4	3	3	3

Figure 3.4
Distance Transform Path Planning.

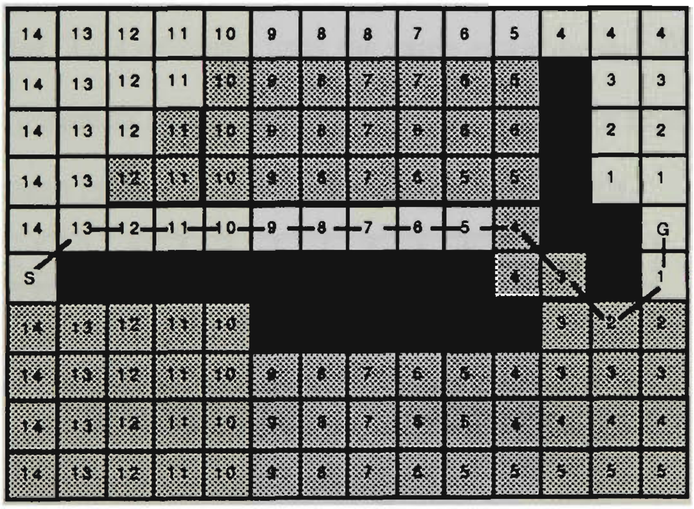
Despite the high computational overhead, distance transform path planning offers several advantages.These include the fact that the shortest path to the goal is known from all free space grid cells, thus supporting multiple robots. The distance transform also readily supports multiple goals (refer to Figure 3.5). In this case a robot heads towards its nearest goal, similar to a fire evacuation drill where people evacuate a building via the closest fire exit.



▨	▨	▨	4	3	2	2	2	2	2
▨	▨	▨	3	3	2	1	1	1	2
1	Goal	1	2	3	▨	1	Goal	1	2
1	1	1	2	3	▨	1	1	1	2
2	▨	▨	3	3	▨	▨	▨	2	2
3	▨	▨	4	4	4	4	3	3	3

Figure 3.5
Distance Transform Path Planning with multiple goals.

Another significant advantage that distance transform path planning has over other path planning methods is that it can easily be induced to exhibit different types of robot navigation behaviours. [Jarvis *et. al.* 86, 88] described how the distance transform could be modified to produce "conservative", "adventurous" and "visit all" path planning behaviours in addition to the "optimum" i.e. shortest path behaviour. [Jarvis *et. al.* 86] used a "factor" function to give a weight to the distance transform depending on whether the grid cell type was known or unknown. If an "optimum" path planning behaviour is required then the same factor is used for both known and unknown cells. Thus neither type of grid cell is favoured, and the distance transform calculates the shortest path to the goal irrespective of whether the path traverses known or unknown regions. Refer to Figure 3.6 for an example of "optimum" distance transform path planning behaviour. In this example the environment consists of three types of grid cells; blocked (shown in black), known free space (shown in white) and unknown free space (shown in grey). The optimum path planning behaviour selects the shortest path from the start (S) to the goal (G) via the free space grid cells without favouring either the known or unknown grid cells.



14	13	12	11	10	9	8	8	7	6	5	4	4	4
14	13	12	11	10	9	8	7	7	6	5		3	3
14	13	12	11	10	9	8	7	6	6	6		2	2
14	13	12	11	10	9	8	7	6	5	5		1	1
14	13	12	11	10	9	8	7	6	5	4			G
S										4	3		1
14	13	12	11	10							3	2	2
14	13	12	11	10	9	8	7	6	5	4	3	3	3
14	13	12	11	10	9	8	7	6	5	4	4	4	4
14	13	12	11	10	9	8	7	6	5	5	5	5	5

Figure 3.6

Distance Transform Optimal Path Planning.

If adventurous behaviour is required then the factor in known cells is doubled, thus causing the robot to favour unknown regions i.e travel in regions which have not previously been visited. Refer to Figure 3.7 for an example of adventurous distance transform path planning behaviour. This example uses the same environment shown in Figure 3.6. The adventurous path planning behaviour selects the shortest path from the start (S) to the goal (G) via the free space grid cells and favours unknown grid cells.

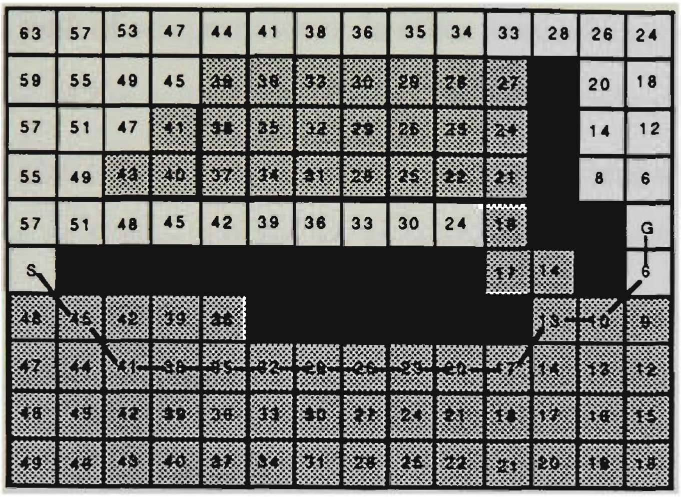


Figure 3.7

Distance Transform Adventurous Path Planning.

If on the other hand a conservative behaviour is sought, which causes the robot to favour known cells i.e. travel in regions which are known, then the factor in unknown cells is doubled. Refer to Figure 3.8 for an example of conservative distance transform path planning behaviour. This example uses the same environment shown in Figure 3.6. The conservative path planning behaviour selects the shortest path from the start (S) to the goal (G) via the free space grid cells and favours known grid cells.

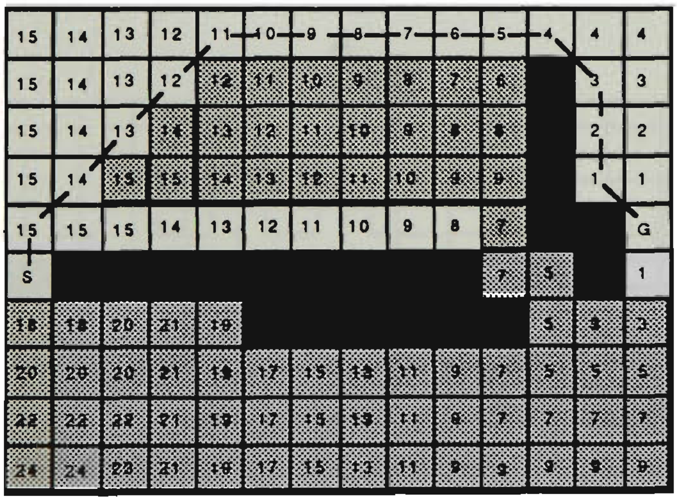
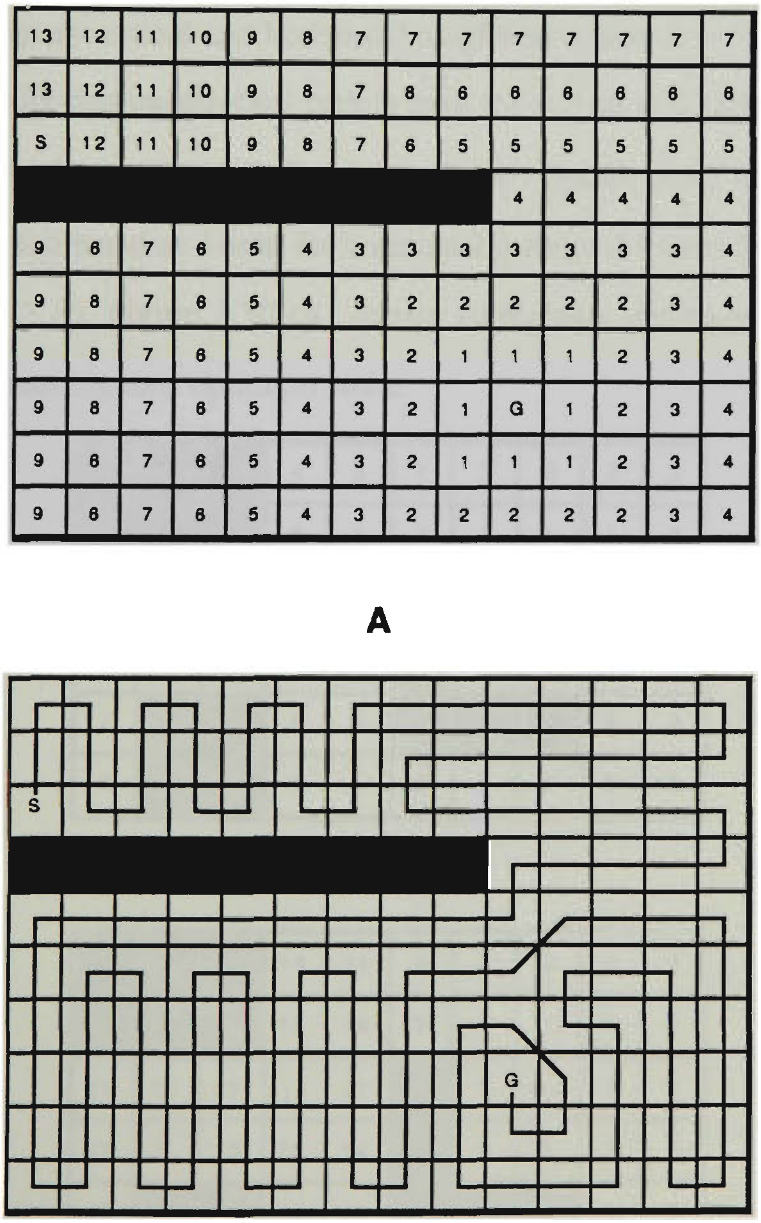


Figure 3.8

Distance Transform Conservative Path Planning.

The experimental results of the "visit all" path planning behaviour were reported in [Jarvis *et. al.* 88], however the algorithm for the behaviour was not published. An algorithm similar to the following was probably used. To achieve the "visit all" path planning behaviour, instead of descending along the path of steepest descent to the goal, the robot follows the path of steepest ascent. In other words the robot moves away from the goal keeping track of the cells it has visited. The robot only moves into a grid cell which is closer to the goal if it has visited all the neighbouring cells which lie further away from the goal. Refer to Figure 3.9 for an example of the "visit all" path planning behaviour. Figure 3.9 (A) shows an environment with one obstacle, start (S) and goal (G) locations and values of the distance transform. Figure 3.9 (B) shows the "visit all" path from S to G.



13	12	11	10	9	8	7	7	7	7	7	7	7	7
13	12	11	10	9	8	7	8	6	6	6	6	6	6
S	12	11	10	9	8	7	6	5	5	5	5	5	5
									4	4	4	4	4
9	6	7	6	5	4	3	3	3	3	3	3	3	4
9	8	7	6	5	4	3	2	2	2	2	2	3	4
9	8	7	6	5	4	3	2	1	1	1	2	3	4
9	8	7	6	5	4	3	2	1	G	1	2	3	4
9	6	7	6	5	4	3	2	1	1	1	2	3	4
9	6	7	6	5	4	3	2	2	2	2	2	3	4

A

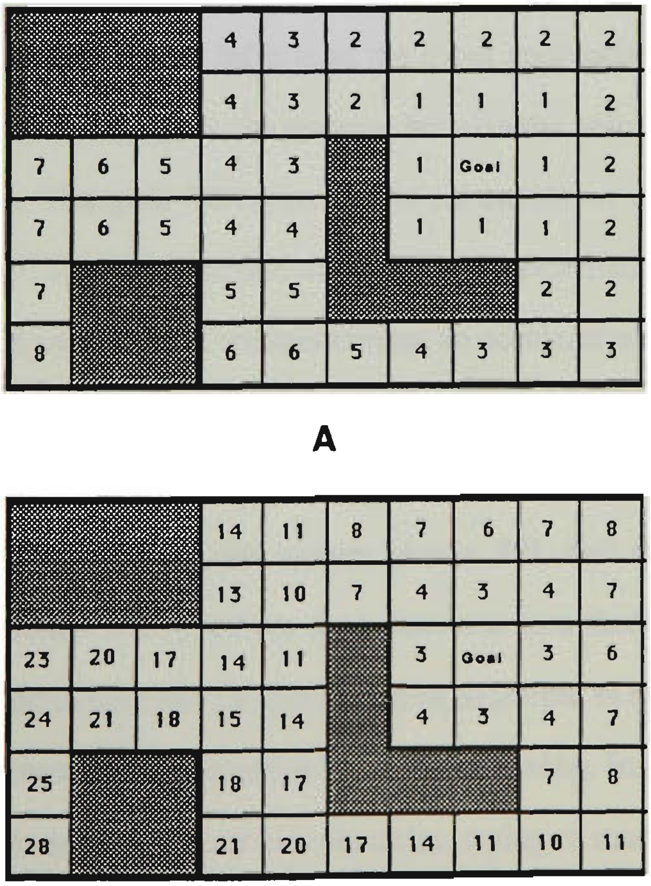
B

Figure 3.9

Distance Transform Visit All Path Planning.

While the [Jarvis *et. al.* 88] strategy does not guarantee the "visit all" path will be an optimum path i.e. the shortest possible and not unnecessarily visiting any cell more than once, the "visit all" produces a reasonable path with minimal secondary visits to grid cells.

The [Jarvis *et. al.* 86] distance transform has a problem with not being able to uniquely specify the shortest path This is caused by considering diagonal neighbours to have the same cost as vertical and horizontal neighbours. However this problem can be overcome by considering the diagonal path to have the correct euclidean distance of √2. It has been shown by [Borgefors 84] that the euclidean distances for distance transforms can be accurately estimated as 3 units for horizontal paths and 4 units for vertical paths, as shown in Figure 3.10. Figure 3.10 (A) shows ambiguous optimum paths and Figure 3.10 (B) shows unambiguous optimum paths.



▨	▨	▨	4	3	2	2	2	2	2
▨	▨	▨	4	3	2	1	1	1	2
7	6	5	4	3	▨	1	Goal	1	2
7	6	5	4	4	▨	1	1	1	2
7	▨	▨	5	5	▨	▨	▨	2	2
8	▨	▨	6	6	5	4	3	3	3

A

▨	▨	▨	14	11	8	7	6	7	8
▨	▨	▨	13	10	7	4	3	4	7
23	20	17	14	11	▨	3	Goal	3	6
24	21	18	15	14	▨	4	3	4	7
25	▨	▨	18	17	▨	▨	▨	7	8
28	▨	▨	21	20	17	14	11	10	11

B

Figure 3.10

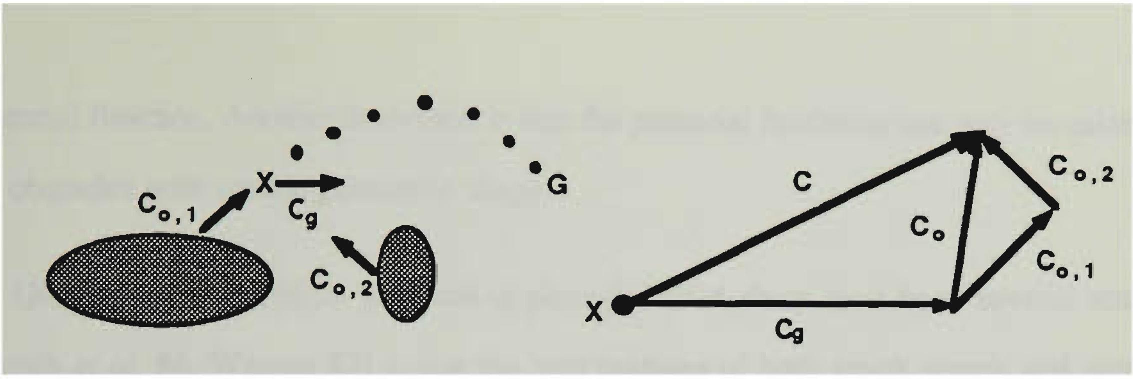
The ambiguity of optimum paths.

All superimposed grid path planning methods suffer from the problem that a grid is inefficient in memory when an environment is largely empty space and contains only a few obstacles.

3.3.4 Potential Field

Potential field path planning [Adams *et. al.* 90, Arkin 89, Khatib 86, Krogh 84] is a method of navigating a robot through an unmapped environment to a goal. This approach attempts to design a real-time path generator which bypasses the computational complexities of other path planning methods. The potential field path planners do not model the environment nor do they build graphs to be searched. Since the environment is not searched globally the solution path is not necessarily the shortest path.

The potential field strategy of navigation is based upon the premise that each obstacle in the environment exerts a repulsion which varies inversely with distance between the robot and the obstacle, and becomes infinite as the robot approaches the obstacle. This force of repulsion depends not only on the position but also the velocity of the robot with respect to the obstacle. The goal however, exerts an attraction upon the robot. The strength and direction of the obstacles and the goal are represented by "avoidance" and "attraction" vectors. The sum of these vectors creates an acceleration vector for the robot to follow. This approach has the advantage that it takes into account the dynamics of the mobile robot when it is generating the solution path [Khatib 86, Krogh 84]. Other approaches ignore the dynamics of the system [Arkin 89], and the potential field is regarded as a cost function. The problem then converts into finding the direction of steepest descent of the potential field i.e. the direction to follow to reach the goal. Refer to Figure 3.11 to for an example of potential field path planning. In this example a robot is located at the location marked X, near two obstacles, trying to reach the goal located at G. The potential field path planner yields the vector C as the direction and magnitude of the robot's acceleration. The dotted path shows the robot's path to the goal.



C_g : Attraction Vector.
$C_{o,1}$; $_{o,2}$: Obstacles in the path Vectors.
Co : Sum of Obstacle Avoidance Vectors.
C : Sum of Attraction and Avoidance vectors, gives the direction of acceleration to the goal.

Figure 3.11

Potential Fields Path Planning.

The major problem with potential field path planners is that they are subject to local minima. Since the planner tends to guide the robot toward lower potential areas, the robot can reach a state of equilibrium, or a potential basin, and becomes trapped. Potential field path planners have problems handling dead end situations, concave obstacles and closely grouped obstacles. A solution to this problem, put forward by [Arkin 89], is to execute a random robot motion, in the hope that the robot will escape the potential minima. [Adams *et. al.* 90] suggests that the goal should be temporarily relocated, when a local minima is detected, and a new potential field to the relocated goal is calculated which may escape the minima. Both the [Adams *et. al.* 90, Arkin 89] approaches are *ad. hoc.* and neither can guarantee that the robot will not remain caught in the local minima.

[Khosla *et.al.* 88] constructed a potential field based on superquadrics to counter the problem of local minima. Using this approach the potential field is less susceptible to local minima, however the problem is not completely eliminated. Constructing potential field without local minima was first reported by [Rimon *et.al.* 88]. The computation effort required to construct the potential field for the [Khosla *et.al.* 88, Rimon *et.al.* 88] approaches can be enormous. Neither of these approaches can easily deal with the obstacles or the goal position being moved, since this requires the recalculation of the potential function. Another drawback is that the potential functions can only be calculated for obstacles with simple geometric shapes.

Given the local minima problem of potential field, there have been several attempts [Krogh *et.al.* 86, Warren 89] to use the best features of both graph search and potential field. These approaches used the graph search solutions for global planning and potential field for local planning. The problem of local minima is not eliminated, but these path planners are less susceptible to local minima than other potential field methods. One drawback of these methods is that the global work space must be known at the time of planning, unlike potential field planners of the past which require no global knowledge.

Another form of path planning which does not require an environment model is heuristic navigation. Heuristic navigation [Cahn *et. al.* 75, Chattergy 85] guides the robot to the goal by using strategies or rules to decide, based on local sensor information, which path of those available is "best". Common heuristic strategies are: minimise the estimated path to the goal or minimise the deviation angle from the path or a linear combination of the previous two strategies. An additional guard can be added to the heuristic to stop the robot from back tracking along the solution path. Heuristic path planners can solve a wide variety of path planning problems, but a problem can always be found where a particular heuristic strategy fails [Cahn *et. al.* 75]. Heuristic path planners like potential field path planners tend to get caught in dead ends.

3.3.5 Path Planning with 3 Degrees of Freedom (3 DOF)

Many path planners do not address the problem that path planning may require rotation as well as translation to negotiate a path to the goal. This problem is called the "piano movers" problem [Schwartz *et. al.* 83]. Refer to Figure 3.12 for an example of path planning with both translation and rotation.

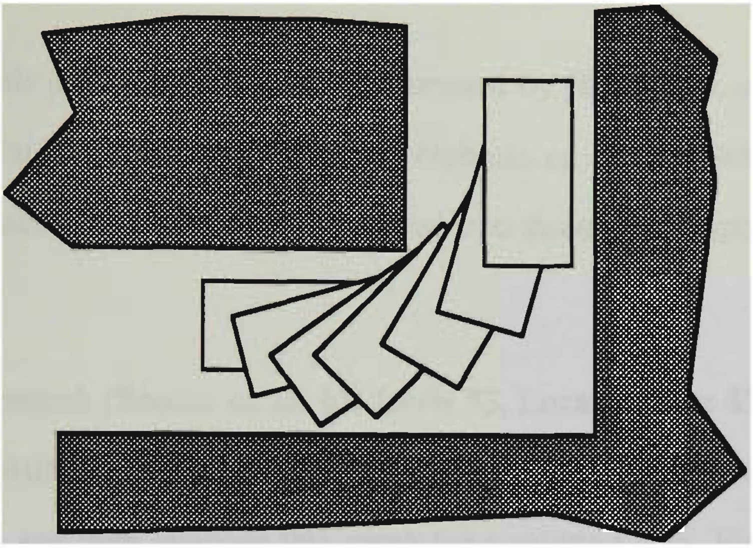


Figure 3.12

Path planning with translation and rotation.

Most path planners ignore the problem of rotation by approximating the robot to be cylindrical. The penalty for this practice is that it excludes potential solution paths, as shown in Figure 3.13. In this example the path between the rectangular shaped robot, and the goal is shown with the heavy line. The obstacles in the environment are drawn as blocks with dark shading. The obstacles which have been expanded by the radius of the cylinder approximating the robot are lightly shaded. The path between the obstacles has been excluded. In the worst case if there exist only solution paths which require rotation, then no solution to the problem will be found.

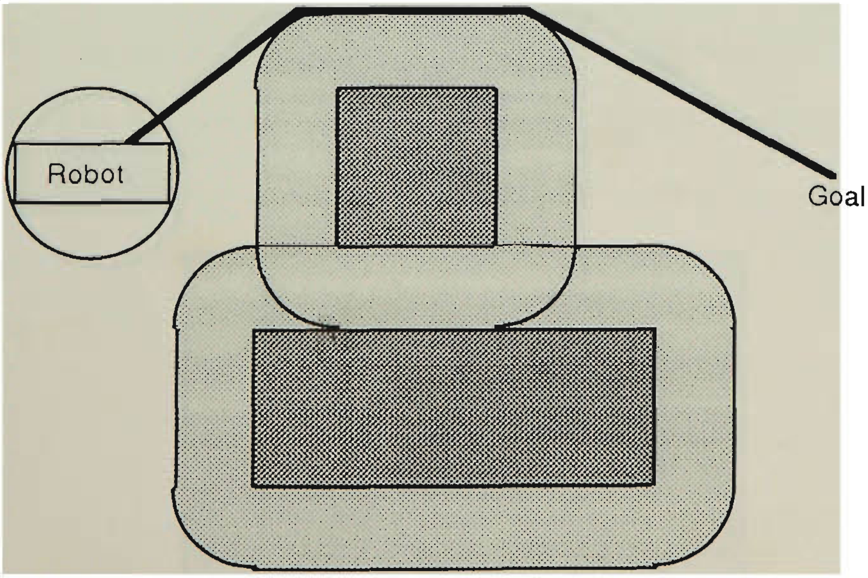


Figure 3.13

Excluding potential solution paths.

Solutions to this problem have been put forward by [Brooks *et. al.* 85, Donald 87, Ilari *et. al.* 90, Jarvis 83, Lozano-Perez 83, Noborio *et. al.* 89, Schwartz *et. al.* 83]. These past approaches can be broadly classified into three groupings; global, local and hybrid methods.

The global approach [Brooks *et. al.* 85, Jarvis 83, Lozano-Perez 83, Schwartz *et. al.* 83] constructs a visibility graph representing all the collision free path configurations of the mobile robot, and then searches this graph for a solution path. Figure 3.14 shows a solution to the problem posed in Figure 3.13. In the case of a 3 DOF robot this graph is a 3 dimensional graph. In practice this 3 dimensional graph is very large in size, since each graph is made up of a collection of 2 dimensional graphs (slices) stacked on top of each other. Each slice represents the visibility graph for the collision free paths for the robot in a particular orientation. It is obvious that such path planners have the drawback of being extremely expensive computationally, and thus are not practical for real mobile robot applications.

The local approach [Donald 87] does not construct a graph representation of all the collision free configurations of the mobile robot. Rather this approach places a grid over the configuration space. The grid is searched for a solution path using heuristics to guide the search. The search heuristics are generated from the information about the geometry of the local robot configuration space. A heuristic guided search does not require any preprocessing, and therefore runs much faster than global planning methods. However as stated previously heuristic path planners are susceptible to failure.

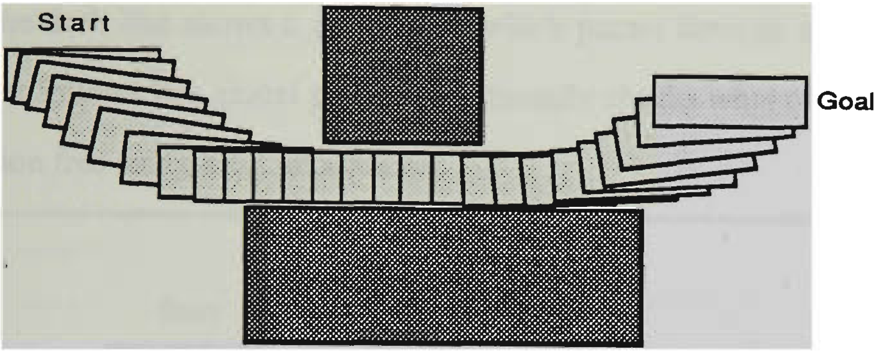


Figure 3.14.

Path planning for a rectangular shaped robot with 3 DOF.

The hybrid approach is a compromise between global and local approaches [Ilari *et. al.* 90, Noborio *et. al.* 89]. Both these approaches compute a coarse path for the robot which is likely to yield a 3 DOF path. This coarse path is then searched for a fine path using a set of heuristics.

The [Ilari *et. al.* 90] approach assumes that the collision free paths for a mobile robot with 3 DOF are likely to lie in the middle of free space between obstacles. Initially the environment model is preprocessed to find all the global paths which exist. To this network of global paths the start and goal locations are added. Refer to Figure 3.15 for an example of the [Ilari *et. al.* 90] path planner. In this figure the dark line shows the global path which lies in the middle of the free space between obstacles. The robot follows this global path and continually checks using local information what orientations of the robot are collision free.

The [Noborio *et. al.* 89] approach uses a quadtree as the model of the environment. The quadtree is searched for a ~~course~~ coarse path of free space quadrants which the minimum width of the robot can negotiate. This coarse path is then searched for a fine path using heuristics. Since the [Noborio *et. al.* 89] path planning technique is based on heuristic search it can fail, although the probability of this occurring is lower than for other heuristic path planning methods. Refer to Figure 3.16 for an example of this path planner.

In this figure the dark line shows a global path which passes through quadrants of free space. The robot follows this global path and continually checks what orientations of the robot are collision free using local information.

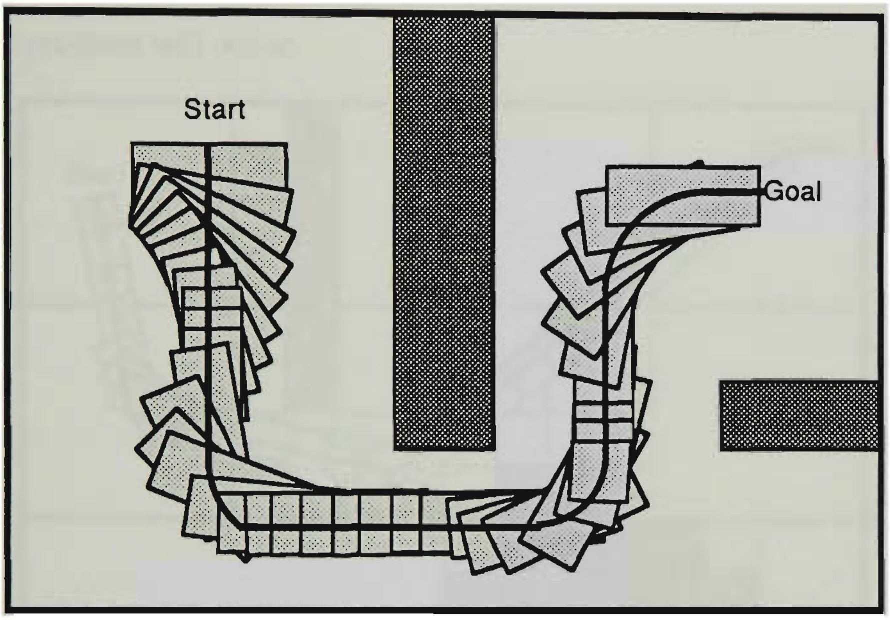


Figure 3.15

Ilari *et. al.* 3 DOF path planner.

Like vertex graph planners, 3 DOF path planners [Brooks *et. al.* 85, Lozano-Perez 83, Schwartz *et. al.* 83] assume that the shortest path is the "best" path, and hence suffer the "too close" problem. Not one of these path planners takes information about clearance from obstacles into account during path generation. The [Ilari *et. al.* 90] approach can select solution paths which maximise the clearance of the path from obstacles while simultaneously minimising the length of the path to the goal. However since this approach is based on selecting paths through the middle of free space, this path planner suffers from the "too far" problem. The [Noborio *et. al.* 89] path planner does not take clearance from obstacles into account. The planner searches for a negotiable path for the minimum width of the robot while at the same time minimising the path length to the goal. This results in the planner selecting a path only through the free space quadrants that can accommodate the minimum width of the robot. The effect of this strategy is that the [Noborio *et. al.* 89] path planner will avoid the smaller sized free space quadrants which are generally located in close proximity to the boundaries of obstacles. Therefore the likelihood of this path planner suffering the "too close" problem is reduced. However it cannot be guaranteed that the [Noborio *et. al.* 89] path planner will not suffer the "too close" problem. The structure of the quadtree environment model will determine whether the "too close" problem will occur.

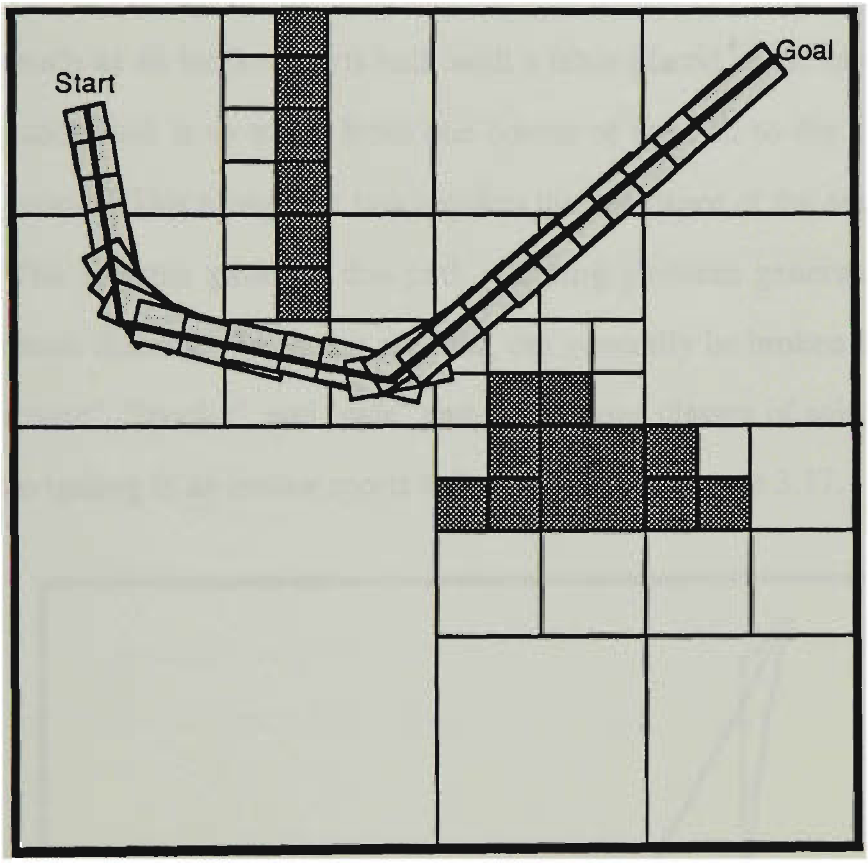


Figure 3.16

Noborio*et. al.* 3 DOF path planner.

3.3.6 Path Planning with Consideration of Robot Safety

It is obvious from the review of past research into mobile robot path planning discussed in earlier sections that much of the research, with the exception of path planning using potential field, has concentrated on minimising the travelling distance between the start and goal locations. The shortest path between the start and goal locations may reduce the robot's travelling time and the computational complexity of the path planning. However, the safety of the robot should not be ignored. The safety of the robot becomes important particularly when there are uncertainties in the environment information, such as the exact shape and position of obstacles. This problem is compounded by the uncertainty in the dynamic control of a robot i.e. the precise position of the robot is not always known by the robot's control system. Thus both minimum distance to a goal and safety of the robot need to be considered simultaneously during path planning.

This problem is illustrated by the following example. Consider a large open space environment such as an indoor sports hall, with a table placed in the middle. A mobile robot's navigation task is to travel from one corner of the hall to the corner which is diagonally opposite. This navigation task requires the avoidance of the desk in the middle of the hall. The solution paths to this path planning problem generated by the path planning methods discussed in earlier sections, can generally be broken down into three classes; "too close", "too far" and "safe" paths. The three classes of solution path tȯ the problem of navigating in an indoor sports hall are shown in Figure 3.17.

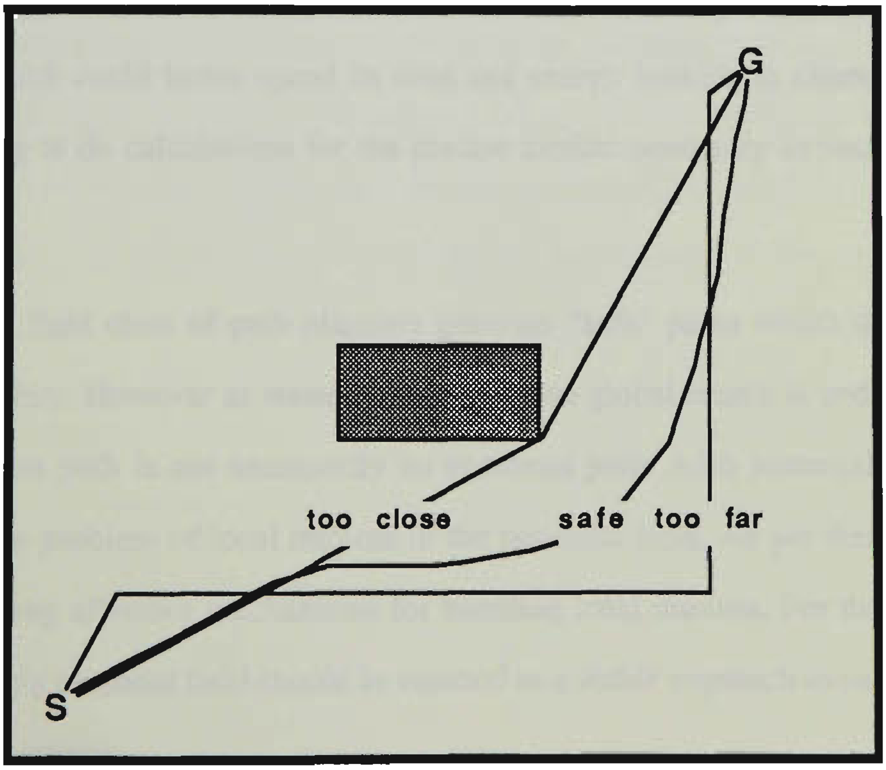


Figure 3.17

Three solution classes to the problem of finding a path between a start (S) and a goal (G).

Path planners which generate path trajectories which are "too close" to obstacles may cause collision with obstacles due to the inaccuracies of a practical mobile robot. As discussed in Section 3.3.1 this problem can be countered by expanding the obstacles by an extra amount to avoid such collisions. This has the effect of adding a safety criteria to the planned motion path. However the penalty for this strategy is that possible solution paths are blocked.

The "too far" class of path planners generate path trajectories which minimise the chance of collision with obstacles. The "too far" path trajectories have the maximum safety from obstacles in the environment. However the penalty for this strategy is that the "too far" path planners can produce unnecessarily long paths. There is no mechanism that allows the control of the degree of safety from obstacles.

The "too close" and "too far" path planning approaches do not take clearance information into account. Both approaches consider the shortest path as best, while there may be a safer path with more clearance. Such information is important for a practical mobile robot which could better spend its time and energy looking at alternative paths rather than trying to do calculations for the precise motion necessary to pass through a narrow corridor.

The potential field class of path planners generate "safe" paths which do take into account robot safety. However as stated earlier, since no global search is undertaken the generated solution path is not necessarily an optimum path. Also potential field path planners have the problem of local minima in the potential field. As yet there have not been developed any effective mechanisms for handling local minima. For these reasons path planning using potential field should be rejected as a viable approach to path planning with robot safety criteria.

Path planning with consideration for robot safety has been reported by [Kambhampati *et. al.* 86, Suh *et. al.* 88, Thorpe 84]. [Thorpe 84] used a grid based approach called "path relaxation" to find the "best" path from the start to the goal. This method is based upon extending the A* algorithm to include three criteria, the cost of the distance from the goal, the cost of nearby objects, and the cost of operating near or in an unmapped environment.

The cost of nearby objects was calculated by searching the grid space for obstacle nodes, calculating a repulsion cost for each obstacle, and finally summing the costs of all the repulsion costs of the nearby objects. This is a heavy computational burden.

A similar approach to [Thorpe 84] was presented by [Kambhampati *et. al.* 86] using a quadtree projected onto the environment instead of a grid. The quadtree is searched using the A* algorithm for the "best" path from the start to the goal. The cost of the path at each quadtree node consisted of two criteria: the cost of the distance from the goal and the cost of nearby objects. The cost of near objects was computed prior to path planning, using a variant of the [Samet 88] distance transform for quadtrees, which is of complexity $O(n)$, where n the number of leaf nodes in the quadtree. While this is an improvement on the [Thorpe 84] approach, since a search at a node for nearby obstacles is not required, the [Kambhampati *et. al.* 86] algorithm can not handle unknown environments. Unlike the [Thorpe 84] path planner the [Kambhampati *et. al.* 86] path planner does not produce a trajectory path which a robot can readily execute. It generates only a coarse path which consists of a chain of free space quadrants which join the start to the goal. For this planning method to be of any use for a practical robot this chain of free space quadrants must be searched for a fine motion path.

[Suh *et. al.* 88] presented a method based on variational calculus, in which the cost for robot safety is considered explicitly in path planning. This approach is based upon decomposing the environment into free space channels, and finding the centre line of these channels, in other words the "too far" paths. [Suh *et. al.* 88] defined the cost of a path P as the sum of costs for its length and its safety. The safety component of the cost is a function of the integration of the distance between a point on the path P and the "too far" path. However this algorithm, like all free space path planning methods, does not give any solution if the environment is not delimited by boundaries and can be decomposed into free space channels.

3.4 Data Structures

The data structures that have been used in path planning research can be broadly classified into two groupings; "adaptive" models and "rigid" models. Adaptive models are high level descriptions, and the structure of these models is dependent upon the nature and clutter of the environment. Typically such models represent the environment as a network of free space regions [Chatila 82, Chatila *et. al.* 85, Crowley 85, Iyengar *et. al.* 86, Rao *et. al.* 86] or as a graph of obstacle vertices [Moravec 80, Thompson 77]. The adaptive model environment mapping methods offer elegant solutions; however these methods require accurate sensor information and are therefore difficult to implement in practice.

On the other hand rigid models impose a structure, typically a grid, onto the environment without any regard to the nature and clutter of the environment [Elfes 87, Jarvis *et. al.* 86, Thorpe 84] and so the implementation of such environment mapping methods is easier than adaptive models. The disadvantage of the rigid model approach is the inefficiency in memory usage to represent large areas of free space.

The quadtree data structure [Samet 88] is a good compromise between adaptive models and rigid models. Since quadtrees use a hierarchical structure, they have the advantage of having a grid like structure, but they are also adaptive to the clutter of the environment. Refer to Figure 3.18 for an example of a quadtree environment model. In this figure the nodes of the tree are labelled numerically, starting from the South Western corner in a clockwise direction. Obstacles are shaded areas, while free spaces are white space.

Both [Kambhampati *et. al.* 86] and [Samet 88] comment on the savings in memory resulting from the use of a quadtree representation compared to a grid based representation. [Kambhampati *et. al.* 86] shows that the number of leaf nodes in a quadtree is proportional to the sum of the perimeters of the obstacles in the environment.

[Samet 88] reports that if the resolution of a quadtree is doubled, then the number of nodes in the quadtree will double, while with grids, doubling the resolution quadruples the grid size. Mobile robot path planning using quadtrees has been reported by [Kambhampati *et. al.* 86, Noborio *et. al.* 88]. Both of these approaches find paths only in known environments.

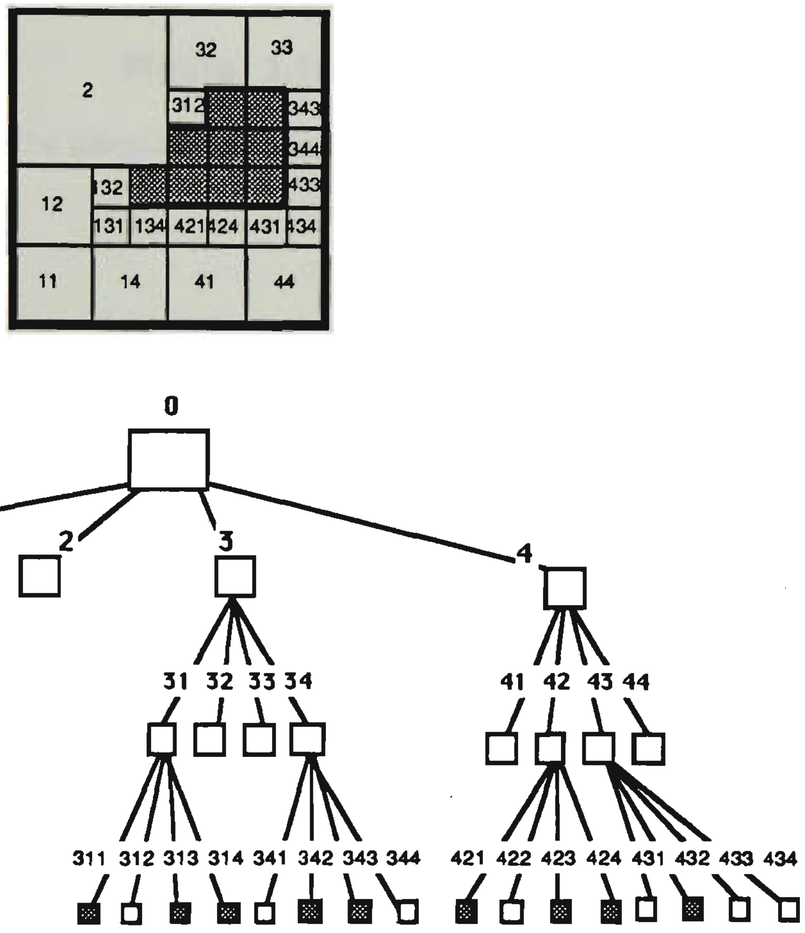


Figure 3.18

A two dimensional environment and its corresponding quadtree.

The quadtree data structure is implemented in the following manner. Each node of the quadtree has storage for the pointers to the node's parent and its four children. Storage is also provided for the node classification; free, obstacle or grey (neither free nor obstacle). Storage within the leaf node is provided for the path cost which is generated during path planning. Refer to Figure 3.19 for a diagram of the node structure.

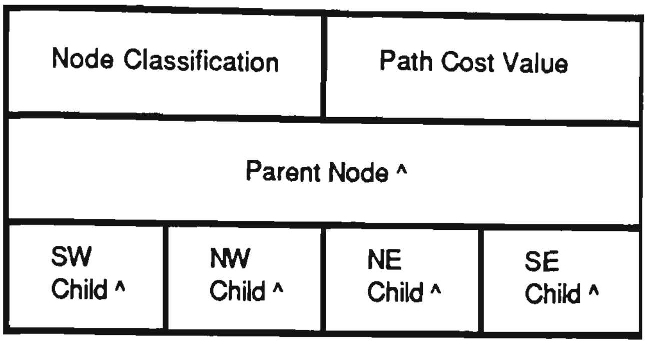


Figure 3.19

The structure of a quadtree node.

3.5 Conclusions

Section 3.2 of this chapter discussed the desirable features of a robot path planner from a human stand point. In Section 3.3 a extensive review of past research into path planning was undertaken. In that section the shortcomings and the benefits of each path planning method were discussed. This section links the conclusions that were reached in earlier sections with the research goals of this thesis to form a list of research questions which require investigation.

To support the research goals stated in Chapter 1 distance transforms were chosen as the methodology to be investigated for path planning. Unlike other path planning methods, distance transforms support the concept of path planning behaviours. However the main drawbacks of using distance transforms with the grid data structure, are the inefficiency of grids when the environment is largely free space, and the zigzag nature of the solution paths. Also, there are no memory savings using grids when a mobile robot is operating in an unknown environment whose structure must be mapped, because the robot must model all the space in the environment whether it is known or unknown. In such situations, an adaptive model of the environment, such as vertex graph or free space methods would be more appropriate. To build an adaptive model using the noisy sonar data extracted in Chapter 2 is a complicated task. It is therefore proposed to investigate quadtrees as the compromise data structure to model the environment.

Quadtrees were explored for the following reasons. Firstly they can be a more efficient data structure than rigid models such as grids. Secondly quadtrees are a much more effective data structure to support the inclusion of noisy sonar data than adaptive models. Thirdly past research into path planning with quadtrees has assumed that the environments are known. However quadtrees could offer a way of supporting path planning in unknown and partially known environments, due to the recursive nature of this data structure. The recursive nature of the quadtree seems to support the strategy humans use to navigate to a goal in an unknown environment. Chapter 4 investigates path planning in unknown and partially known environments using the quadtree data structure and distance transforms. That chapter also explores how noisy sonar data can be incorporated into the quadtree model as the robot learns the environment en route to the goal.

One problem that must be overcome using distance transforms for path planning with quadtrees is that the distance transform will only yield a ~~course~~ coarse path of free space quadrants joining the start and goal positions, thus producing similar results to the [Kambhampati *et. al.* 86] algorithm. The problem of extracting a fine robot motion path from a coarse solution path necessitates further investigation. The potential spin off from solving this problem is that the zigzag path problem in grids can be overcome. Fine paths in quadtrees will have less points in the path, longer straight sections and less zigzags. In Chapter 4 this problem is scrutinised and a solution proposed.

Applying the distance transform to quadtrees is a much more expensive approach to path planning than the [Kambhampati *et. al.* 86, Noborio *et. al.* 88] approaches. This is because the distance transform generates paths from every location to the nearest goal, while the [Kambhampati *et. al.* 86, Noborio *et. al.* 88] approaches search only for the shortest path to a goal. This disadvantage is offset by the fact that distance transforms support path planning behaviours of the type described in Section 3.2, multiple robots and multiple goals. Chapter 5 investigates how the path planning behaviours can be implemented with quadtrees.

A drawback of using distance transforms for path planning is that they suffer from the "too close" problem since they do not take obstacle clearance information into account. Chapter 5 shows how the distance transform can be extended to take obstacle clearance information into account. This extension can be regarded as the construction of a potential field function between the start and goal locations which contains no local minima. This extension takes care of the "too close" and "too far" problems which hampers other path planners.

Applying the distance transform to quadtrees in certain situations e.g large areas of free space, can significantly improve the storage efficiency and the execution time of the distance transform, compared to the use of grids as reported by [Jarvis *et. al.* 86]. However it is unclear in exactly which situations the quadtrees will out perform grids. In robot environments which are of low resolution i.e. contain a small number of cells, the grid will out perform the quadtree, due to the memory overheads of storing the hierarchical data structure. In Chapter 6 a comparative study of distance transforms using grids and quadtrees is presented. This study determines when it is best to use quadtrees instead of grids. The study determines the resolution size which is most economical to represent the environment with quadtrees, and which obstacle shapes, what obstacle sizes, and what degree of obstacle clutter, make the quadtree a more efficient data structure.

As discussed in Section 3.3.5, existing 3 DOF path planning algorithms have heavy computational burdens because of the high cost of building and searching a 3 dimensional graph. In Chapter 7, a new 3 DOF path planning algorithm is presented which only constructs and searches two slices of the 3 dimensional graph. This new algorithm is based upon further extensions to distance transforms.

The key idea of this algorithm is to calculate a distance transform between the start and goal configurations taking into account obstacle clearance information. By taking this information into account the steepest descent path through the distance transform provides a coarse path for the robot to follow. Searching the coarse path for a solution path gives the robot a higher chance of finding a 3 DOF fine path. A solution path is most likely not to lie in close proximity to obstacles. Existing 3 DOF path planners waste valuable time searching for paths in close proximity to obstacles, and as a result suffer the "too close" problem. The [Ilari *et. al.* 90] approach solved this problem, but it suffers from the "too far" problem. The "too close" and "too far" problems are taken care of by the new 3 DOF algorithm.

The new algorithm treats the values of the distance transform as guiding heuristics. However this algorithm is guaranteed to find a solution if one exists, unlike other heuristic based 3 DOF path planners. Since this algorithm has a heuristic nature it does not have the computational burden which is associated with other 3 DOF path planners.

Chapter 4
Path Planning

4.1 Introduction

Exploring an environment with a mobile robot can be accomplished in one of two ways; either by operating in "mapping" or "learning" modes. When a robot is operating in "mapping" mode [Crowley 85, Lumelsky *et. al.* 89, Moravec 80], it traverses the entire environment in a systematic manner, while scanning with on board sensors and updating a map. The map is then used for all subsequent path planning exercises. Difficulties arise with this method if the environment is allowed to alter after the mapping has been completed. The other mode of learning is to sense the environment, while executing paths which have been generated by a path planner. As obstacles are encountered en route to a goal, the model of the environment is updated and a new path to the goal is planned to avoid the obstacles [Iyengar *et. al.* 86, Rao *et. al.* 86].

This chapter describes a new environment exploration algorithm (referred to as "EEA" for short) which is based on the "learning" mode of environment exploration. This algorithm was implemented as part of the research for this thesis. The EEA is not restricted to the recognition of line of sight distances to obstacles, and it can be used with current generation sensing technology. The EEA can be induced to exhibit the "mapping" or "visit all" path planning behaviour. This feature is discussed in Chapter 5.

Unlike most other path planners the EEA does not expand all the objects in the environment by the robot's radius. Therefore it is the responsibility of the path execution procedure to ensure that the robot does not collide with any obstacles. This is achieved by making sure the robot stays entirely within the boundaries of the free space quadrant which the robot is traversing. Conceptually this is equivalent to shrinking the robot to a point, and shrinking the boundaries of the quadrant by the robot radius. The quadrants in the quadtree of the lowest resolution are of sufficient size to fully accommodate the mobile robot. This strategy has the penalty of excluding possible solution paths. Since paths are generated through free space nodes of a quadtree which have been reduced, only paths in the horizontal and vertical directions can be considered. Considering paths that pass over the shared corner of adjacent free space quadrants could cause a collision. Diagonal corner paths tend to clip obstacles [Kambhampati *et. al.* 86]. Thus the restriction to horizontal and vertical paths will result in safer robot paths. However the penalty to the EEA for this approach is that paths are no longer distance optimum [Kambhampati *et. al.* 86].

The EEA assumes that the robot accurately knows its own current world coordinates at all times, as well as the coordinates of the goal. The robot has no knowledge about the location and shape of obstacles. No constraints are imposed upon the shape or location of the obstacles.

The remainder of this chapter is organised in the following manner. Section 4.2 provides an overview of the design EEA. This section describes how the EEA decomposes into a set of modules, each module has a specific task. Sections 4.3 - 4.8 in this chapter explain the workings of each module of the EEA. The EEA algorithm is based on the use of distance transforms.Section 4.3 describes an efficient algorithm to generate distance transforms for quadtrees. The distance transform yields only a coarse solution path, which consists of a chain of free space quadrants between the start and goal positions. Section 4.4 presents a procedure to extract the coarse solution path, and shows how the shortest path through the free space quadrants can be extracted from the coarse solution path. Section 4.5 describes a procedure for the inclusion of fresh sensor data into the quadtree model. Section 4.6 describes a mechanism to speed up the generation of the distance transform. Examples of a robot navigation system using the EEA are presented in Section 4.7. These results were obtained using a simulation program developed on a Macintosh II microcomputer. Section 4.8 presents the experimental results using the sonar range data collected in Chapter 2. Finally Section 4.9 presents the conclusions that were reached and the insights that were gained from investigating the problem of environment exploration.

4.2 Environment Exploration Algorithm

Upon initialisation the EEA requires two pieces of information; the notional size m x n of the environment to be learnt, and a position reference (x,y) of the cylindrical robot to some reference point. A quadtree Q of sufficient size is generated to cover the area in which the robot will operate. The smallest quadtree leaf resolution size is of diameter d, a size which allows the robot to pass through. The size of quadtree $Q = (2d)^i$, where i is an integer such that $(2d)^i > s$ and $s = max\,(m, n)$.

The EEA is composed of a number of processes. Figure 4.1 shows the architecture of the processes which constitute the EEA. The main process in the EEA is the Navigation process. This process is responsible for three subprocesses; Path Planning, Path Execution and Model Update.

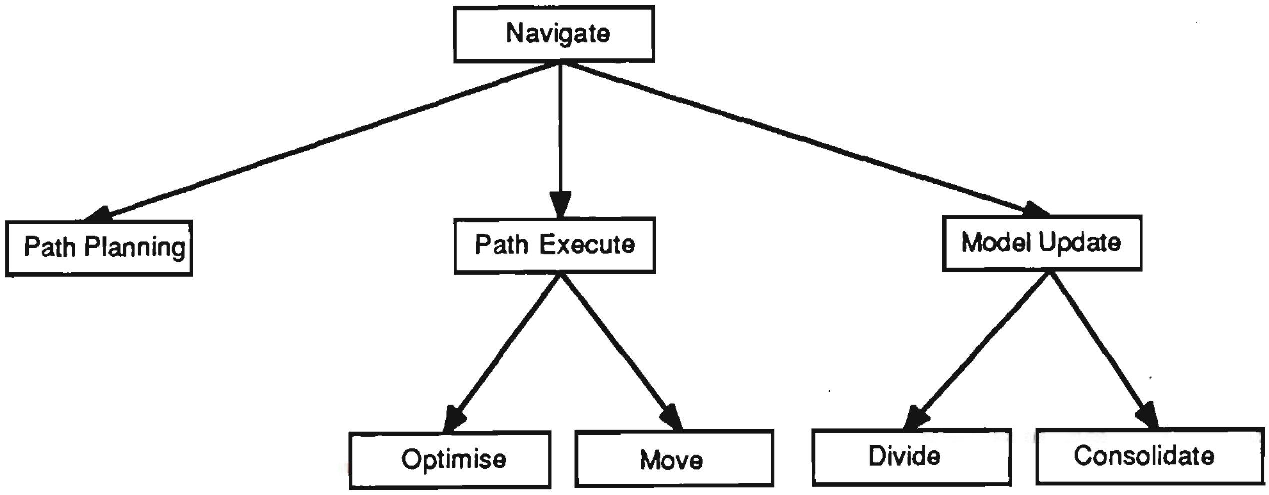


Figure 4.1
Process Architecture of the EEA.

The Path Planning process, when supplied a goal location applies the distance transform to the quadtree model of the environment, and produces all the possible solution paths from any location in the environment to the goal. The exact workings of this process are explained in Section 4.3.

The Path Execution process locates the start location in the quadtree, and then traces a coarse least cost path through the free space quadrants to the goal. The subprocess

Optimise is used to find the shortest path through the coarse solution path. Once the shortest path to the goal has been planned, the robot executes this path. The Move process controls the motion of the robot and the sensors on board the robot. Sensing the environment is the responsibility of the Path Execution process and is done within the Move process. The EEA is an algorithm which only senses for obstacles which obstruct the robot's path to a goal. The Path Execute process is described in detail in Section 4.4.

The Model Update process is invoked when the Path Execute process detects an obstacle which is not present in the quadtree model of the environment. The Divide process has the responsibility of recursively subdividing the quadtree until leaves are obtained which span regions that are entirely free space or obstacle filled. The fresh sensor data is incorporated into the quadtree. The Consolidate module is responsible for merging the leaves which are of the same node classification and share a common parent. The workings of the Model Update process are explained in Section 4.5.

Algorithm 4.1 shows how the three main processes which constitute the Navigation process fit together and interact.

```
procedure NAVIGATION( Q, start, goal )
  repeat
    cost = 0
    perform PATH_PLANNING( Q, goal, cost )
    if ( goal reachable ) then
      perform PATH_EXECUTE( Q, start, goal, stop, sensors )
      if ( stop ≠ goal ) then
        perform MODEL_UPDATE( Q, start, stop, sensors )
        start = stop
      end if
    end if
  until ( stop = goal or goal not reachable )
end procedure
```

Algorithm 4.1

Component Processes of the EEA.

Given the cartesian coordinates of the start and goal locations, the best path planning strategy in an unknown environment is to optimistically assume that the unknown regions of the environment are free space. The confidence in this assumption is low. The NAVIGATION process invokes the PATH_PLANNING process, which plans a straight line path to the goal and calculates the distance transform cost between the start and goal locations.

If the path cost returned by the path planner is finite, then a solution path exists between the start and goal locations; otherwise the goal is unreachable. If the goal is reachable, the path generated by the planning process is passed onto the PATH_EXECUTE process, which ensures the planned path is executed by the robot. Since the environment is unknown, the robot proceeds cautiously towards the goal.

One of two conditions will occur; either the robot reaches the goal or it encounters an obstacle. If the goal is attained the PATH_EXECUTE process reports a success to the NAVIGATION procedure. In the case of an obstacle blocking the robot's path, the PATH_EXECUTE process returns the location of the robot and the robot's sensor readings at this location, to the NAVIGATION process.

The NAVIGATION process invokes the MODEL_UPDATE process, which given the robot's location and sensor readings updates the environment model Q i.e. the quadtree structure. Upon completion of the updating of the environment model, the PATH_PLANNING process is invoked again.

Essentially the PATH_PLANNING process locates the leaves of the quadtree where the current location and goal are found and then applies the distance transform to generate a solution path, or deduces that no solution path exists. The revised plan is then attempted by the robot. This cycle of plan - execute - update continues until the robot successfully reaches the goal, or the NAVIGATION process deduces that the goal is unreachable.

4.3 Path Planning using Distance Transforms

When the distance transform is applied to the quadtree structure, distances are propagated through the quadtree from the goal quadrant leaf to the neighbouring quadrant leaves, which in turn propagate the distance transform to their neighbouring leaves. This process is continued until the distance transform flows through the the whole quadtree.

The distance transform is measured in multiples of the minimum sized quadrant. Except in the case of the quadrant leaf containing the goal, the distance transform is calculated as the straight line distance from the goal location to the nearest edge of neighbouring leaves (note: the goal can be located anywhere within the goal quadrant). This measure identifies the neighbouring quadrants which are closest to the goal. Refer to Figure 4.2 for a result of applying this algorithm. Once the distance transform has been computed, the shortest path to closest goal is known for every quadrant of free space. Note: The distance transform values stored in all the quadrants are initialised to ∞ (in practice this is the largest available integer) before the path planner is invoked.

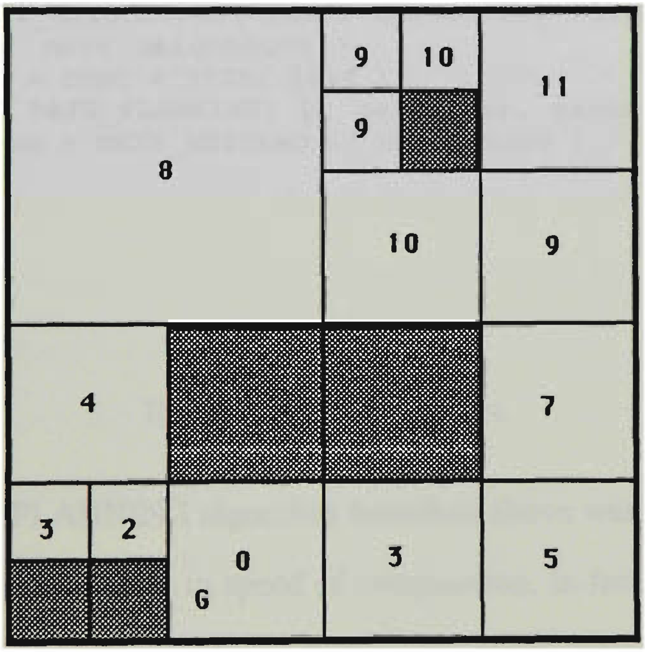


Figure 4.2

The distance transform applied to a quadtree.

In this research the first algorithm which was developed computed the distance transform based on the concept described in the previous paragraph i.e. one of radiating the paths from the goal. The algorithm is given in the procedure PATH_PLANNING shown in Algorithm 4.2. In this algorithm the procedure GET_NEIGHBOURS finds the neighbouring leaves to the current leaf in a given direction, and returns a list of neighbours. If more than one neighbour exists then they are accessed in EAST-WEST or NORTH-SOUTH order, using the NEXT_NEIGHBOUR function.

GET_NEIGHBOURS is based on Samet's neighbour finding algorithm [Samet 82]. The function EXTRACT retrieves the distance transform value of a quadrant leaf from the quadtree data structure. The STORE function records a new distance transform value for a quadtree leaf in the quadtree data structure. The function SIZE returns the size of a quadtree node. The returned size is a multiple of the size of the smallest allowable quadrant in the quadtree.

```
procedure PATH_PLANNING( Q, leaf, cost )
   minimum = EXTRACT( leaf )
   if ( cost < minimum ) then
    perform STORE( leaf, cost )
    for direction = EAST, WEST, NORTH and SOUTH do
      perform GET_NEIGHBOURS( leaf, direction, neighbours )
      do while ( more neighbours )
         newcost = cost + SIZE( leaf )
         perform PATH_PLANNING( Q, neighbour, newcost )
         neighbour = NEXT_NEIGHBOUR( neighbours )
      end do
    end for
   end if
end procedure
```

Algorithm 4.2

The path planning algorithm.

When the PATH_PLANNING algorithm described above was implemented, it was found to be extremely inefficient in speed of computation. In fact the performance was inferior to the [Jarvis *et. al.* 86] grid based distance transform. This is due to the recursive nature of the algorithm, which propagates the distance transform from the goal in a selected direction along a path until a boundary of the quadtree is reached or an obstacle is encountered. At this point the algorithm backtracks one quadrant along the propagated path and then selects another direction to propagate the distance transform further into the quadtree. Thus this algorithm propagates the distance transform in a "spike" fashion instead of the preferred "wave front" fashion. The effect of this strategy is that most nodes in the quadtree are visited many times, and a significant portion of these visits are unnecessary.

To overcome this problem a new algorithm was developed which is described in the procedure FAST_PATH shown in Algorithm 4.3. This algorithm is based on the concept of alternatively sweeping the distance transform from the North Western (NW) and South Eastern (SE) corners of the quadtree. In the first sweep from the NW corner, the distance transform is propagated only in the eastern and southern directions. Once the distance transform sweep from the NW corner has covered the entire quadtree, another sweep of the distance transform is then undertaken. This time the distance transform sweeps from the SE corner of the quadtree back to the NW corner of the quadtree, and the distance transform is propagated only in the western and northern directions. This procedure of alternatively sweeping the distance transform from the opposite corners of the quadtree is repeated until the distance transform values of the free space quadrants stabilise i.e. stop changing. Refer to Figure 4.3 which illustrates with an example how the distance transform is propagated using the alternate corner sweeps algorithm. Figure 4.3 (A - F) shows the propagation of the distance transform from the NW corner of the quadtree. Figure 4.3 (G - L) shows the propagation of the distance transform from the SE corner of the quadtree. After only two sweeps the distance transform has almost converged to its correct final values. One more sweep from each corner will yield the correct distance transform values.

```
procedure FAST_PATH( Q, goal, cost )
    leafg = LOCATE( goal )
    perform STORE( leafg, cost )
    repeat
       change = FALSE
       perform PATH_NW( Q, change )
       perform PATH_SE( Q, change )
    until ( change = FALSE )
 end  procedure
```

Algorithm 4.3

Faster path planning algorithm.

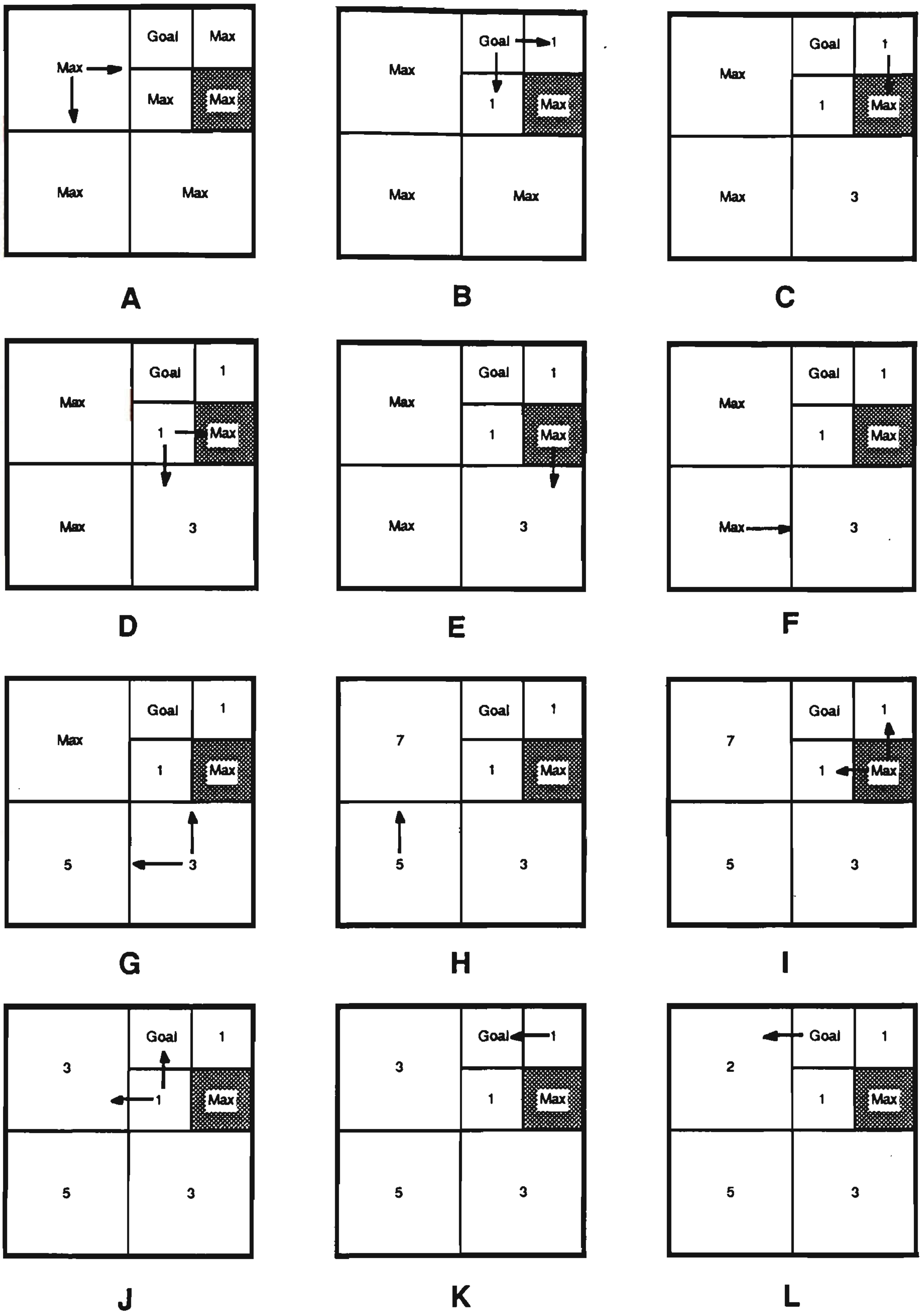


Figure 4.3

Distance transform computed using the alternate corner sweeps algorithm.

When the FAST_PATH algorithm was implemented it was found to be significantly faster in speed of computation than the PATH_PLANNING algorithm. In some cases the new algorithm out performed the old by a factor of 20. Consequently all the results reported in this thesis use the FAST_PATH algorithm. In the pseudo code explanation of FAST_PATH, the function LOCATE is used to determine in which free space leaf the goal is located. The procedures PATH_NW and PATH_SE perform the alternate corner sweeps of the distance transform. The pseudo code for PATH_NW and PATH_SE is almost identical, and is shown in algorithms 4.4 and 4.5 respectively. In both PATH_NW and PATH_SE, the functions GREY and WHITE are used to determine the type of a quadtree node. A node is GREY if the node is an interior node of the quadtree i.e. not a leaf node. A node is WHITE if it is a free space leaf node. The function NEXT_DIRECTION generates the new direction in which the distance transform is to be propagated. The new direction is generated in clockwise order, so the next direction after NORTH is EAST, and the next direction after EAST is SOUTH etc.

```
procedure PATH_NW( Q, change )
   if ( GREY( Q ) ) then
      for quadrant = NW, NE, SW and SE do
         perform PATH_NW( SON(Q, quadrant) )
      end for
   else if ( WHITE( Q ) ) then
      direction = EAST
      do while ( direction != WEST )
         perform GET_NEIGHBOURS( Q, direction, neighbours )
         do while ( more neighbours  )
            cost = SIZE( neighbour ) + EXTRACT( Q )
            minimum = EXTRACT( neighbour )
            if ( cost < minimum ) then
               change = TRUE
               perform STORE( neighbour, cost )
            end if
            neighbour = NEXT_NEIGHBOUR( neighbours )
         end do
         direction = NEXT_DIRECTION( direction )
      end do
   end if
end procedure
```

Algorithm 4.4

Propagate the distance transform the NE corner algorithm.

The PATH_NW algorithm works in the following manner. Firstly, the procedure finds the north western quadrant which is white i.e. free space and the distance transform is propagated to quadrants on the southern and western boundaries of this quadrant. Next, the north eastern quadrant is found and the the distance transform is propagated to quadrants on the southern boundary of this quadrant. Lastly, the south western quadrant is found and the the distance transform is propagated to quadrants on the eastern boundary of this quadrant. If any of the quadrants which are visited are grey i.e. interior quadtree nodes, then the PATH_NW procedure recursively calls itself, until a leaf quadtree node is found i.e. black or white. The PATH_SE algorithm works in a similar manner as PATH_NW and therefore its workings will not be explained.

```
procedure PATH_SE( Q, change )
   if ( GREY( Q ) ) then
      for quadrant = SE, SW, NE and NW do
         perform PATH_SE( SON(Q, quadrant) )
      end for
   else if ( WHITE( Q ) ) then
      direction = WEST
      do while ( direction != EAST )
         perform GET_NEIGHBOURS( Q, direction, neighbours )
         do while ( more neighbours  )
            cost = SIZE( neighbour ) + EXTRACT( Q )
            minimum = EXTRACT( neighbour )
            if ( cost < minimum ) then
               change = TRUE
               perform STORE( neighbour, cost )
            end if
            neighbour = NEXT_NEIGHBOUR( neighbours )
         end do
         direction = NEXT_DIRECTION( direction )
      end do
   end if
end procedure
```

Algorithm 4.5

Propagate the distance transform the SW corner algorithm.

4.4 Path Execute Algorithm

Upon the completion of path planning, the NAVIGATION process is ready to execute the planned path. The general algorithm for path execution is given in the procedure PATH_EXECUTE shown in Algorithm 4.6. Before the robot can be moved to the goal location, a path must be selected using the distance transform information stored in the quadtree. The algorithm must first isolate the location of the robot into a leaf of the quadtree Q. The distance transform stored in this leaf is examined. If the distance transform equals zero, then the start and goal points are located in the same leaf; therefore the robot can proceed directly to the goal. If the distance transform is greater than zero, the vertical or horizontal neighbouring leaf with the minimum distance transform must be found. A subgoal point in the minimum valued distance transform quadrant is selected, and the robot proceeds to this subgoal. The robot will either reach the subgoal or it will encounter an obstacle, which is not stored in the map. If the robot reaches the subgoal, the next subgoal is found and the path to this subgoal is attempted. This method is applied repeatedly until the goal is attained or an obstacle blocks the robot's path. If an obstacle is sensed the robot stops. The path execution is terminated and the robot location and the sonar sensor values are returned to the NAVIGATION process.

```
procedure PATH_EXECUTE( Q, start, goal, stop, sensors )
   obstacle = FALSE
   repeat
      leaf = LOCATE( Q, start )
      distance_transform = EXTRACT( leaf )
      if ( distance_transform = 0 ) then
         subgoal  = goal
      else
         subgoal = OPTIMISE( Q, leaf,goal )
      end if
      perform MOVE( subgoal, stop, sensors )
      if ( stop ≠ subgoal ) then
            obstacle = TRUE
      end if
      start = subgoal
   until ( start = goal or obstacle )
end procedure
```

Algorithm 4.6

Path Execution algorithm.

The MOVE procedure is responsible for interfacing the NAVIGATION process to the motion control system of the mobile robot. This procedure ensures that the robot physically reaches the planned subgoal. If an obstacle is encountered en route to the subgoal, the MOVE procedure returns the location where the robot stopped and a sonar map of the obstacle. The sonar map is generated using the mapping techniques presented in Chapter 2.

The function OPTIMISE is responsible for selecting the robot's next subgoal. This function searches the vertical and horizontal neighbours of the quadrant leaf in which the robot is located, to find the leaf quadrant which yields the least cost path. Finding a neighbouring quadrant with the minimum distance transform does not guarantee a least cost path, since the distance transform stored in a quadrant represents the cost of traversing the quadrant from a boundary edge to the goal. To determine the true cost of the path to the goal the path cost between the robot's location and the boundary edge of the neighbour quadrant must be calculated and added to the quadrant's distance transform. Once the free space quadrant with the least cost has been found, a path can be executed to this quadrant from the robot's current location.

During path execution a number of strategies can be used to generate a path through the free space quadrants. A reasonable strategy would be to steer through the middle of the intersection of entry and exit boundary edges of quadrants. Such a strategy generates relatively "safe" paths, and is reasonable during the exploration of an environment by the robot. However such a strategy suffers the "too far" problem, in situations where the free space quadrants are large.

An optimum path can be found by constructing a visibility graph, between the robot's location and the goal. This visibility graph can be easily constructed since a coarse path of free space quadrants to the goal is already known. The function FULL given in Algorithm 4.7 dynamically constructs and searches a visibility graph for an optimum solution. An example of the application of the FULL function is shown in Figure 4.4. This is an example of how the shortest path between a start (S) and a goal (G) is found. The path is found by extending the visibility graph into the next free space quadrant. This is done by joining the end points of the intersection edge between the two quadrants, to the closest nodes of the visibility graph. The ADD_TO_PATH function is responsible for extending the visibility graph. Figure 4.4 (A) - (B) show how the visibility graph is extended. The NEXT_QUADRANT function is responsible for finding the next free space quadrant in the solution path to the goal. The FIND_INTERSECTION function finds the overlapping segment which is shared by the current quadrant and the next quadrant in the solution path. The STRAIGHTEN_PATH function straightens paths in the visibility graph. In this function, checks are undertaken to see whether or not the nodes that have been freshly added to the visibility graph, can bypass any of the parent nodes of the new nodes. To allow a parent node to be bypassed, a clear path must be visible between the new node and its grandparent node. The clear path must lie inside the boundaries of the free space quadrants which form the coarse solution path. Figure 4.4 (C) shows paths being straightened. The ADD_TO_PATH extends the visibility graph by adding new nodes to the closest end points of the current visibility graph. This practice can introduce redundant paths into the visibility graph i.e. paths which lead to nowhere. The DELETE_PATH function is responsible for removing any redundant paths from the visibility graph. Figure 4.4 (D) shows the deletion of redundant paths. Once the visibility graph between the robot's location and the goal has been constructed, as shown in Figure 4.4 (E), the graph is searched for the shortest path. The SHORTEST_PATH function checks path lengths in the visibility graph and deletes all paths except the shortest path, refer to Figure 4.4 (F).

```
function FULL( Q, start, goal )
   path = NIL
   path = ADD_TO_PATH( path, start )
   leafs = LOCATE( Q, start )
   repeat
      leafn = NEXT_QUADRANT( Q, leafs )
      distance_transform = EXTRACT( leafn )
      if ( distance_transform = 0 ) then
         path = ADD_TO_PATH( path, goal )
      else
         edge = FIND_INTERSECTION( leafs, leafn )
         path = ADD_TO_PATH( path, BEGIN( edge ) )
         path = ADD_TO_PATH( path, END( edge ) )
      end if
      path = STRAIGHTEN_PATH( path )
      path = DELETE_PATH( path )
      leafs = leafn
   until ( distance_transform = 0 )
   path = SHORTEST_PATH( path )
   return ( path )
end function
```

Algorithm 4.7

Full Optimum Path algorithm.

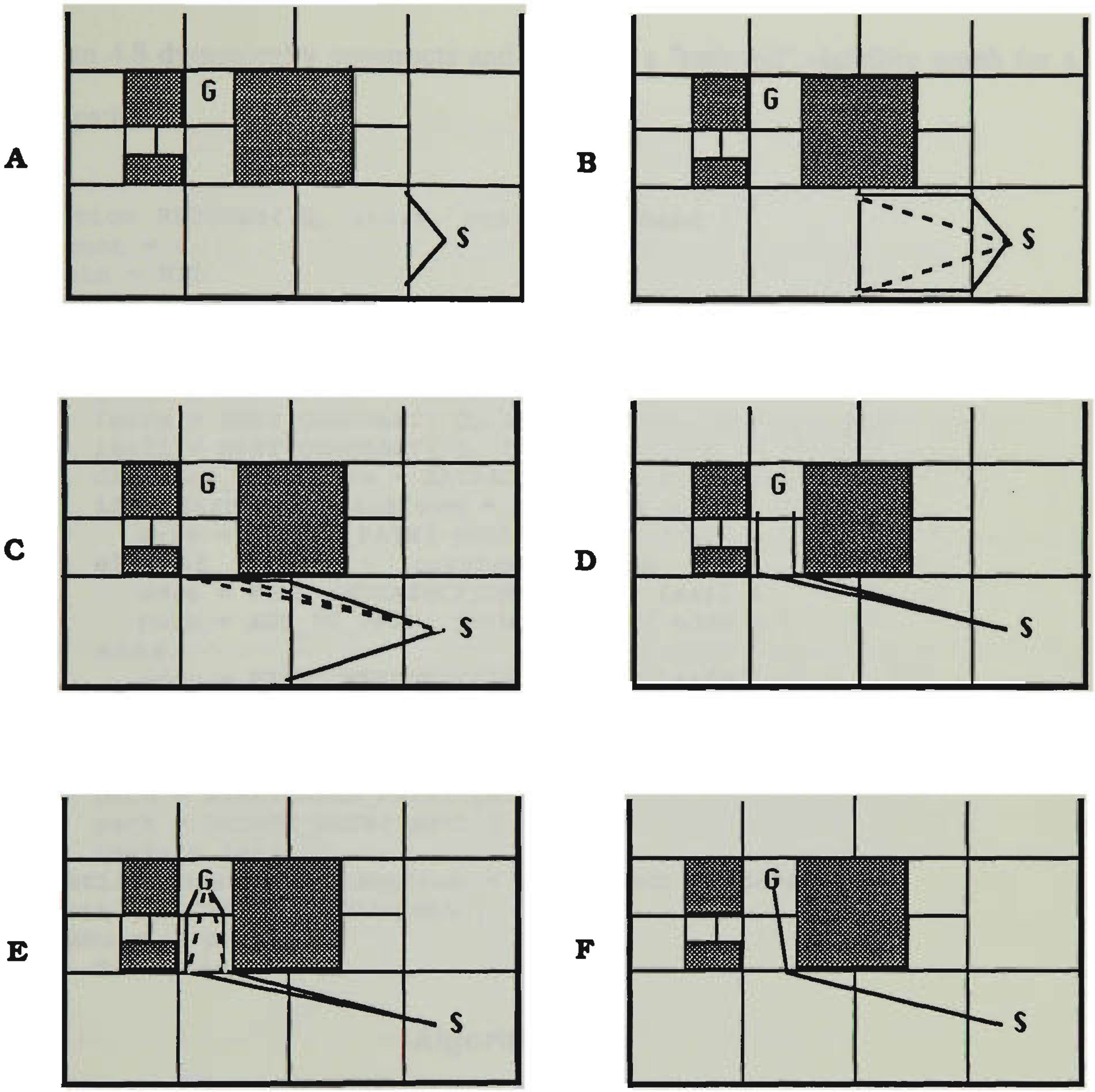


Figure 4.4

Finding the full optimum path between S and G.

The work effort of computing the visibility graph is likely to be wasted in an environment which is not well known, since an unexpected obstacle will result in path replanning. A solution to this problem is as follows. Instead of constructing the "full" visibility graph, look ahead *n* quadrants and construct a "reduced" visibility graph between the robot's location and the mid point of the exit edge of the *nth* quadrant (except in the case of the goal quadrant where the goal takes the place of the midpoint of the exit edge). The "reduced" visibility graph approach will not always yield the optimum path, but it will give a path which is near optimum. The function REDUCED given in

Algorithm 4.8 dynamically constructs and searches a "reduced" visibility graph for a solution path.

```
function REDUCED( Q, start, goal, lookahead )
  count = 0
  path = NIL
  path = ADD_TO_PATH( path, start )
  leafs = LOCATE( Q, start )
  repeat
     count = count + 1
     leafn = NEXT_QUADRANT( Q, leafs )
     leafl = NEXT_QUADRANT( Q, leafn )
     distance_transform = EXTRACT( leafn )
     if ( distance_transform = 0 ) then
        path = ADD_TO_PATH( path, goal )
     else if ( count = lookahead ) then
        edge = FIND_INTERSECTION( leafn, leafl )
        path = ADD_TO_PATH( path, MIDDLE( edge ) )
     else
        edge = FIND_INTERSECTION( leafs, leafn )
        path = ADD_TO_PATH( path, BEGIN( edge ) )
        path = ADD_TO_PATH( path, END( edge ) )
     end if
     path = STRAIGHTEN_PATH( path )
     path = DELETE_PATH( path )
     leafs = leafn
  until ( distance_transform = 0 or count = lookahead)
  path = SHORTEST_PATH( path )
  return ( path )
end function
```

Algorithm 4.8

Reduced Optimum Path algorithm.

Choosing the size of the look ahead can be related to the confidence in the knowledge of the environment. In a completely unknown environment a one quadrant look ahead is sufficient. Once an environment is well known, the optimisation of the paths can be improved by looking ahead more than one quadrant.

Figure 4.5 shows an example of constructing an optimum path using a "reduced" visibility graph. The optimum path which is generated by the construction and search of a "full" visibility graph is shown with the broken line in Figure 4.5 (F). The "reduced" visibility graph is constructed by looking ahead a number of quadrants to the middle of the exit edge of the look ahead quadrant. The number of quadrants that is looked ahead in this example is 1 (one). Figure 4.5 (A) - (E) shows the construction of "reduced" visibility graphs. A new visibility graph is constructed each time the robot enters the next quadrant. This strategy produces a reasonable path, shown with the heavy line in Figure 4.5 (F). The look ahead is performed at the locations marked with *. While this path is not optimum, it is superior to the "too far" paths, which are generated by steering down the middle of free space quadrants.

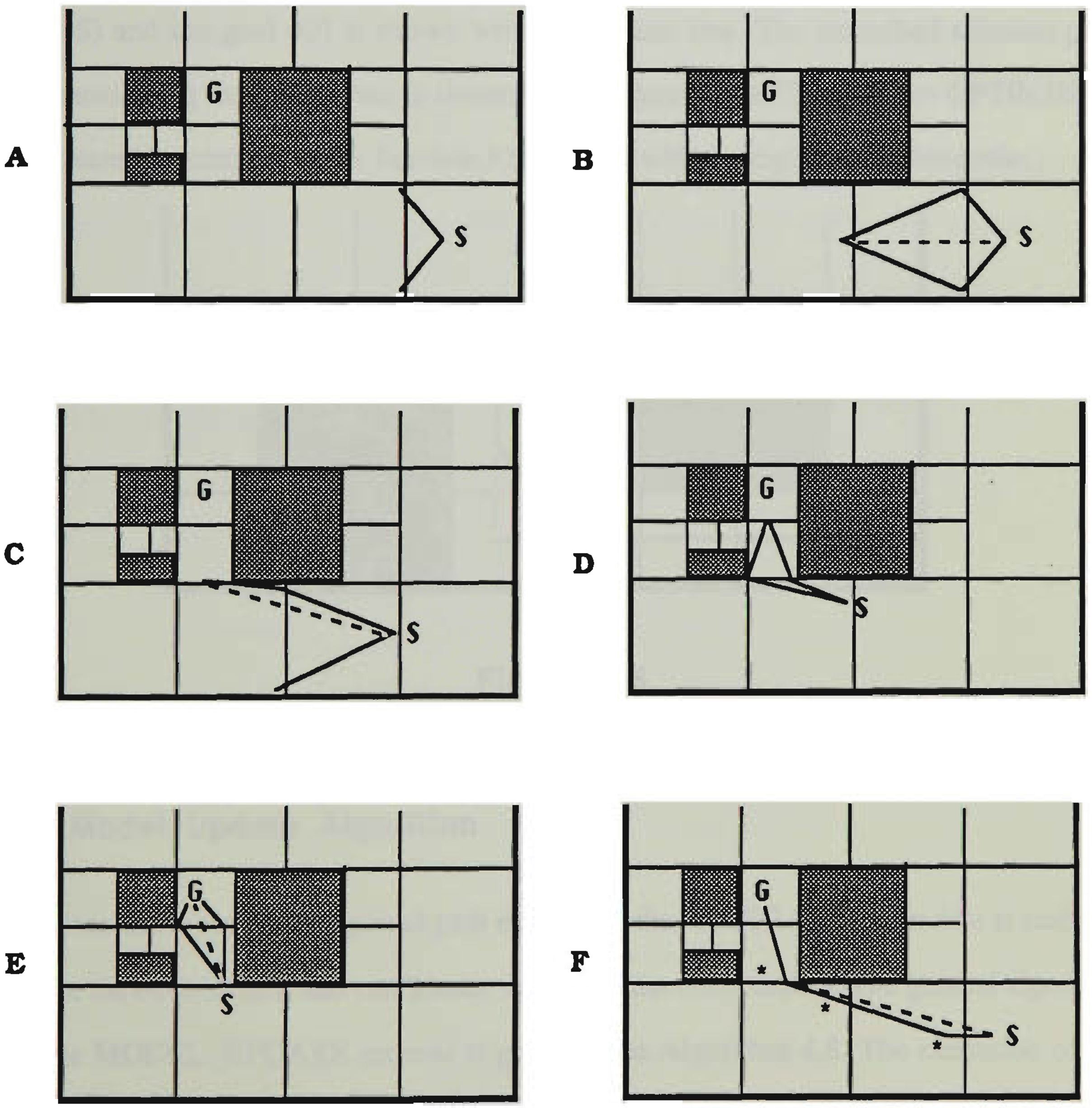


Figure 4.5

Finding the reduced optimum path between S and G.

This path optimisation strategy produces reasonable paths. However situations do arise where the robot must steer through a sequence of quadrants which result in zigzag paths despite the optimisation strategy. This is because paths between quadrants sharing a corner are not considered. This can be countered with the rule: "if the entry and exit edges of a free space quadrant are normal to one another, and the neighbouring corner quadrant is free space, then the path segment through this quadrant can be omitted". This causes the robot to steer diagonally between two free space quadrants. Refer to Figure 4.6 for an example of the zigzag rule. In this example the original solution path between the start (S) and the goal (G) is shown with the broken line. The smoothed solution path generated using the zigzag rule is shown with the heavy line.The function OPTIMISE is constructed by combining the function REDUCED with the zigzag smoothing rule.

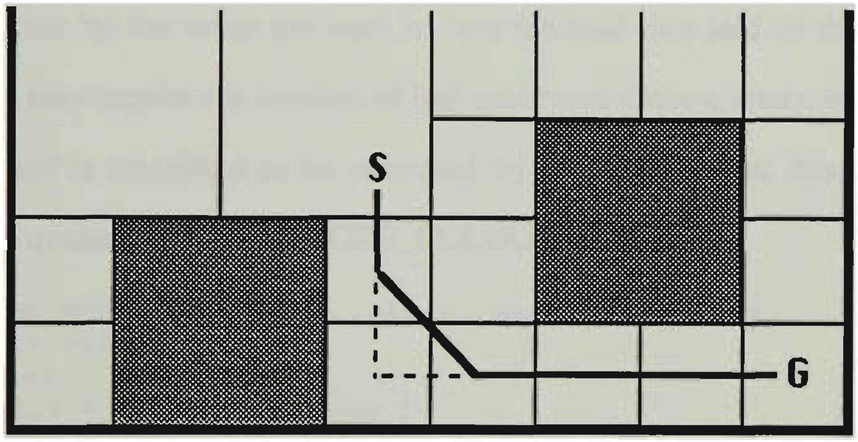


Figure 4.6

Smoothing a Solution path using the zigzag rule.

4.5 Model Update Algorithm

Once the robot has completed path execution, the NAVIGATION module is ready to update the environment and confidence models of the environment. The general algorithm for the MODEL_UPDATE process is given in the Algorithm 4.8. The execution of the planned path terminates on one of two conditions; either the robot reaches the goal or an obstacle is encountered. If the robot reaches the goal, the environment model does not change. If an obstacle is encountered the environment model must be updated to reflect the presence of the freshly sensed obstacle. To perform the update of the model, the algorithm must know the current location of the robot, the sensor readings at this location, and the location of the robot when this procedure was last invoked.

Once the algorithm updating the environment models is invoked, the current leaf location of the robot in the quadtree is found with the LOCATE function. This leaf is checked using the SAFE function to make sure that none of the obstacle sensor readings are inside the leaf quadrant. If the sensor readings occur inside the leaf quadrant in which the robot is currently located, then the leaf quadrant is divided into four quadrants using the DIVIDE procedure. Leaf division continues until the leaf is isolated from the sensor readings, or the size of the leaf reaches the smallest allowable resolution. Upon the completion of isolating the current robot leaf location from the sensor readings, the sensor readings detected by the robot are each in turn isolated to a leaf of the smallest size resolution; this may require the division of leaf quadrants. Once a sensor reading has been isolated, the leaf is classified to be occupied by an obstacle and this information is recorded in the quadtree using the STORE_COLOUR procedure.

```
procedure MODEL_UPDATE( Q, start, stop, sensors )
    exit = FALSE
    repeat
        leaf = LOCATE( Q, stop )
        if ( SAFE( leaf, sensors )  then
            perform STORE_COLOUR( leaf, WHITE )
            exit = TRUE
        else if ( leaf is smallest resolution )  then
            colour = CLASSIFY( leaf, sensors )
            perform STORE_COLOUR( leaf, colour )
            exit = TRUE
        else
            perform STORE_COLOUR( leaf, GREY )
            perform DIVIDE( leaf, child )
        end if
    until ( exit )
    for ( i = 1 to number of sensor readings ) do
        exit = FALSE
        repeat
            leaf = LOCATE( Q, sensor[i] )
            if ( leaf is smallest resolution )  then
                perform STORE_COLOUR( leaf, BLACK )
                exit = TRUE
            else
                perform STORE_COLOUR( leaf, GREY )
                perform DIVIDE( leaf, child )
            end if
        until ( exit )
    end for
    perform CONSOLIDATE( Q )
end procedure
```

Algorithm 4.8

Model Update algorithm.

Isolating a sensor reading to the smallest resolution leaf may be seen as unnecessarily fragmenting the leaves of the quadtree which span obstacles. The CONSOLIDATE procedure detects the neighbouring leaves in a quadtree that are part of the same obstacle, and prunes these leaves back to their parent node. If the path planning procedure deduces that a goal is not reachable from any direction this implies that the goal is surrounded by obstacles. The CONSOLIDATE procedure can then prune the quadtree, to reflect the knowledge that the region surrounding the goal is one obstacle. Refer to Figure 4.7 for an example of the consolidation algorithm. Figure 4.7 (A) shows the environment prior to consolidation. The letter F marks quadrants that are free space, and the letter B marks quadrants that are blocked with obstacles. Figure 4.7 (B) shows the same environment after consolidation of the quadtree.

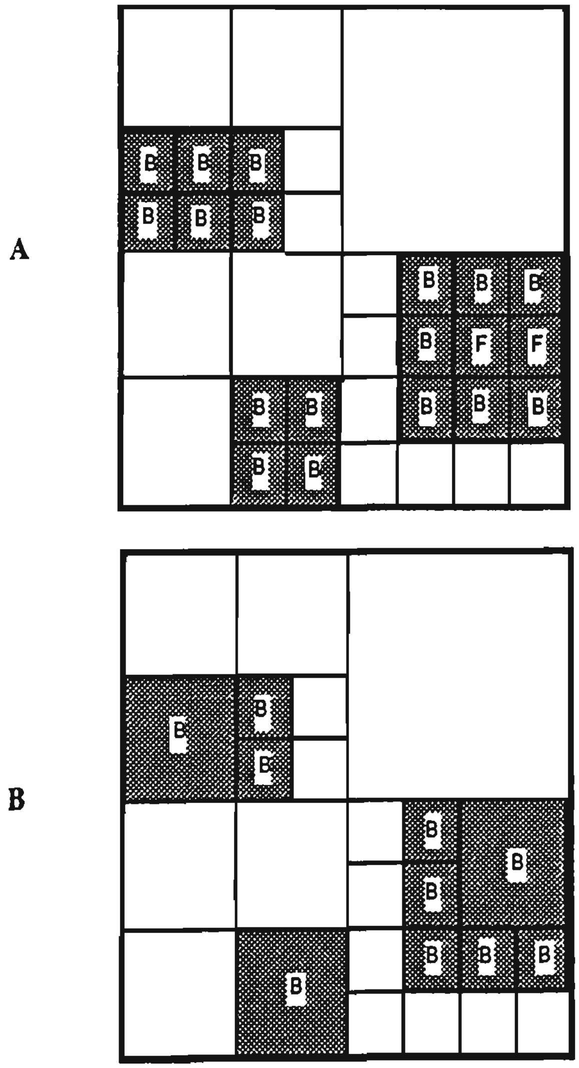


Figure 4.7

The Consolidation algorithm at work .

4.6 Partial Distance Transform Algorithm

Computing the full distance transform every time the robot encounters an obstacle is an unacceptable computational burden. Considerable savings can be made if the distance transform is only partially updated when the environment model changes. A novel method is presented here which allows the quadtree to be efficiently used to limit the recomputation of the distance transform.

If the robot start and goal locations can be found in the same minimal subtree of the quadtree, the distance transform need only be calculated for the subtree. Considering a subset of the possible solution paths will yield either locally optimum paths or no solution paths. Finding locally optimum paths is acceptable when the robot is learning an environment. To find globally optimum paths, or if no solution paths can be found locally, the algorithm simply moves up one level in the quadtree structure and computes the distance transform for the fresh subtree. Refer to Figure 4.8 for an example of the partial distance transform in partially known environments. Known portions of obstacles are shaded dark, while unknown portions are shaded lightly. Figure 4.8 (A) shows the distance transform values for an entire quadtree. In this case a large portion of the distance transform is redundant since the start (S) and the goal (G) are close to each other. If S and G can be isolated to a common subtree, then the distance transform need only be calculated for this subtree as shown in Figure 4.8 (B).

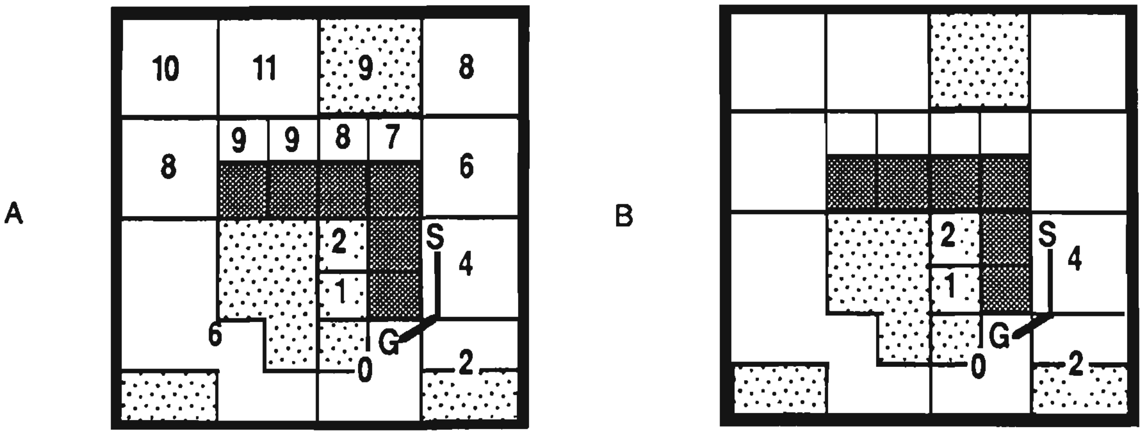
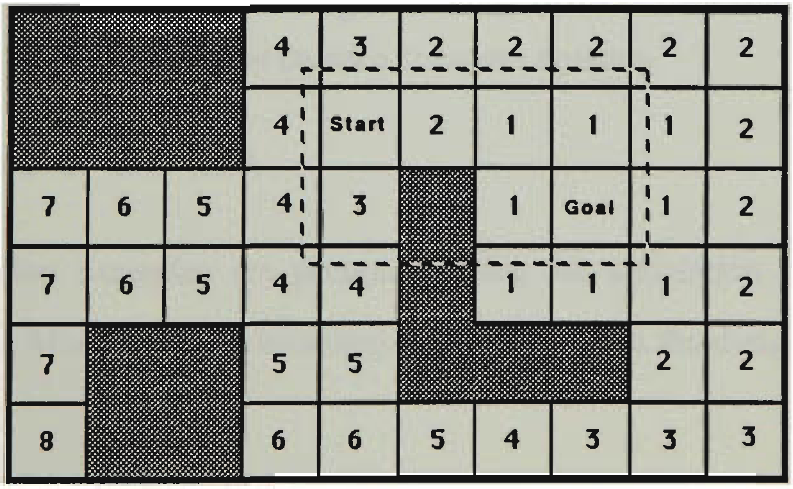


Figure 4.8

Quadtree based Partial Distance Transforms.

Such a strategy is useful when the robot gets closer to the goal. The partial distance transform update mechanism can be easily incorporated into the NAVIGATION process. It should be noted that the penalty for this strategy is that the navigation algorithm no longer supports multiple robots. However for a single robot operating in an unknown or partially unknown environment the partial distance transform offers substantial computational savings.

As an aside, the [Jarvis *et. al.* 86] distance transform based on grids can be modified in a similar fashion to take advantage of partial distance transforms. In this case the start and goal locations are isolated to a common subgrid. The distance transform is then calculated for this subgrid. Refer to Figure 4.9 for an example of the partial distance transform based on grids. This figure shows the distance transform values for an entire grid. In this case a large portion of the distance transform is redundant since the start (S) and the goal (G) are close to each other. If S and G can be isolated to a common subgrid, then the distance transform need only be calculated for this subgrid. The common subgrid in this figure is highlighted by the broken line box .



▨	▨	▨	4	3	2	2	2	2	2
▨	▨	▨	4	Start	2	1	1	1	2
7	6	5	4	3	▨	1	Goal	1	2
7	6	5	4	4	▨	1	1	1	2
7	▨	▨	5	5	▨	▨	▨	2	2
8	▨	▨	6	6	5	4	3	3	3

Figure 4.9

Grid based Partial Distance Transforms.

A modified NAVIGATION process which incorporates the partial distance transform is described by the procedure NAVIGATION_2 and is given in Algorithm 4.9. The first step this algorithm performs is to find the two quadtree leaves which contain the start and goal locations. In the next step the algorithm finds the common subtree of the two quadrant leaves containing start and goal. This is done by the procedure ISOLATE. The steps of path planning, path execution and model updating are performed in the same manner they were performed in the NAVIGATION process. The difference is that if the algorithm deduces that a goal is unreachable, it attempts to move up one level in the quadtree and recomputes the distance transform.

```
procedure NAVIGATION_2( Q, start, goal )
   SQ  = ISOLATE ( Q, LOCATE ( Q, start ), LOCATE ( Q, goal ) )
   repeat
      perform PATH_PLANNING ( SQ, goal )
      if ( goal reachable ) then
         perform PATH_EXECUTE ( SQ, start, goal, location, sensors )
         if ( location ≠ goal ) then
            perform MODEL_UPDATE ( SQ, start, location, sensors )
            start = location
            SQ = ISOLATE ( SQ, LOCATE( SQ, start ), LOCATE( SQ, goal )
         end if
      else
         SQ = PARENT ( Q, SQ )
         if ( SQ = nil ) then
            goal not reachable
      end if
   until ( location = goal or goal not reachable )
end procedure
```

Algorithm 4.9

Partial Distance Transform algorithm.

4.7 Examples of the EEA

In this section examples are presented using the simulation program that was developed on a Macintosh II microcomputer to verify that the design of the EEA was correct.

The first example of the EEA using the full distance transform running on an 8 x 8 pixel map in a completely unknown environment is shown in Figure 4.10. This example shows a robot exploring an environment by planning a path from a start location (S) to a goal location (G) and executing the path. Planned paths are shown as broken lines and actual paths are shown as solid lines. Known portions of obstacles are shaded dark, while unknown portions are shaded lightly. Finally after the robot reaches G, the best known path from S to G is shown.

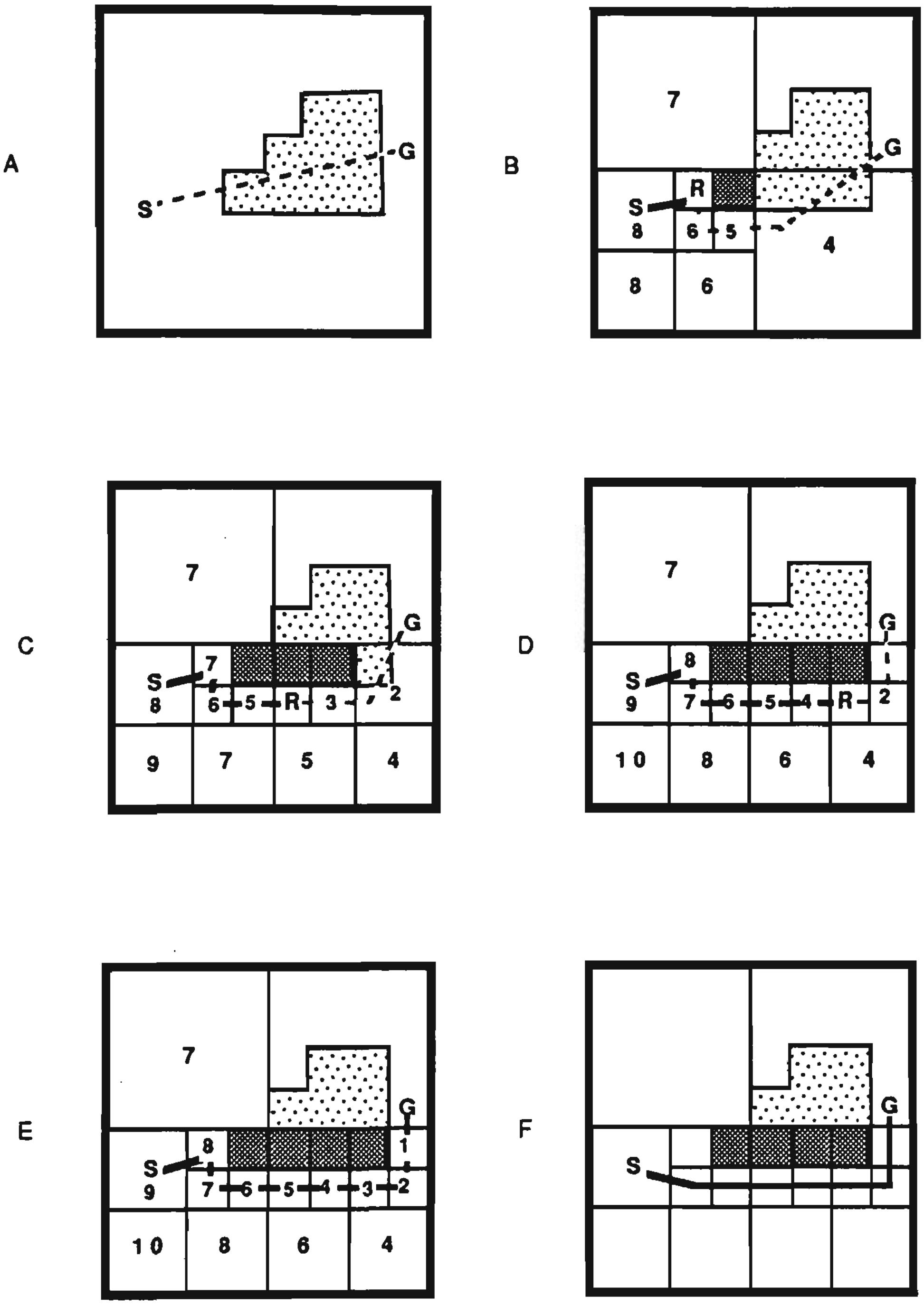


Figure 4.10

Path Planning an unknown environment.

The second example which is presented in Figures 4.11 - 4.16 shows the variety of path execution strategies that can be used to find a path between the start and goal locations. This example uses a full distance transform running on a 32 x 32 pixel map in a completely known environment. Figure 4.11 (A) shows a 512 x 512 pixel map cluttered with obstacles and the start and goal locations. This 512 x 512 pixel map is converted into a quadtree in Figure 4.11 (B). The quadtree is constructed with a leaf resolution of 16 pixels which results in a 32 x 32 pixel quadtree map. The coarse solution path of free space quadrants between the start and goal locations is shown in Figure 4.11 (C). Figure 4.11 (D) shows the execution path which is generated by steering through the middle of the intersection edge of the free space quadrants. This path zigzags due to the nature of the coarse solution path.

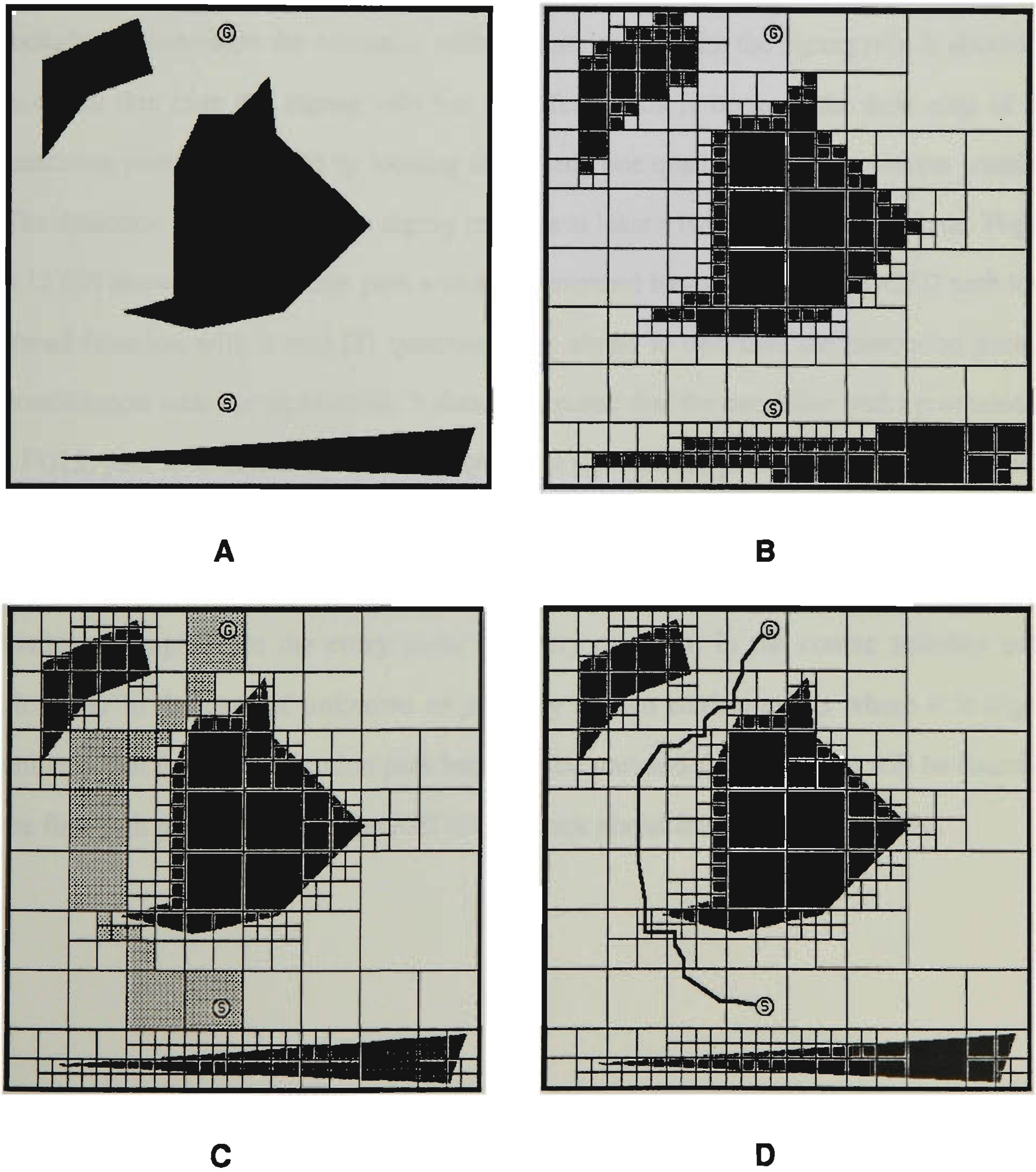


Figure 4.11

Path Planning in an known environment.

The execution path shown in Figure 4.11 can be improved by using the zigzag rule described in Section 4.4.1. Figure 4.12 (A) shows the improved execution path which is generated by steering through the middle of the intersection edge of the free space quadrants in combination with the zigzag rule. Figure 4.12 (B) shows the execution path which is generated by using the FULL path look ahead function to optimise the execution path in combination with the zigzag rule. Figure 4.12 (C) shows the execution path which is generated by using the REDUCED path look ahead function with a one (1) quadrant lookahead to optimise the execution path in combination with the zigzag rule. It should be noted in this case the zigzag rule has no effect. This is because the next step of the execution path is generated by looking ahead only one quadrant from the current position. The detection and removal of a zigzag requires at least a two quadrant look ahead. Figure 4.12 (D) shows the execution path which is generated by using the REDUCED path look ahead function with a two (2) quadrant look ahead to optimise the execution path in combination with the zigzag rule. It should be noted that the execution paths produced by a FULL path look ahead and a two (2) quadrant REDUCED path look ahead are identical, as shown by Figures 4.12 (B) and (D). In a known environment FULL look ahead should be used, since the REDUCED look ahead requires the construction of a small visibility graph from the entry point of every quadrant in the coarse solution path. However in the case of unknown or partially known environments where it is highly unlikely that the final execution path between the start and goal locations will be found on the first path planning effort, the REDUCED look ahead function is very useful.

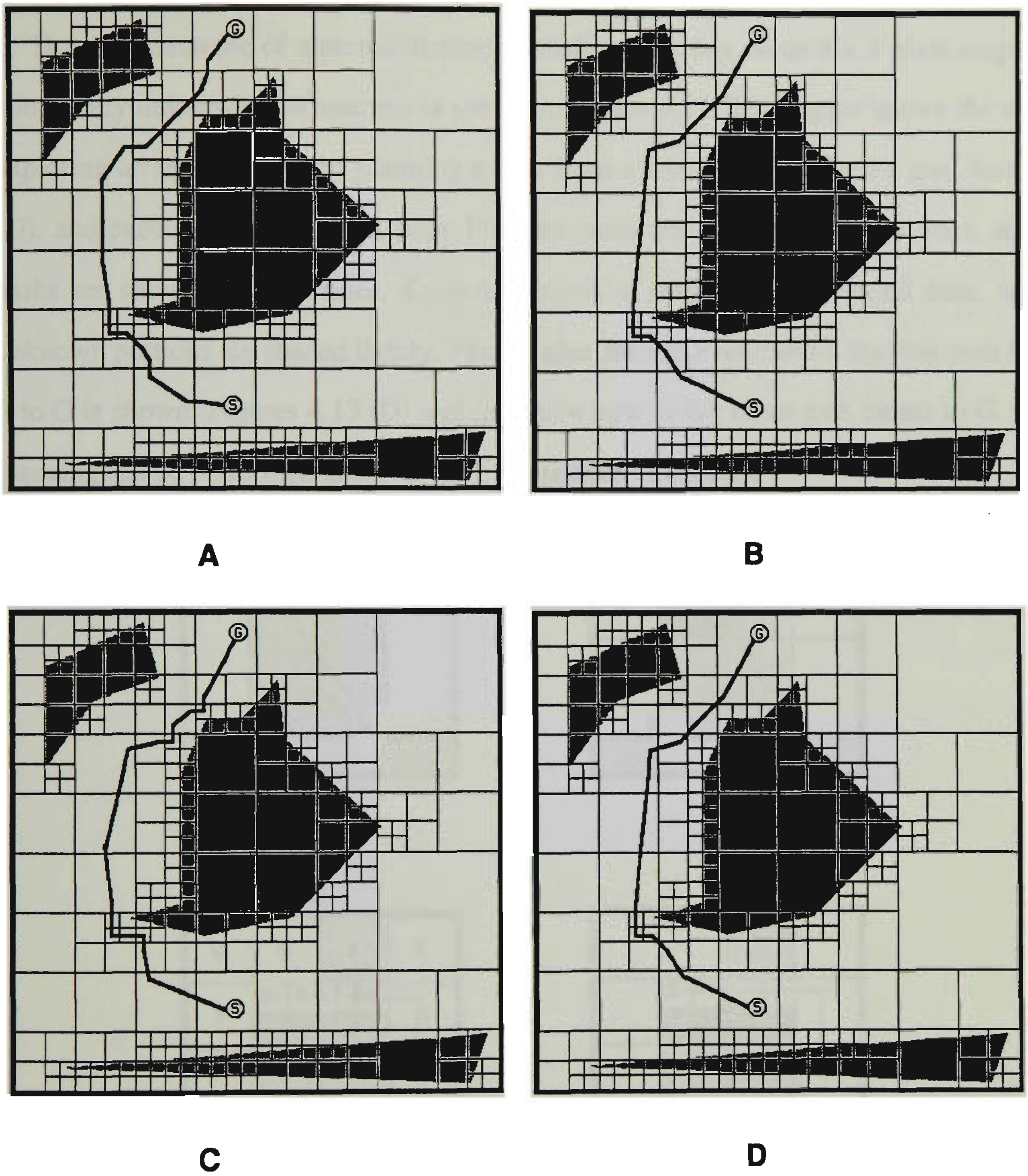


Figure 4.12

Improved Path Planning in an known environment.

The third example of a partial distance transform running on an 8 x 8 pixel map in a completely unknown environment is shown in Figure 4.13. This figure shows the robot exploring an environment by planning a path from a start location (S) to a goal location (G), and executing the planned path. Planned paths are shown as broken lines, actual paths are shown as solid lines. Known portions of obstacles are shaded dark, while unknown portions are shaded lightly. Finally after the robot reaches G, the best path from S to G is shown. Figures 4.13 (D) and (E) show how as the robot gets closer to G, path planning can be performed using the Partial Distance Transform.

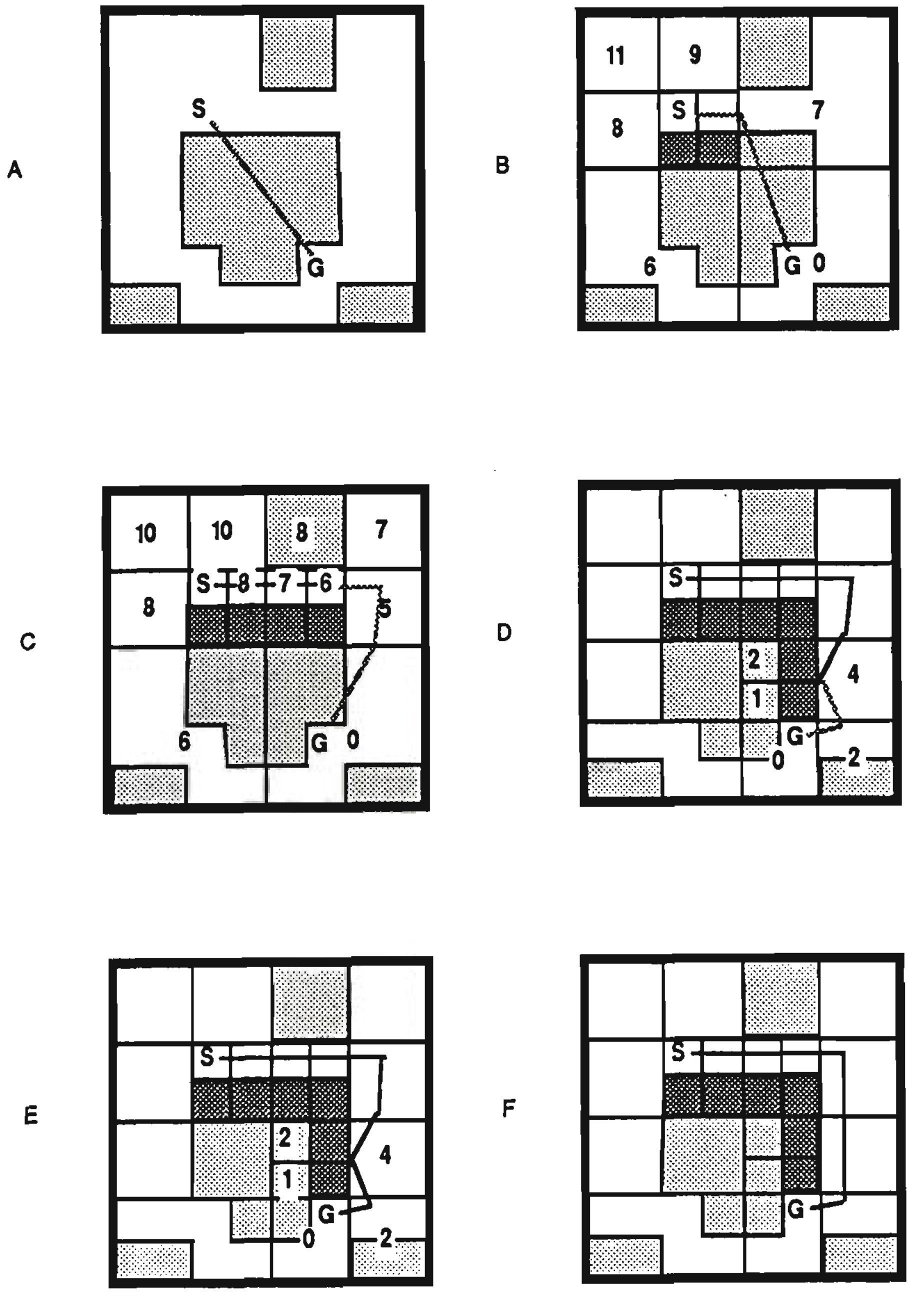


Figure 4.13

The Partial Distance Transform.

4.8 Experimental Results of the EEA

In this section four experimental results of the EEA using sonar data are presented. In Chapter 2 of this thesis a method was presented for building environment maps for a mobile robot from live sonar data. Chapter 2 presented four experimental results of map making (namely Figures 2.17 - 2.20). These four results have been used as input to the EEA. The EEA was implemented on a Macintosh II microcomputer.

Figure 4.14 shows the results of the EEA operating on the environment map described in Figure 2.17. Figure 4.14 (A) shows the initial map together with the start (S) and goal (G) locations, which were supplied to the EEA. From the map data a quadtree representation of the environment was built as shown in Figure 4.14 (B). A path between the start and goal positions was planned using the two (2) quadrant REDUCED look ahead function. The planned path is shown in Figure 4.14 (C). During the course of path execution to the goal the robot encounters an obstacle at the position (R) as shown in Figure 4.14 (C). At this point the navigation system instructs the mobile robot to collect more sonar range readings and build a new map. From the new map data the quadtree is reconstructed and a new path is planned to the goal as shown in Figure 4.14 (D).

Figures 4.15, 4.16 and 4.17 show the results of the EEA operating on the environment maps described in Figures 2.18, 2.19 and 2.20. Diagram A in all the figures shows the initial map together with the start and goal locations, which were supplied to the EEA. The quadtree representation of the environment map, and the planned path between the start and goal locations using the two (2) quadrant REDUCED look ahead function are shown in Diagram B in all of the figures.

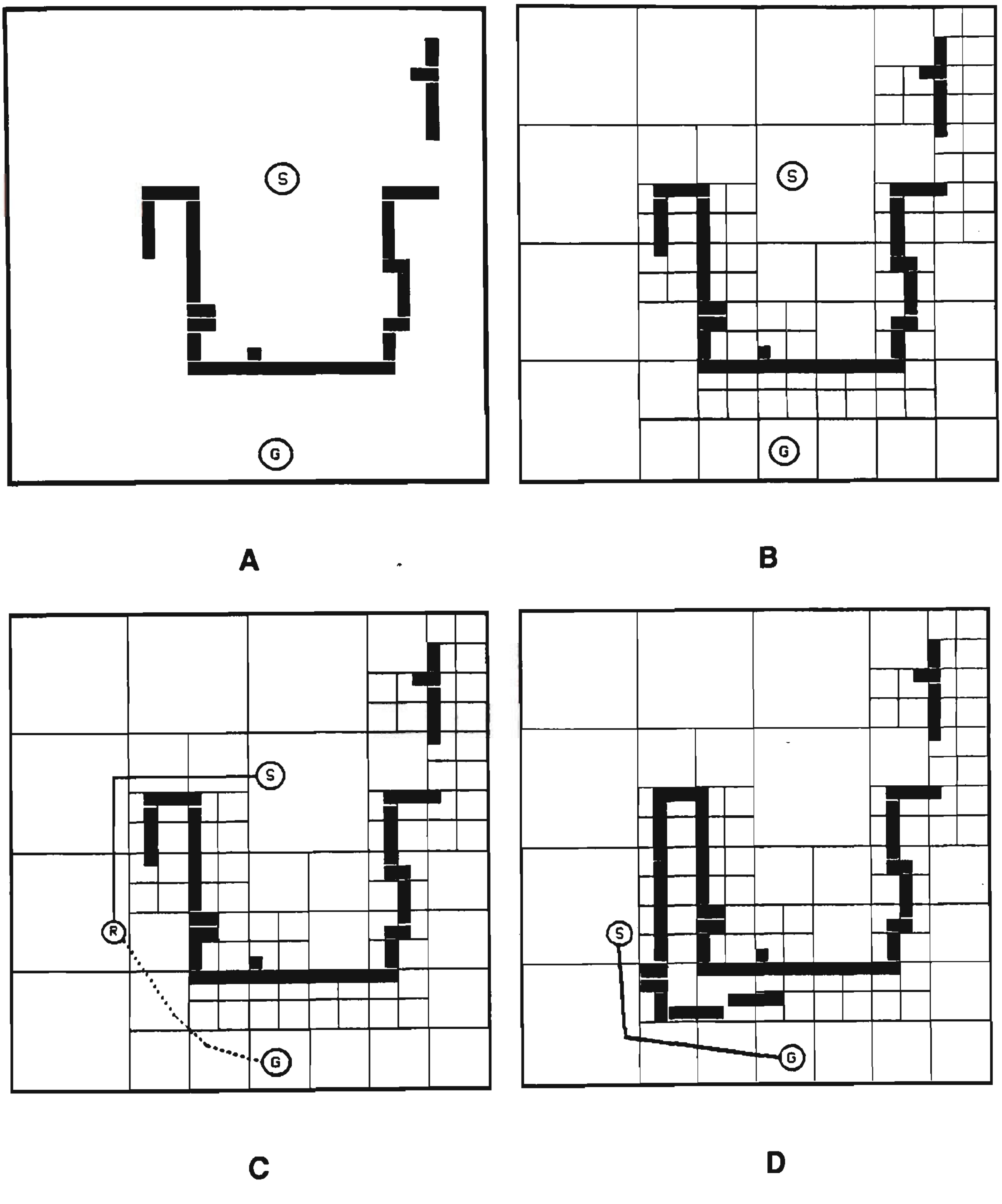


Figure 4.14

Operating the EEA with sonar range data from Figure 2.17.

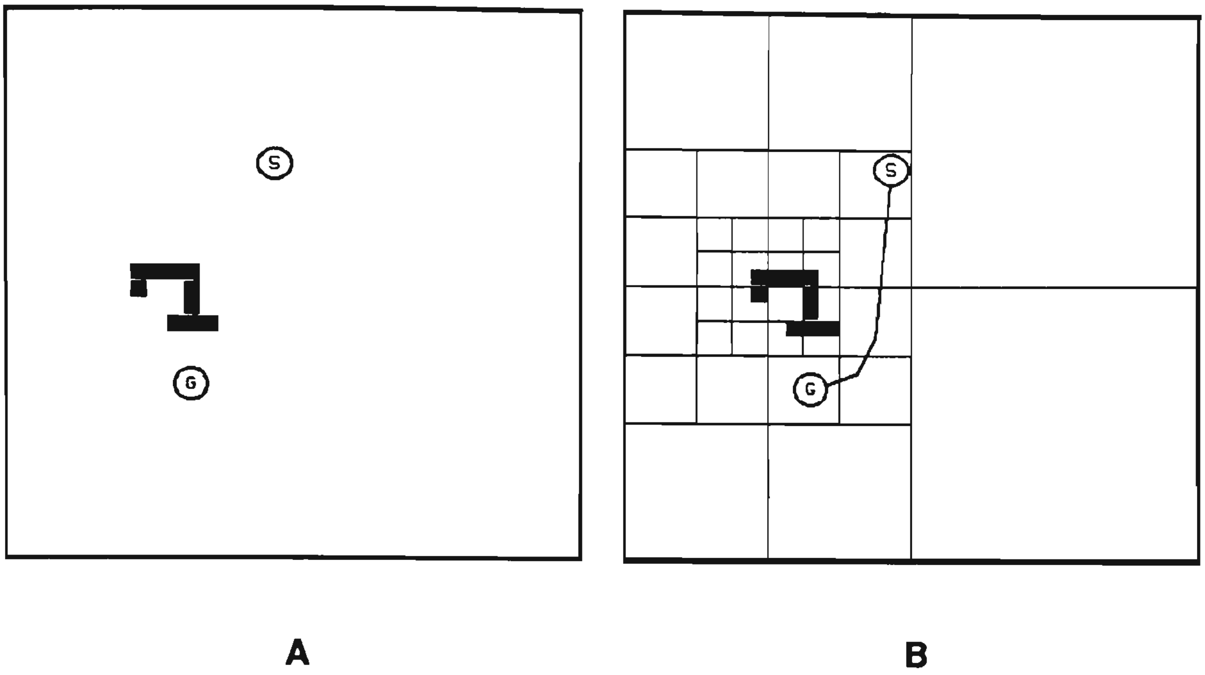


Figure 4.15

Operating the EEA with sonar range data from Figure 2.18.

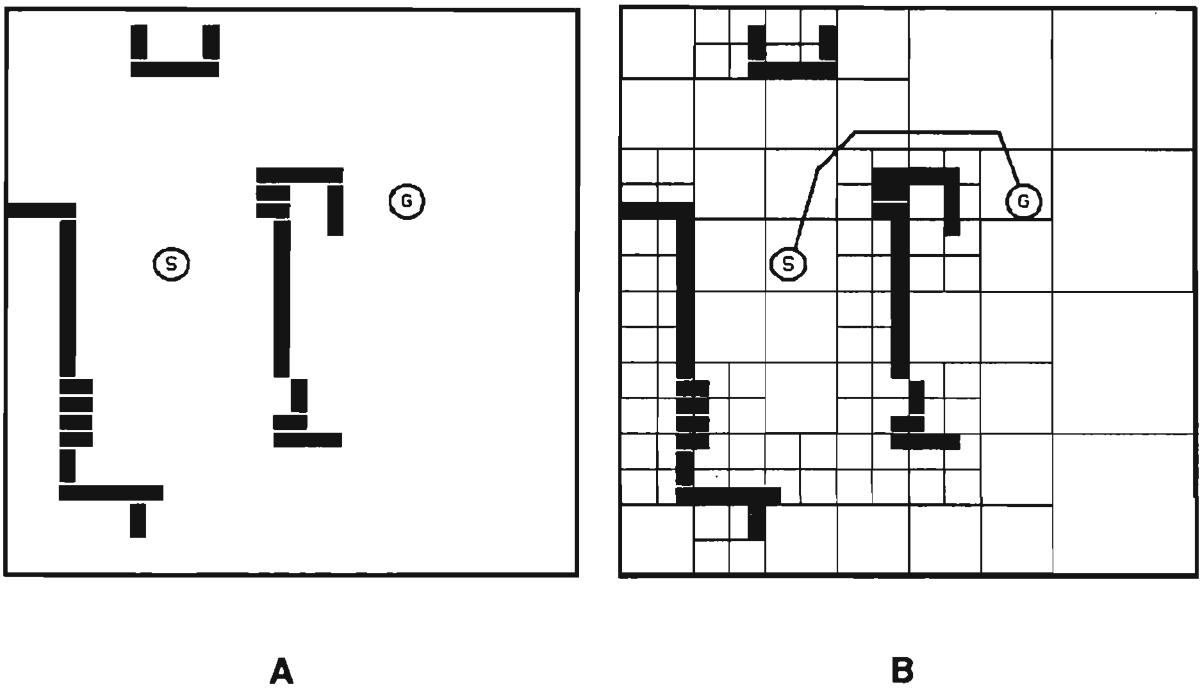


Figure 4.16

Operating the EEA with sonar range data from Figure 2.19.

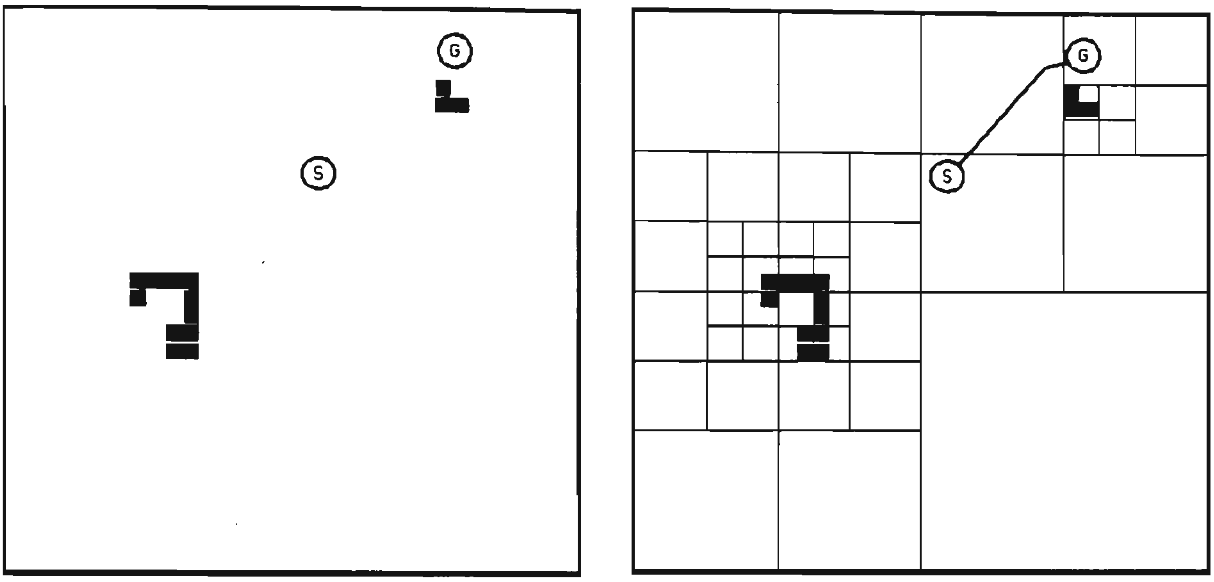


Figure 4.17

Operating the EEA with sonar range data from Figure 2.20.

4.9 Conclusions

This chapter has shown that quadtrees and distance transforms provide an effective mechanism for developing an algorithm (EEA) to explore an environment with a mobile robot. It has been shown that environments can be efficiently modelled with quadtrees, and distance transforms can be applied to plan paths in known and unknown portions of an environment. The EEA provides an elegant way of planning paths in unknown regions of an environment. The algorithm assumes that all unknown areas are free space. This assumption allows unknown regions to be efficiently modelled and path planning is straight forward. However, should the assumption prove to be false i.e the unknown areas are not free space, the EEA can readily incorporate new environment data into the quadtree and replan a fresh path. The EEA approach to path planning which consists of making an optimistic plan, attempting to execute the plan and modifying the plan in the event of a failure of the original plan, is in keeping with human path planning strategies which were discussed in Section 3.2.

An efficient method for propagating the distance transform through a quadtree was developed. Despite the efficiency of this new technique the distance transform is computationally expensive. The problem of expensive computation of the distance transform was addressed by limiting the recomputation of the distance transform to a partial update. The affect of the partial distance transform on the EEA is that the algorithm does less work as the robot gets closer to the goal.

Path planning using quadtrees and distance transforms yields a coarse solution path of free space quadrants. In this chapter a new mechanism was developed for the efficient extraction of a fine path from the coarse path. This was done by the construction of a dynamic visibility graph. The type of fine path that can be generated with this new technique is flexible. The visibility graph can be used to fully or partially optimise the path between the start and goal locations. Full path optimisation is only useful if the environment is known. Fully optimising a path in an unknown environment is likely to be a computational waste, since an unexpected obstacle will necessitate replanning the path. The partial optimisation of the fine path is a cheap and useful strategy in an unknown environment.

Despite the limitation that the EEA stipulates that paths must lie completely within the boundaries of free space quadrants, and that paths between quadrants are allowed only in the horizontal and vertical directions, the EEA produces "reasonable" paths. While a solution path is not a shortest path to the goal, the path is a "negotiable" path. A negotiable path avoids clipping the corners of obstacles and does not run along the edges of obstacles. Such a path strategy is acceptable since there are uncertainties in the exact shape and position of obstacles in the environment, and the precise position of the robot is not always known.

In this chapter it was shown that the EEA can operate with a real mobile robot using live sonar data to navigate autonomously in an unknown environment. Experiments were performed which required the robot to navigate to goals that were located in unknown regions of the environment. In all the experiments the robot was able to successfully reach each desired goal, and in doing so, explore and build maps of the environment.

Chapter 5
Path Planning Behaviours

5.1 Introduction

The second goal of this thesis was to build a robot path planner that is capable of inducing different types of path planning behaviours. A robot path planner should not only find the "optimum" paths i.e. shortest distance to the goal, but the system should also be able to generate "conservative", "adventurous", and "visit all" paths. A robot path planner should also produce a behaviour, which allows the robot to systematically "learn all" the unknown regions of an environment. In all the discussions this far, it has been assumed that the robot is dealing with a static non-changing environment. Thus once the robot discovers an obstacle the information regarding the shape and position of the obstacle does not change thereafter. However, in practice environments are dynamic. The position of obstacles in the environment can change. A robot path planner can be adapted to operate in a dynamic environment by the addition of a "forgetful" behaviour. This behaviour can co-exist with the other robot path planning behaviours. The "forgetful" behaviour causes the robot to gradually forget information that has been acquired about the environment. Knowledge about parts of the environment which have not been sighted by the robot for a significant time gradually decay and finally disappear. The effect of the forgetful behaviour upon robot knowledge of the environment can be looked upon as the vapour trail behind a jet airplane.

Section 5.2 describes a method for extending the capabilities of the Environment Exploration Algorithm (EEA), to include "conservative", "adventurous", "visit all", "learn all" and "forgetful" path planning behaviours. Section 5.3 describes how the original algorithm for the EEA which was presented as Algorithm 4.1 can be reformulated to include the concept of path planning behaviours.

A great deal of past research into mobile robot navigation has concentrated on the problem of finding the shortest paths through known environments. The biggest challenge in building a competent path planner is to reconcile the conflicting requirements of finding what is a "safe" path from a start location to a goal location. The "safe" path should be the shortest possible path, which maintains a safe distance from obstacles. Section 5.4 presents a mechanism which allows the EEA to plan "safe" paths for a robot to execute. This section also describes how the consideration of safety distance from obstacles can be extended to the [Jarvis *et. al.* 86] path planner which is based on distance transforms applied to grids.

The third goal of this thesis was to construct a path planner which could find the "best" path from a start location to a goal location. The "best" path should be the shortest possible path, staying outside unknown areas, and at the same time keeping a safe distance from obstacles. Section 5.5 presents a method for further extending the capabilities of the EEA, to include the consideration of safety distance from obstacles while staying outside unknown areas.

Section 5.6 presents experimental results of robot path planning behaviours using the sonar range data which was collected in Chapter 2.

Finally in Section 5.7 the conclusions that were reached and the insights that were gained from investigating the problem of path planning behaviours are presented.

The path planning behaviours presented in this chapter are applicable to a robot which has been approximated as a cylinder.

5.2 Path Planning Behaviours

This section describes how the following six different path planning behaviours can be implemented:

- * optimum path behaviour
- * conservative path behaviour
- * adventurous path behaviour
- * learn all behaviour
- * visit all behaviour
- * forgetful behaviour

Section 3.3.3 of this thesis reviewed the work of [Jarvis *et. al.* 86], who used distance transforms based on grids to generate "optimum", "conservative", "adventurous", and "visit all" paths. This approach used a "factor" function to give different weights to the distance transform depending on the type of grid cell; known or unknown. Using the factor function set to 1 for known and unknown cells produced an optimum path behaviour. However the factor function could be varied to induce different behaviour. If an "adventurous" behaviour was required the factor in known cells was doubled. If on the other hand, a "conservative" behaviour was sought in which the robot avoided unknown cells, the factor in unknown cells was doubled.

To implement path planning behaviours with the EEA, a mechanism for representing known and unknown regions in a quadtree is required. To achieve this aim, additional space needs to be provided within each quadtree node to store the "confidence" the navigation system has in a node belonging to a particular class e.g. 90% confidence that this node is a free node. The confidence value in a quadtree node is updated by the navigation system every time the robot visits or observes the node. When the robot visits a quadtree node the confidence value of the node is calculated by determining the area that has been swept by the robot inside the quadrant during the execution of a planned path, and dividing it by the area of the quadrant. Quadtree nodes which are sighted by the robots environment sensors have their confidence values updated. This update is done by determining the area that has been sighted by the robot, and dividing it by the area of the quadrant.

The allocation of memory to store confidence values in internal nodes of the quadtree can be viewed as unnecessary overhead since confidence values apply only to the leaf nodes of the quadtree. However the tree structure of the quadtree can be exploited to keep track of how much of the environment is known. The leaf nodes in the quadtree contain the confidences about the structure of the environment. A postorder traversal of the quadtree summing up the confidences of the leaves, dividing this sum by (4) four, and passing the result up to the parent node, will yield at the root node the overall confidence level that the EEA possesses about the structure of the environment.

Figures 5.1 and 5.2 show the growth in the confidence values of the leaf nodes of a quadtree environment model as a robot equipped with a tactile sensor navigates in an unknown environment, from a start location (S) to a goal location (G). Figure 5.1 (A) shows the initial conditions in which the robot has no knowledge about its environment. The robot assumes that the environment is free space. However the confidence in this assumption is 0%. As the robot navigates towards the goal, as shown in Figures 5.1 (B) - (D) and 5.2 (A) - (D), it acquires more information about the environment and the confidence values of the free space quadrants that have been traversed grow accordingly.

The implementation of path planning behaviours described in this research uses a "factor" function, similar to the [Jarvis *et. al.* 86] approach. The "factor" function uses the confidence values of the free space quadrants to give the relevant weighting to the distance transform, for each path planning behaviour. The EEA evaluates the "factor" function by a set of rules. The evaluation rules for each behaviour are described in the following three subsections.

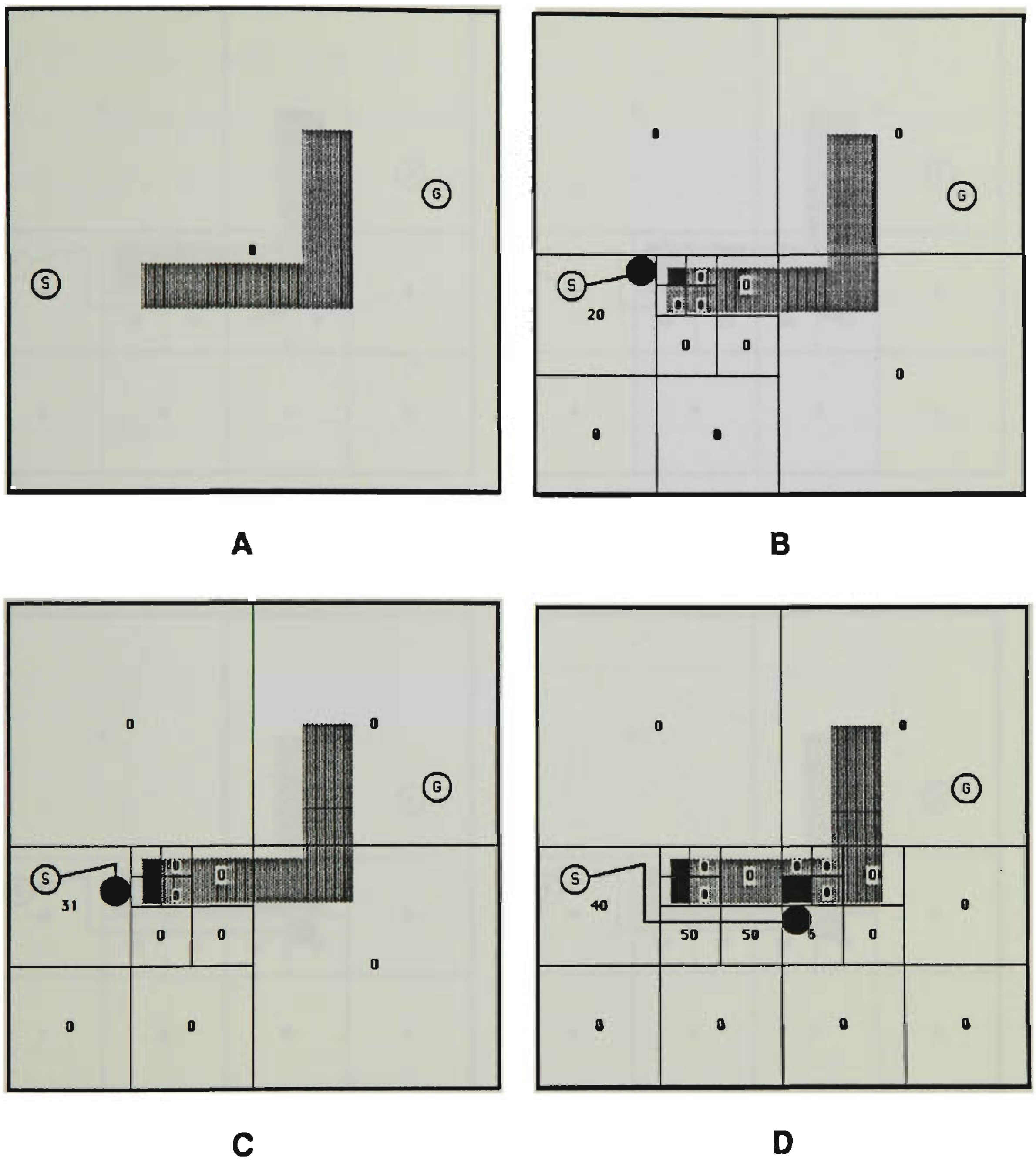


Figure 5.1

Growth in the confidence values of free space quadrants during path execution.

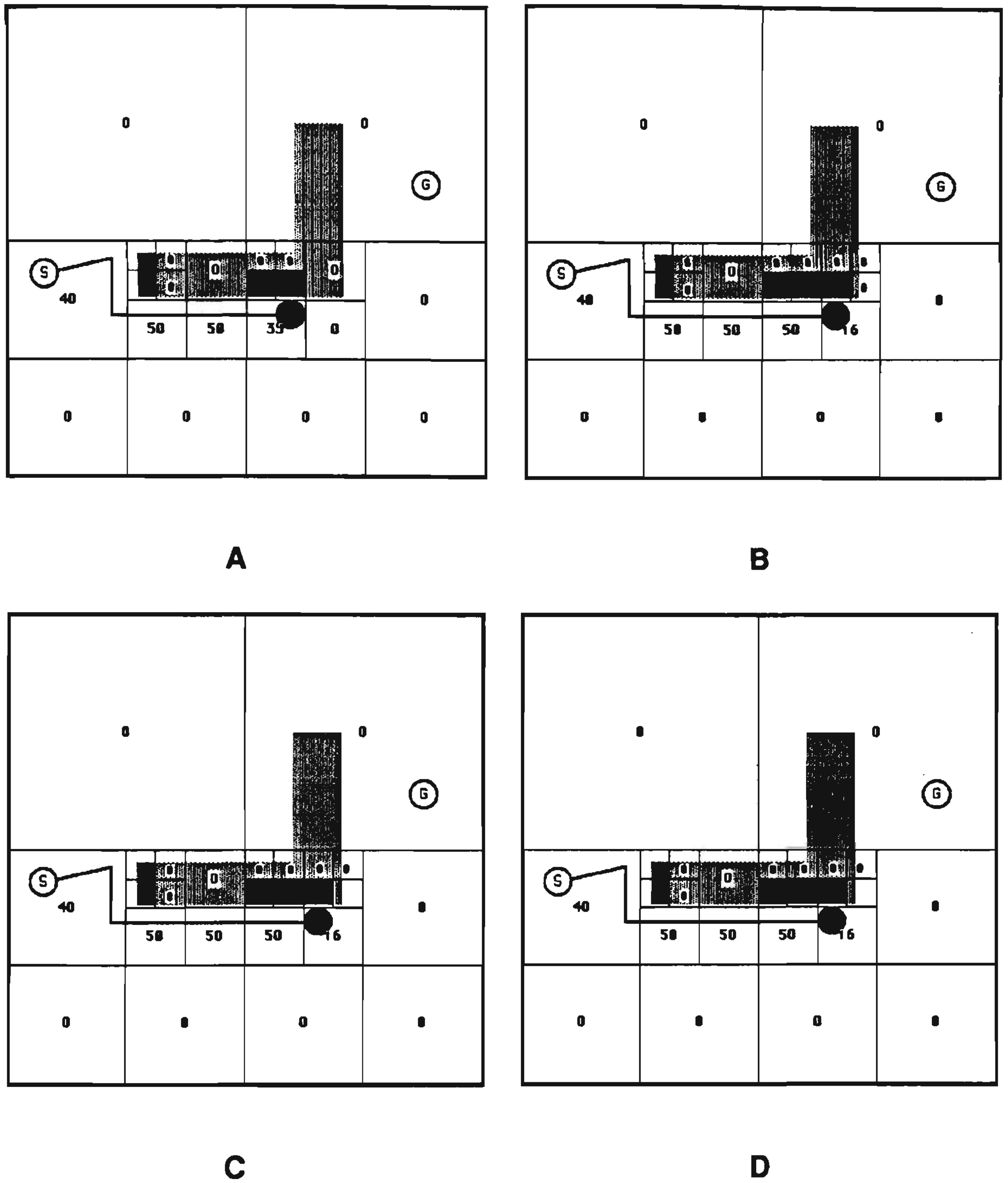


Figure 5.2

This figure continues the experiment which was started in Figure 5.1.

5.2.1 Optimum Path Planning Behaviour

If the robot is in the behaviour mode of planning "optimum" paths, free spaces and unknown spaces are considered to be the of the same class, since the best assumption that can be made of unknown space is that it is free space. In this case the "factor" function is set to a value of 1. When the distance transform is propagated through the quadtree it is multiplied by the "factor" function. In the case of planning "optimum" paths this results in a normal distance transform propagation. Figure 5.3 shows the distance transform which corresponds to the behaviour of planning an "optimum" path, for the environment exploration expedition which was described in Figures 5.1 and 5.2. In this figure the broken line shows the "optimum path" between the start (S) and goal (G) locations.

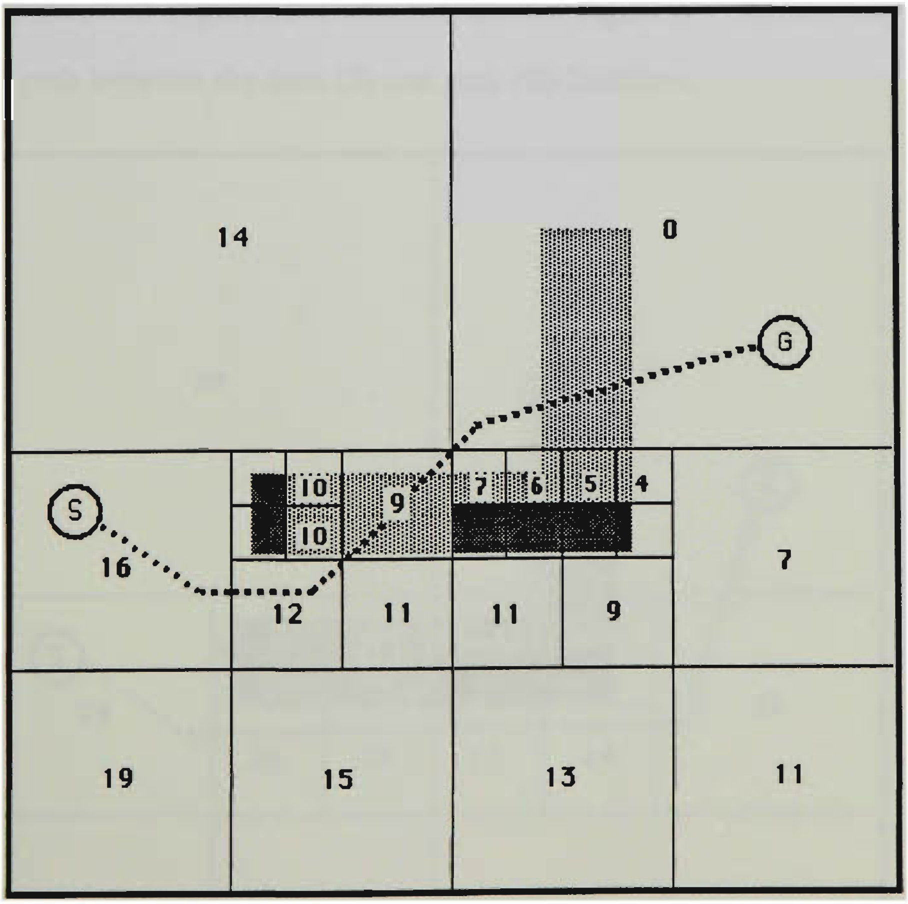


Figure 5.3

Optimum path planning behaviour.

5.2.2 Conservative Path Planning Behaviour

Once a robot has learnt a portion of the environment, there could be a need to find a path to a goal using the free space quadrants in which the EEA possesses the greatest knowledge. In this case the factor function evaluates to *1 + [1 - confidence]*, where *[1-confidence]* is a measure of the confidence the system has in a leaf *not* being free space. Unknown quadrants are assumed to be free spaces, with zero (0) confidence. When the distance transform is propagated through the quadtree using the factor function, this results in a weighted distance transform which has higher costs for the traversal of unknown regions. Figure 5.4 shows the distance transform which corresponds to the behaviour of planning a "conservative" path, for the environment exploration expedition which was described in Figures 5.1 and 5.2. In this figure the broken line shows the "conservative" path between the start (S) and goal (G) locations.

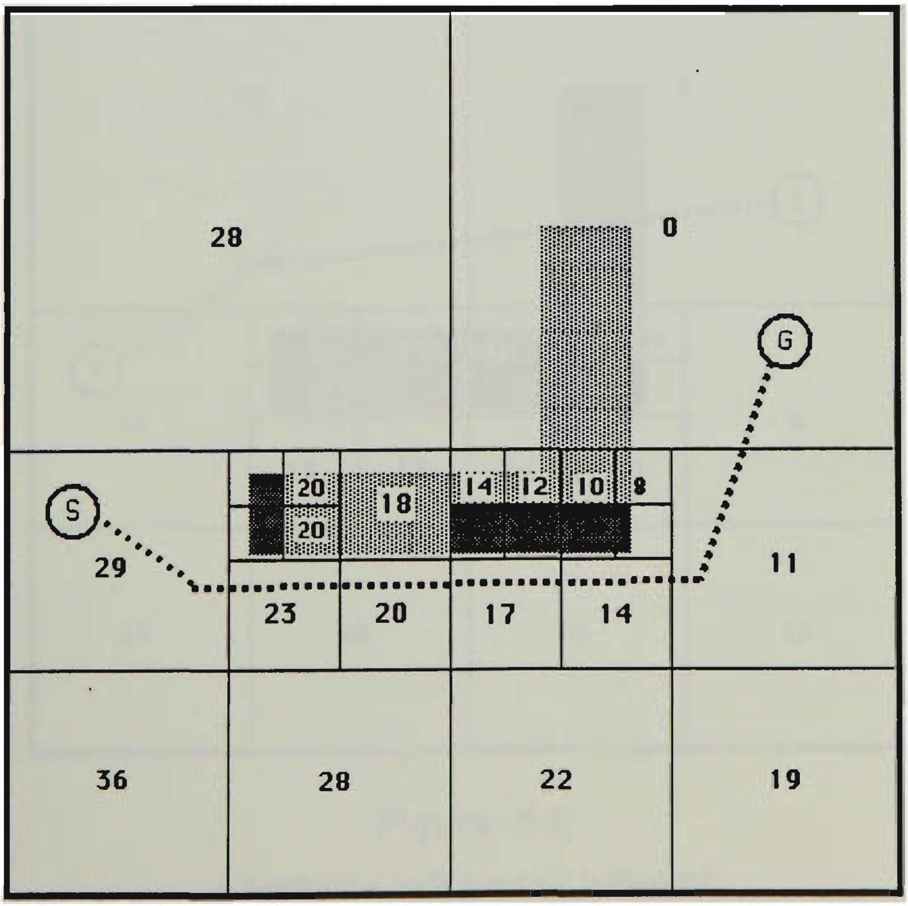


Figure 5.4

Conservative path planning behaviour.

5.2.3 Adventurous Path Planning Behaviour

A robot may operate in an exploratory frame of mind and favour unknown spaces en route to a goal. In this case the function factor evaluates to *1 + confidence*, where *confidence* is a measure of the confidence the system has in a quadrant being free space. When the distance transform is propagated through the quadtree, this results in a weighted distance transform which has higher costs for the traversal of known regions. Figure 5.5 shows the distance transform which corresponds to the behaviour of planning an "adventurous" path, for the environment exploration expedition which was described in Figures 5.1 and 5.2. In this figure the broken line shows the "adventurous" path between the start (S) and goal (G) locations.

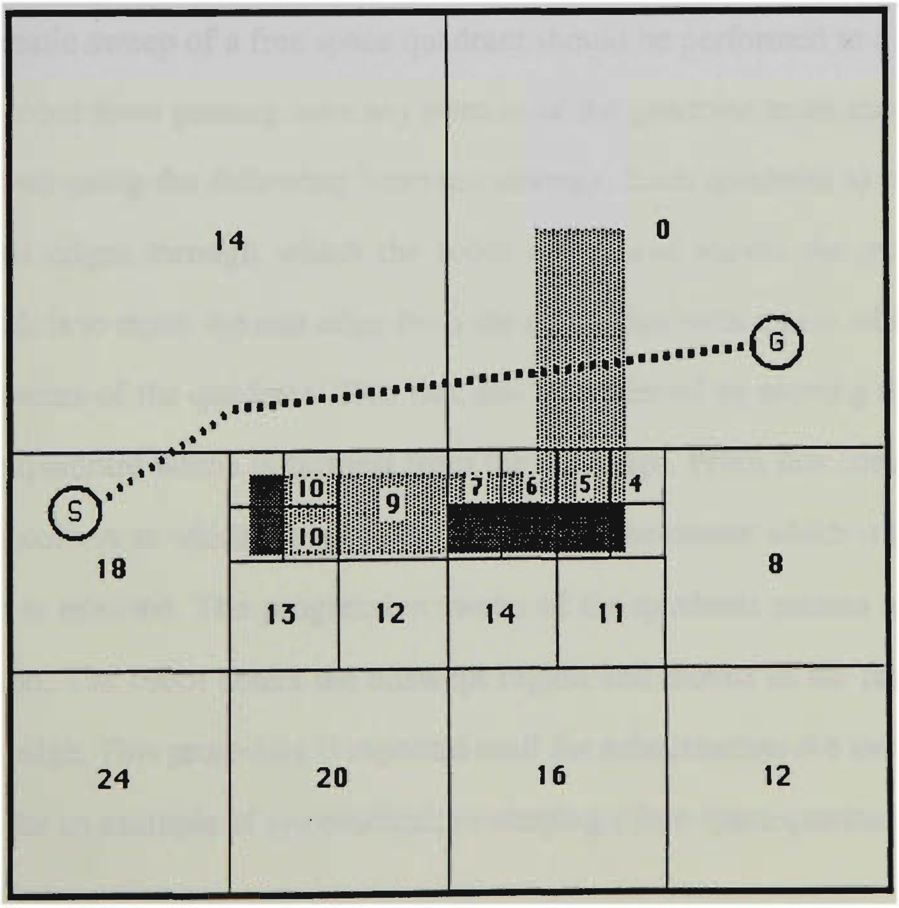


Figure 5.5

Adventurous path planning behaviour.

5.2.4 Learn All Path Planning Behaviour

The robot may operate in the behaviour mode of planning "learn all" paths. This path planning behaviour can be accomplished by making the centroids of free space quadrants with the lowest confidence values the goals. Once the robot has entered a goal quadrant, the robot traverses the whole quadrant in a systematic manner, so that the robot gains 100% confidence about the contents of the quadrant. To achieve the "learn all" mode of robot navigation, the distance transform for "adventurous" path planning is used. This causes the robot to traverse free space quadrants with low confidence values en route to the goal. To improve the efficiency of the learn all mode, the robot should also systematically sweep the low confidence free space quadrants en route to the goal.

The systematic sweep of a free space quadrant should be performed in a manner that prevents the robot from passing over any portion of the quadrant more than once. This can be achieved using the following heuristic strategy. Each quadrant to be swept has entry and exit edges through which the robot enters and leaves the quadrant. The navigation task is to reach the exit edge from the entry edge with a path which traverses the entire contents of the quadrant. This task can be achieved by moving the robot to a corner of the quadrant which is furthest from the exit edge. From this corner there is a choice of two corners to which the robot can be moved. The corner which is furthest from the exit edge is selected. The progressive sweep of the quadrant creates a rectangular unswept region. The robot enters the unswept region and moves to the furthest corner from the exit edge. This procedure is repeated until the robot reaches the exit edge. Refer to Figure 5.6 for an example of systematically sweeping a free space quadrant.

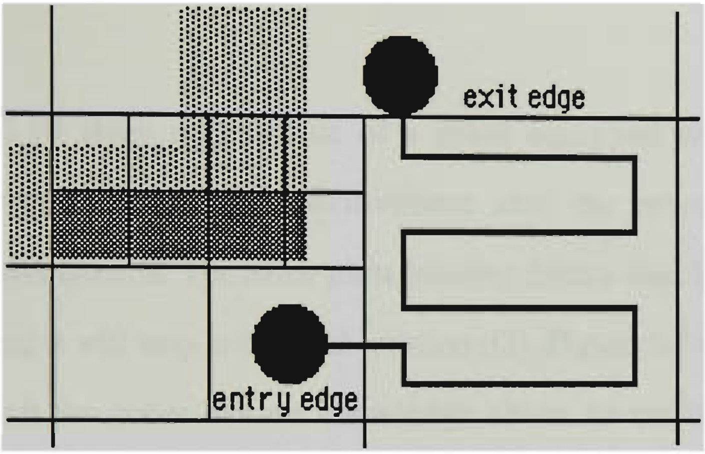


Figure 5.6

Systematic sweeping of a free space quadrant with a cylindrical robot.

The tree structure of the quadtree is exploited to keep track of how much of the environment has been learnt. A postorder traversal of the quadtree summing up the confidences of the leaves, dividing this sum by (4) four, and passing the result up to the parent node, will yield at the root node the overall confidence level that the EEA possesses about structure of the environment. The "learn all" path planning behaviour monitors the overall confidence value, and continues exploring the environment until the overall confidence value has reached 100%. Using this approach it is possible to set a threshold on the level of knowledge that is to be acquired by the "learn all" path planning behaviour. For example it could be decided to set the threshold at 80%, the EEA will continue to learn the environment until the overall confidence value stored in the root of the quadtree has reached the threshold value. Adopting a hierarchical structure to store the confidence levels about the environment assists in the search for the lowest confidence quadrant which needs to be explored next by the EEA. Rather than perform a breadth first search of the quadtree for the quadrant with the lowest confidence value, a depth first search is performed. The child of the root node with the lowest confidence value is selected. If the selected node is a leaf node then the search is terminated, otherwise the children of the selected node are searched for the lowest confidence valued child. This search continues until a leaf node is reached. While this strategy does not guarantee that the lowest confidence node in the quadtree is selected, a quadrant will be selected which requires exploration.

Figures 5.7 - 5.10 show an example of a robot equipped with a tactile sensor progressively learning an unknown environment until the robot has gained 100% confidence in the environment. The robot starts learning from a start location (S) and once it completes learning it will stop at the goal location (G). Figure 5.7 (A) shows the initial conditions in which the robot has no knowledge about its environment. The robot assumes that the environment is free space. However the confidence in this assumption is 0%. The robot systematically sweeps the environment, as shown in Figures 5.8 - 5.10. When an obstacle is encountered the environment quadtree model is updated and the confidence values of the free space quadrants that have been traversed are updated accordingly. Figure 5.10 (D) shows that the robot has learnt the entire contents of all the free space quadrants that can be reached. Upon completion of learning the robot returns to the goal specified in Figure 5.7 (A).

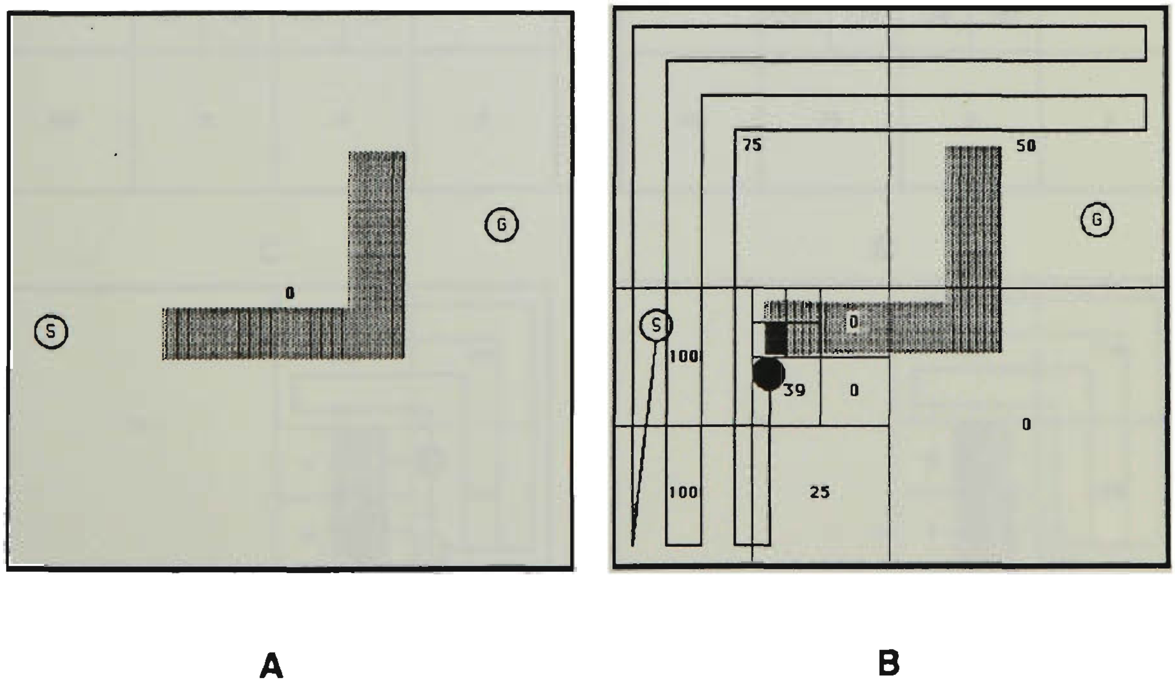


Figure 5.7

Learn all path planning behaviour.

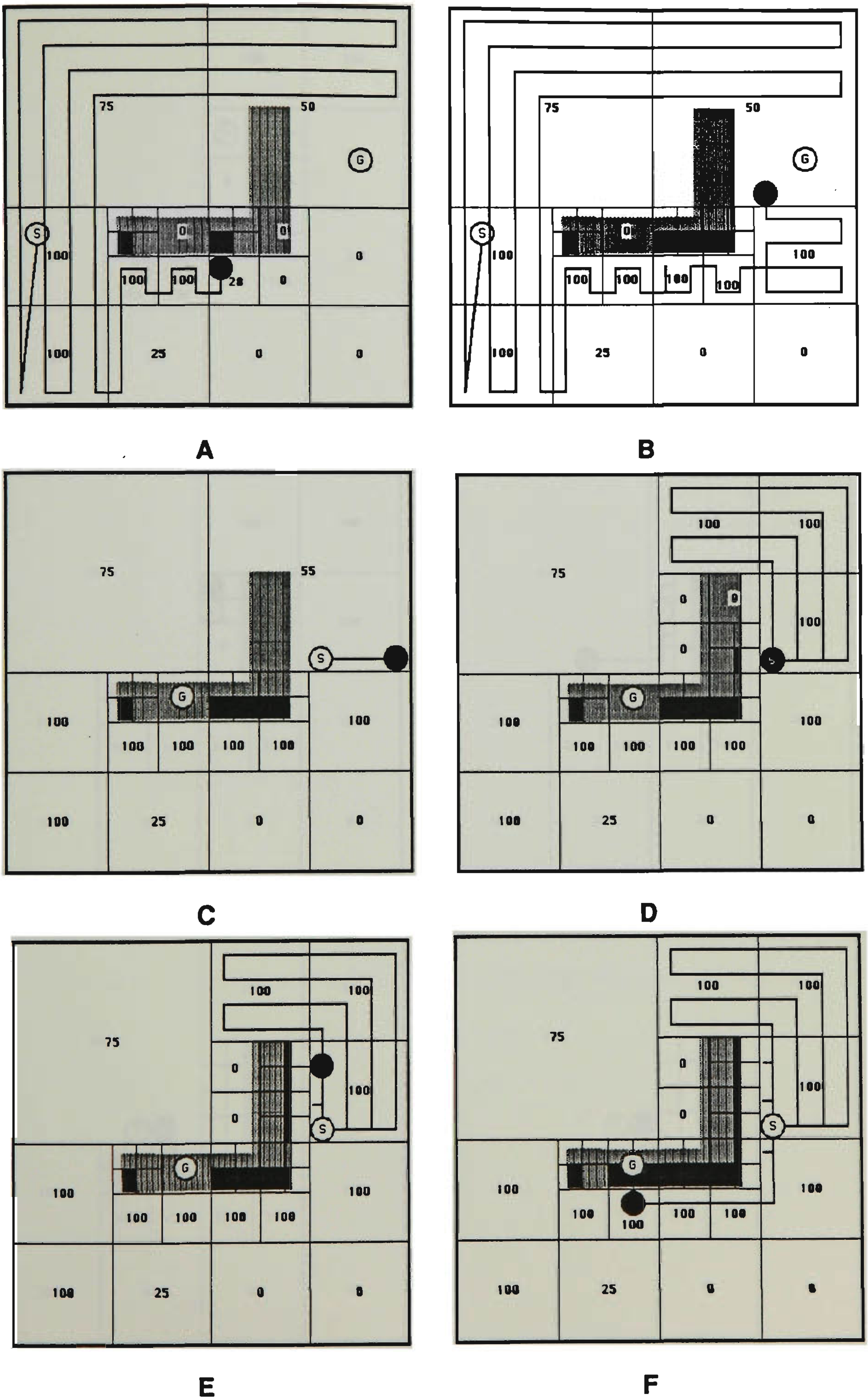


Figure 5.8

This figure continues the experiment which was started in Figure 5.7.

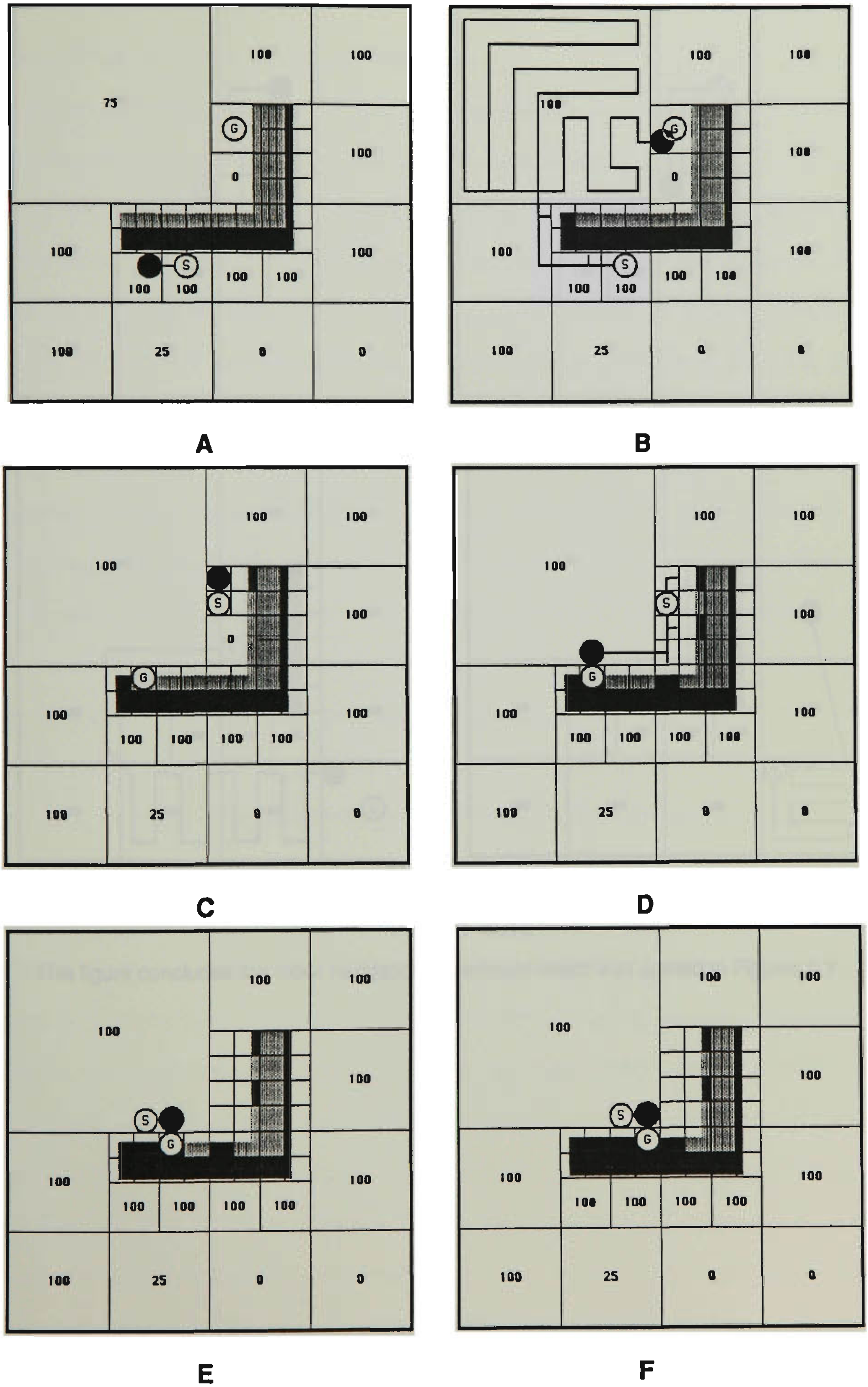


Figure 5.9

This figure continues the experiment which was started in Figure 5.7.

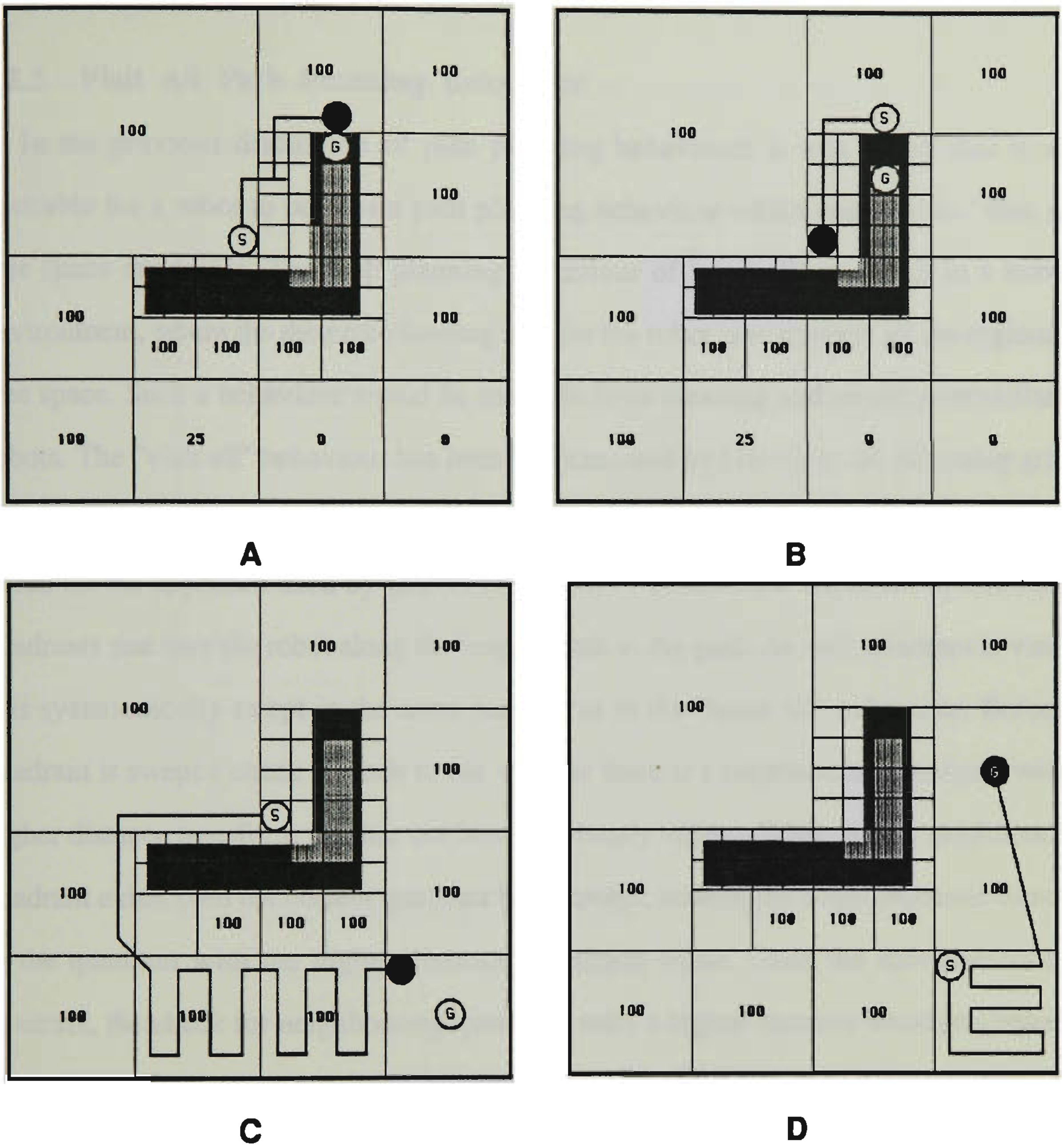


Figure 5.10

This figure concludes the robot navigation experiment which was started in Figures 5.7.

5.2.5 Visit All Path Planning Behaviour

In the previous discussion of path planning behaviours it was stated that it was desirable for a robot to possess a path planning behaviour which causes it to "visit all" free space quadrants. The path planning behaviour of "visit all" is useful in a known environment, where the the path planning task for the robot is to traverse all the regions of free space. Such a behaviour would be useful in floor cleaning and security surveillance robots. The "visit all" behaviour has been implemented by [Jarvis *et. al.* 88] using grids. This behaviour can also be implemented using quadtrees. The quadtree implementation is based on the approach used by [Jarvis *et. al.* 88]. It follows the sequence of free space quadrants that take the robot along the longest path to the goal. As each quadrant is visited it is systematically swept in the same manner as in the "learn all" behaviour. Before a quadrant is swept a check is made to see whether there is a neighbouring quadrant with a higher distance transform that has not been previously visited. When such a neighbouring quadrant exists then the current quadrant is not swept, instead the robot proceeds directly to the quadrant with the higher distance transform value. Once the robot enters this quadrant, the check for neighbouring quadrants with a higher distance transform value is repeated. A free space quadrant is not swept unless the quadrant is surrounded by quadrants with lower distance transform values or quadrants that have already been systematically swept. Such a strategy is necessary to prevent the robot from crossing quadrants which have been swept. This may seem to be an unnecessary measure; however if the robot was a floor painting robot, the robot could easily paint itself into a corner and would have no recourse other than traversing areas covered with wet paint. The visit all behaviour implemented with quadtrees produces a reasonable path between the start and goal locations. However the resulting path using quadtrees is generally inferior to the one produced using grids. This is due to the fact that quadtree distance transform values are less exact measures of the distance to the goal than grid distance transform values. It is desirable to follow the rings of distance transform contours as they radiate from the goal. It is easier to follow the distance transform contours with grids rather than with quadtrees. Refer to Figure 5.11 for an example of the visit all behaviour implemented with quadtrees and grids for an identical environment with the same start(S) and goal(G) locations.

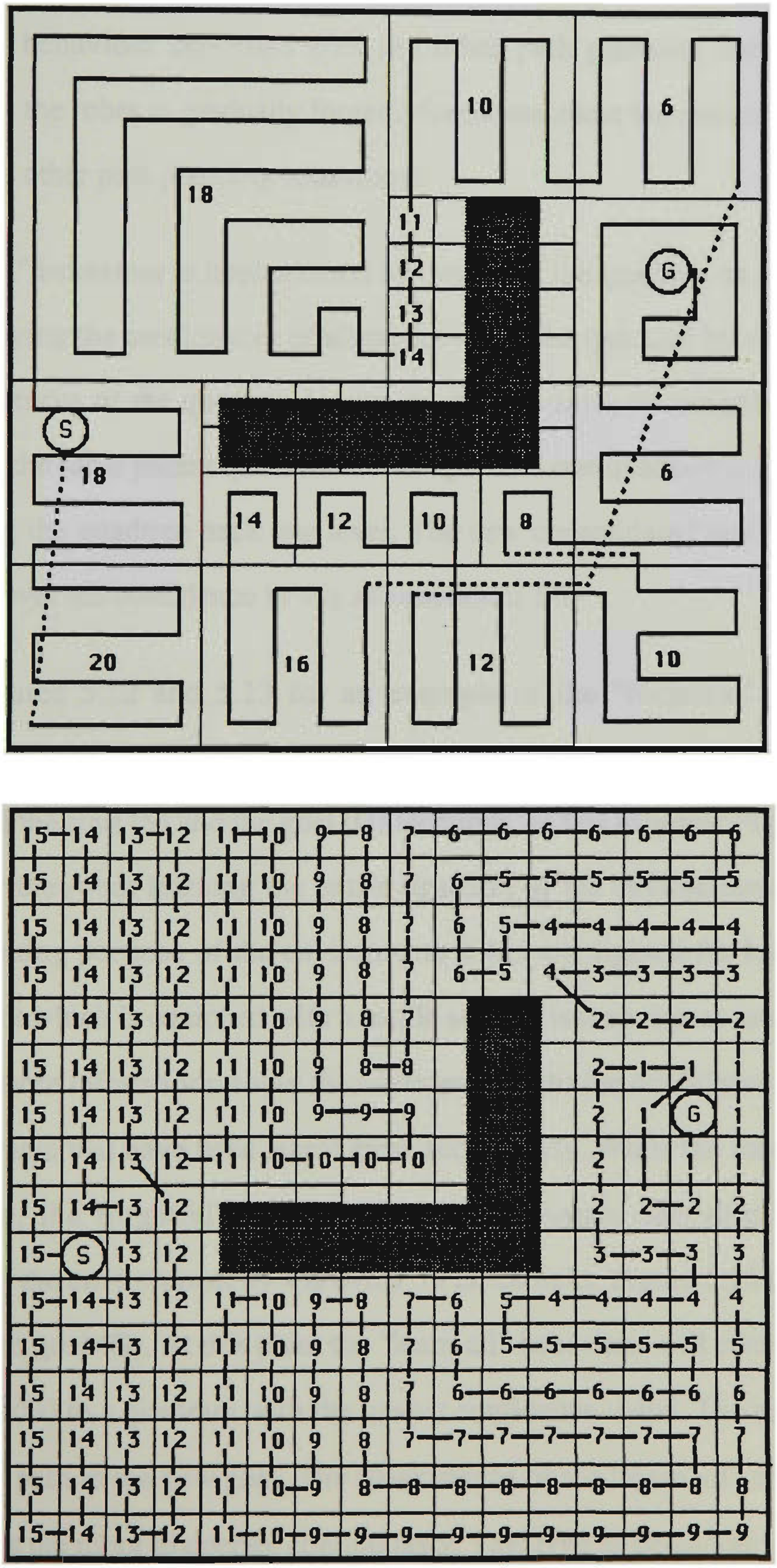


Figure 5.11

This Figure shows the Visit All path planning behaviour for quadtrees and grids.

5.2.6 Forgetful Path Planning Behaviour

In the introductory discussions of this chapter, it was stated that a "forgetful" behaviour is a useful mechanism that allows a robot to operate in a dynamic environment. The "forgetful" behaviour co-exists with the other path planning behaviours. This behaviour causes the robot to gradually forget information about the environment that has been acquired by other path planning behaviours.

The "forgetful" behaviour is implemented by traversing the quadtree on a regular clock period, and decaying the confidences of all the leaves of the quadtree by a fixed amount. Once the confidences of the quadtree leaves reaches (0) zero, the neighbouring leaves which belong to the same parent quadrant are merged into one quadrant leaf. This has the affect of pruning the quadtree back one level. The new consolidated leaf is regarded as free space. However the confidence in this assumption is 0%.

Refer to Figures 5.12 and 5.13 for an example of the "forgetful" behaviour in combination with the "learn all" behaviour. Figure 5.12 (A) shows an unknown environment and the start (S) and the goal (G) locations. In this example Figures 5.12 (B) - 5.13 (D) show the robot learning the entire structure of the environment while at the same time forgetting portions of the environment it has not sighted for long periods of time. As the robot which is equipped with a tactile sensor, as it navigates towards the goal (G), it acquires more information about the environment and the confidence values of the free space quadrants that have been learnt grow accordingly. While the robot is learning the environment, the forgetful behaviour is simultaneously decaying the acquired knowledge about the environment by 4% every 30 clock ticks. Figure 5.13 (F) shows the robot reaching the goal (G). At this point the "learn all" behaviour will cause the robot to relocate the goal (G) to a quadrant with the lowest confidence value. The robot will then plan a "learn all" path to the new goal. The co-existence of the "learn all" and "forgetful" behaviours causes the robot to operate continuously. This type of behaviour is suitable for a security surveillance robot.

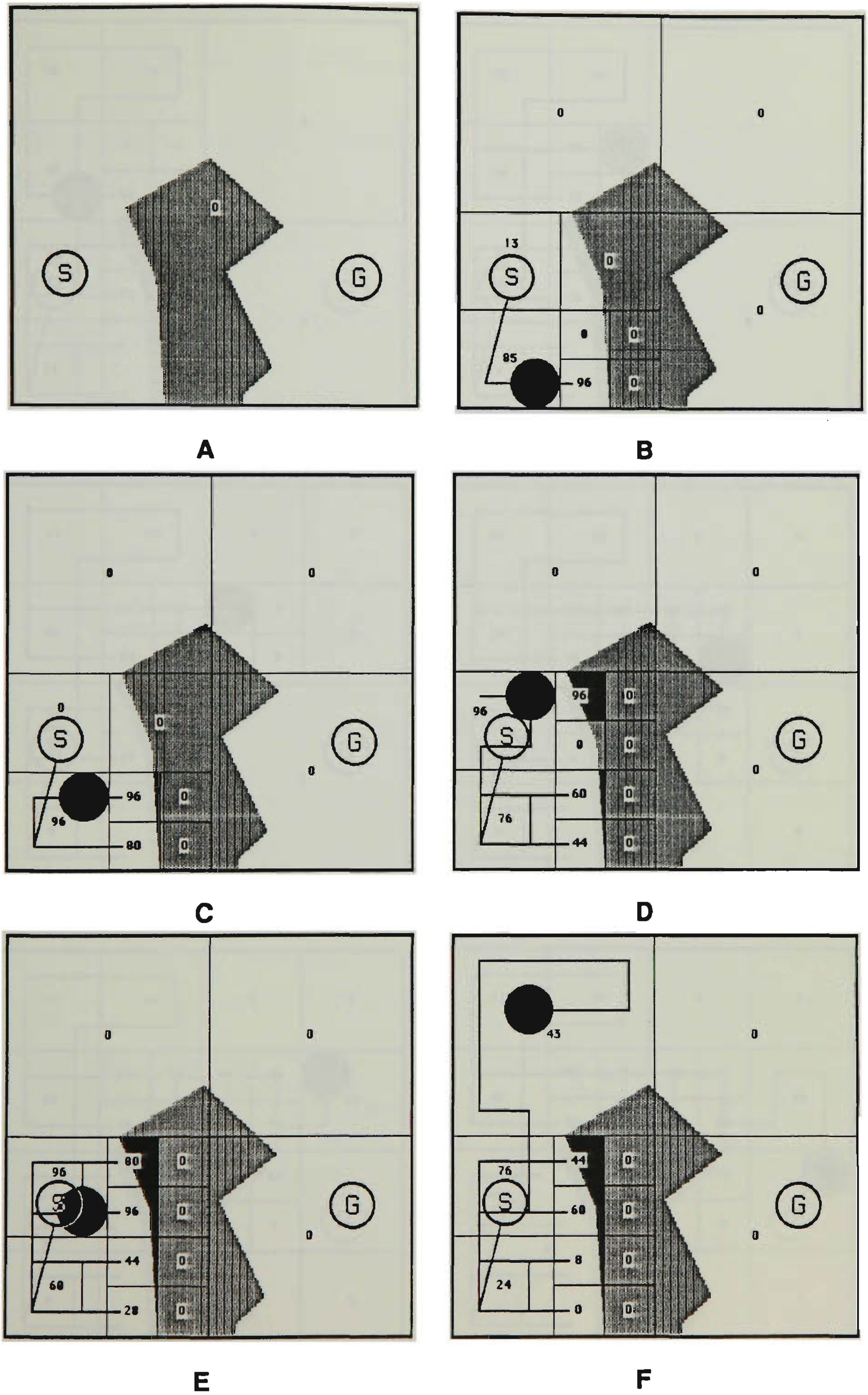


Figure 5.12

Learn all behaviour in combination with the forgetful behaviour.

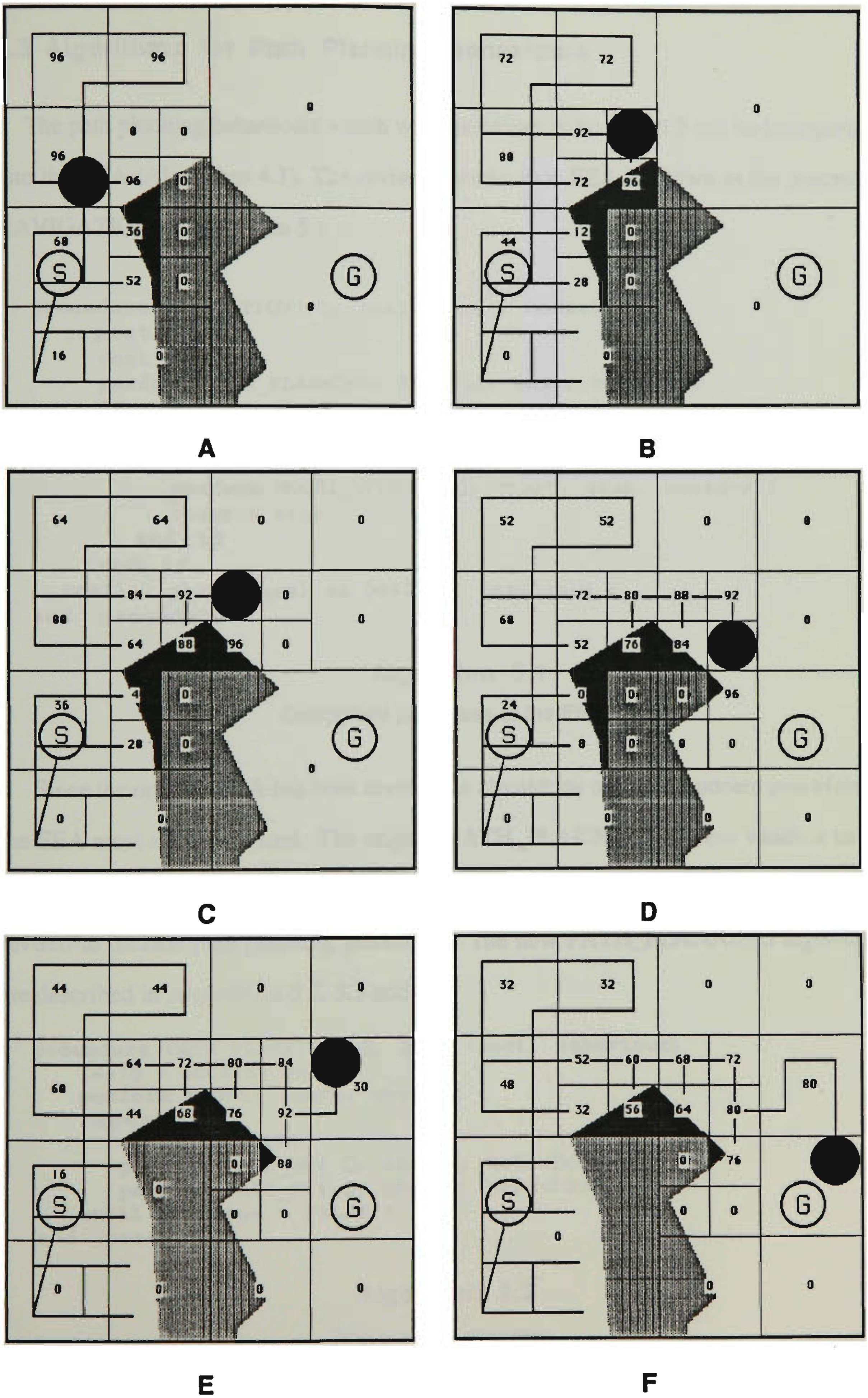


Figure 5.13

Continuation of the experiment which was started in Figure 5.12.

5.3 Algorithms for Path Planning Behaviours

The path planning behaviours which were described in Section 5.2 can be incorporated into the EEA (Algorithm 4.1). The revised pseudo code EEA is shown as the procedure NAVIGATION in Algorithm 5.1.

```
procedure NAVIGATION( Q, start, goal, behaviour )
  repeat
    cost = 0
    perform PATH_PLANNING( Q, goal, cost, behaviour )
    if ( goal reachable ) then
      perform PATH_EXECUTE( Q, start, goal, stop, sensors )
      if ( stop ≠ goal ) then
        perform MODEL_UPDATE( Q, start, stop, sensors )
        start = stop
      end if
    end if
  until ( stop = goal or goal not reachable )
end procedure
```

Algorithm 5.1

Component processes of the EEA.

Since the original EEA has been revised the algorithms of the component processes of the EEA must also be revised. The original PATH_PLANNING process which is based on the fast computation of the distance transform (Algorithms 4.3, 4.4 and 4.5) has been revised to include path planning behaviours. The new PATH_PLANNING algorithms are described in Algorithms 5.2, 5.3 and 5.4.

```
procedure PATH_PLANNING( Q, goal, cost, behaviour)
    leafg = LOCATE( goal )
    perform STORE( leafg, cost )
    repeat
      change = FALSE
      perform PATH_NW( Q, change, behaviour )
      perform PATH_SE( Q, change, behaviour )
    until ( change = FALSE )
end procedure
```

Algorithm 5.2

Path planning algorithm.

```
procedure PATH_NW( Q, change, behaviour )
   if ( GREY( Q ) ) then
      for quadrant = NW, NE, SW and SE do
         perform PATH_NW( SON(Q, quadrant) )
      end for
   else if ( WHITE( Q ) ) then
      direction = EAST
      do while ( direction != WEST )
         perform GET_NEIGHBOURS( Q, direction, neighbours )
         do while ( more neighbours  )
            cost = FACTOR( behaviour, leaf )*SIZE( neighbour )+EXTRACT( Q )
            minimum = EXTRACT( neighbour )
            if ( cost < minimum ) then
               change = TRUE
               perform STORE( neighbour, cost )
            end if
            neighbour = NEXT_NEIGHBOUR( neighbours )
         end do
         direction = NEXT_DIRECTION( direction )
      end do
   end if
end procedure
```

Algorithm 5.3

Propagate the distance transform the NE corner algorithm.

```
procedure PATH_SE( Q, change, behaviour )
   if ( GREY( Q ) ) then
      for quadrant = SE, SW, NE and NW do
         perform PATH_SE( SON(Q, quadrant) )
      end for
   else if ( WHITE( Q ) ) then
      direction = WEST
      do while ( direction != EAST )
         perform GET_NEIGHBOURS( Q, direction, neighbours )
         do while ( more neighbours  )
            cost = FACTOR( behaviour, leaf )*SIZE( neighbour )+EXTRACT( Q )
            minimum = EXTRACT( neighbour )
            if ( cost < minimum ) then
               change = TRUE
               perform STORE( neighbour, cost )
            end if
            neighbour = NEXT_NEIGHBOUR( neighbours )
         end do
         direction = NEXT_DIRECTION( direction )
      end do
   end if
end procedure
```

Algorithm 5.4

Propagate the distance transform the SW corner algorithm.

In Algorithms 5.3 and 5.4 the function FACTOR is the only process which has not been previously described. In this process the function CONFIDENCE extracts the confidence value of a quadtree node. The details of FACTOR are given in Algorithm 5.5.

```
function FACTOR( behaviour, Q )
   if ( BLACK( Q ) ) then
      cost = ∞
   else if ( WHITE( Q ) ) then
      if ( behaviour = OPTIMUM ) then
         cost = 1.0
      else if ( behaviour = CAUTIOUS ) then
         cost = 1.0 + (1 - CONFIDENCE(Q)
      else
         cost = 1.0 + CONFIDENCE(Q)
      endif
   else
      cost = 0.0
   endif
   return ( cost )
end function
```

Algorithm 5.5

Factor algorithm.

During the course of exploring an environment the robot will traverse free space quadrants which have been visited earlier in the robot's journey. A mechanism is needed to update the free space confidence of a quadrant that has been revisited. The free space confidence, which is calculated for the current visit by the robot to the quadrant, must be added to the free space confidence generated on previous visits to the quadrant by the robot. The free space confidence currently stored in a quadrant must be adjusted to describe the free space confidence of the portion of the quadrant not swept by the robot on this visit. This is done by multiplying the confidence value stored in the leaf by the free space confidence of the area not swept by the robot on this visit. Algorithm 5.6 details a procedure called UPDATE_FREE_CONFIDENCE. This procedure describes the calculation of the updated free space confidence of a quadrant given the path length of the current journey through the quadrant and the confidence value currently stored in the quadrant.

```
procedure UPDATE_FREE_CONFIDENCE( quadrant, path_length )
   confidence = [path_length * ROBOT_AREA] / AREA( quadrant )
   new = GET_CONFIDENCE( quadrant ) * [1 - confidence] + confidence
   perform STORE_CONFIDENCE( quadrant, new )
end procedure
```

Algorithm 5.6

Update free space confidence algorithm.

The UPDATE_FREE_CONFIDENCE procedure must be incorporated into the PATH_EXECUTION algorithm which was described in Algorithm 4.6. The revised algorithm is described in Algorithm 5.7.

```
procedure PATH_EXECUTE( Q, start, goal, stop, sensors )
   obstacle = FALSE
   repeat
      leaf = LOCATE( Q, start )
      distance_transform = EXTRACT( leaf )
      if ( distance_transform = 0 ) then
         subgoal  = goal
      else
         subgoal = OPTIMISE( Q, leaf,goal )
      end if
      perform MOVE( subgoal, stop, sensors )
      if ( stop ≠ subgoal ) then
            obstacle = TRUE
      end if
      perform UPDATE_FREE( Q, start, stop )
      start = subgoal
   until ( start = goal or obstacle )
end procedure
```

Algorithm 5.7

Path Execution algorithm.

The update of free space confidence values can also include the information detected by sonar sensors on board the robot. As the sonar maps described in Chapter 2 are grid based, it a straight forward procedure to calculate what percentage of a quadrant is covered with free space cells detected by sonar. This value can be incorporated into the updated confidence value in exactly the same manner as the area swept by the robot during path execution. Examples of this update are provided in Section 5.6.

Once the robot has completed path execution, the navigation system is ready to update the environment map and the confidence model of the environment. Path execution terminates on one of two conditions; either the robot reached the goal or it encountered an obstacle. If the robot reaches the goal, the environment map does not change; however the confidence that the environment is being correctly modelled increases. If the robot encounters an obstacle the environment map is updated to show the presence of the obstacle and the confidence model of the environment is also updated. The general algorithm for updating the quadtree model of the environment was presented in Algorithm 4.10. This algorithm must be revised to include the update of the confidence model of the environment.

Once the algorithm to update the environment models is invoked, it finds the leaf node in the quadtree where the robot is located. This leaf is checked to see if the obstacle sensor readings occur inside this quadrant. If sensor readings occur inside this leaf quadrant then the leaf quadrant is divided into four sub quadrants. Quadrant division continues until the leaf in which the robot is located is isolated from the leaves the sensor readings are in, or the leaf reaches the smallest size resolution allowable. Every time a leaf is divided the free space confidence of the new leaves is calculated by determining what portion of each child quadrant has been swept by the robot. Once the location of the robot has been isolated from the sensor readings and the free space confidences have been updated, the sensor readings detected by the robot are each in turn isolated to a leaf of the smallest size resolution. This operation may require the division of leaf quadrants and the updating of free space confidences.

Isolating the sensor reading to the smallest resolution leaf may be seen as unnecessarily fragmenting the leaves which span obstacles. A consolidation procedure which can detect that neighbouring leaves are part of the same obstacle is called to prune and the quadtree back to the parent of the leaves. Also the consolidation procedure combines the free space confidences of the merged children nodes. This is done by summing up the confidence values of all the children nodes and dividing the sum by (4) four. The revised MODEL_UPDATE algorithm is presented in Algorithm 5.8.

```
procedure MODEL_UPDATE( Q, start, stop, sensors )
   exit = FALSE
   repeat
      leaf = LOCATE( Q, stop )
      if ( SAFE( leaf, sensors )  then
         perform STORE_COLOUR( leaf, WHITE )
         confidence = CALCULATE_CONFIDENCE( start, stop, leaf )
         perform STORE_CONFIDENCE( leaf, confidence )
         exit = TRUE
      else if ( leaf is smallest resolution )  then
         colour = CLASSIFY( leaf, sensors )
         perform STORE_COLOUR( leaf, colour )
         perform STORE_CONFIDENCE( leaf, 1.0 )
         exit = TRUE
      else
         perform STORE_COLOUR( leaf, GREY )
         perform DIVIDE( leaf, child )
         for i = 1 to 4 do
            confidence = CALCULATE_CONFIDENCE( start, stop, leaf )
            perform STORE_CONFIDENCE( child[i], confidence )
         end for
      end if
   until ( exit )
   for ( i = 1 to number of sensor readings ) do
      exit = FALSE
      repeat
         leaf = LOCATE( Q, sensor[i] )
         if ( leaf is smallest resolution )  then
            perform STORE_CONFIDENCE( leaf, 1.0 )
            perform STORE_COLOUR( leaf, BLACK )
            exit = TRUE
         else
            memory = GET_CONFIDENCE( leaf ) / 4
            perform DIVIDE( leaf, child )
            for i = 1 to 4 do
               perform STORE_CONFIDENCE( child[i], memory )
            end for
         end if
      until ( exit )
   end for
   perform CONSOLIDATE( Q )
end procedure
```

Algorithm 5.8

Model Update algorithm.

5.4 Planning Safe Paths

Section 3.3.6 discussed the problem of considering the safety of a robot during path planning. This section presents a method for planning paths that take into account robot safety. The method is based upon an extension to the distance transform methodology of path planning. This section will show how path planning with robot safety criteria can be incorporated into EEA and the [Jarvis *et. al.* 86] grid based path planner.

The distance transform can be modified to include distance from obstacles safety information. Conceptually the distance transform is equivalent to dropping the goal "pebble" into the environment "pond" and watching the resulting wave front flow around obstacles and eventually through all free space in the environment. The "pebble in the pond" concept can be extended further, by observing that as the wave front flows into obstacles, some of the wave front is reflected back. In other words the obstacles are exerting a repulsive potential, and the strength of this potential varies inversely with the distance from the obstacle. The distance transform can be inverted into an "obstacle transform" where the obstacle cells become the goals. The resulting transformation yields for each free cell in the data structure the minimal distance from the centre of the free space cell to the boundary of an obstacle cell. Refer to Figure 5.14 for an example of the obstacle transform. Figure 5.14 (A) shows the grid based obstacle transform and Figure 5.14 (B) shows the quadtree based obstacle transform. The obstacle transform values for the quadtree represent the distance to the nearest obstacle from the centre of the quadrant. The grid based obstacle transform values represent the distance to the nearest obstacle from the furthest boundary of the grid cell. To determine the distance to nearest obstacle from the centre of any grid cell, subtract 0.5 from the obstacle transform value.

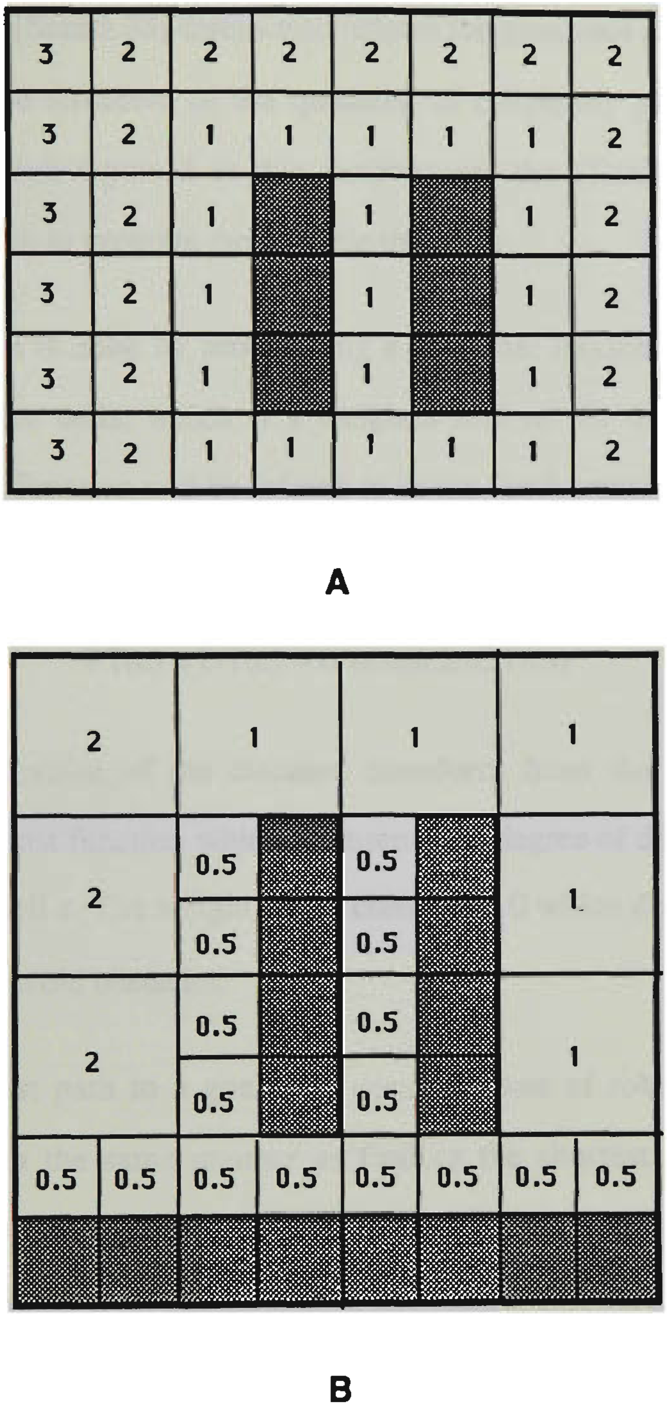


Figure 5.14

Obstacle Transforms grid based and quadtree based.

Prior to planning a path a preprocessing step generates the obstacle transform. In the grid based distance transform, this is a straight forward affair. The obstacles cells are marked as goals, and the distance transform is calculated. The use of such a preprocessing step in quadtree based path planning was first described by [Kambhampati *et. al.* 86]. This approach used the obstacle transform information together with the A* algorithm to plan safe paths. The preprocessing algorithm used by [Kambhampati *et. al.*

86] is a variant of the [Samet 88] distance transform for quadtrees algorithm. The Samet algorithm exploits the structure of the quadtree to efficiently generate the obstacle transform. The research reported in this section uses the [Kambhampati *et. al.* 86] preprocessing algorithm to generate the obstacle transform.

Planning safe paths is done by propagating a new cost function from the goal cell through the free space cells, which is a weighed sum of the distance and obstacle transforms. This cost function will be refered to as the "path transform" (*PT*). The path transform for a cell *c* is defined as:

$$\mathrm{PT(c) = DT(c) + \alpha\ obstacle(OT(c))}$$

where *DT(c)* is the value of the distance transform from the goal. The function *obstacle(OT(c))* is a cost function which represents the degree of discomfort the nearest obstacle exerts on a cell *c*. The weight α is a constant ≥ 0 which determines by how far the solution path will avoid obstacles.

Finding the shortest path to a goal with consideration of robot safety using path transforms is done in the same manner as finding the shortest path using distance transforms, by following the steepest descent path of the path transform. The path transform unlike potential field planning does not yield a transform with local minima, because all the costs of paths to the goal from each cell are calculated. The path transform value stored for each cell is the minimum propagated path cost to the goal. Examples of the path transform applied to grids and quadtrees with different values of α are shown in Figures 5.15 and 5.16 respectively. The lightly shaded quadrants in both figures indicate the solution path from a start location (S) to a goal location (G). Figures 5.15 (A) and 5.16 (A) show the path transform with α = 0.0. This results in a normal distance transform where the shortest path to the goal is best. Figures 5.15 (B) and 5.16 (B) shows the path transform with α = 0.5. This results in a solution path which deviates around the obstacles, and takes a safer path to the goal. Figures 5.15 (C) and 5.16 (C) show the path transform with α = 1.0. This results in a solution path which deviates further around the obstacles, and takes the safest path to the goal.

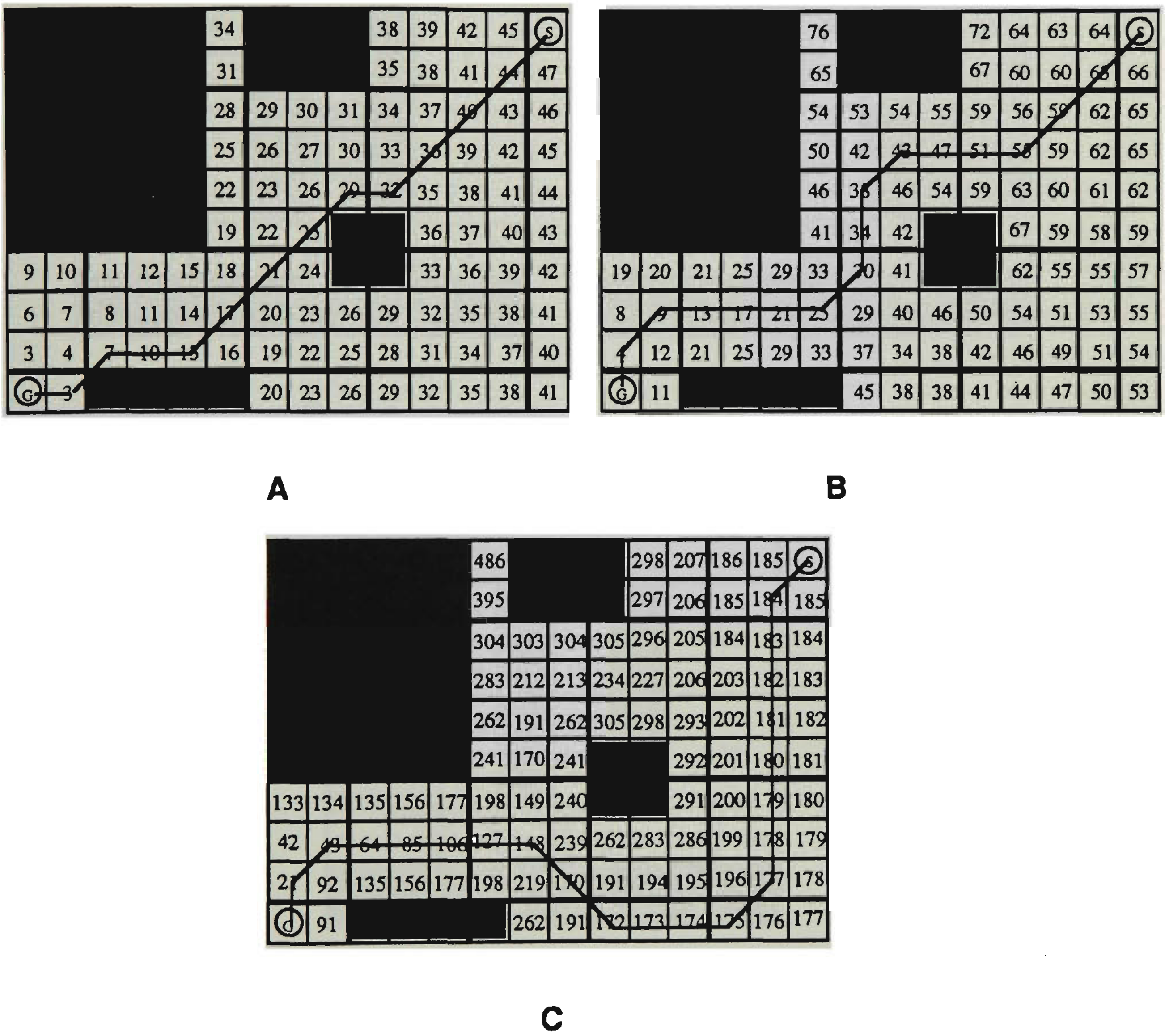


Figure 5.15

The path transform applied to grids with different values of α.

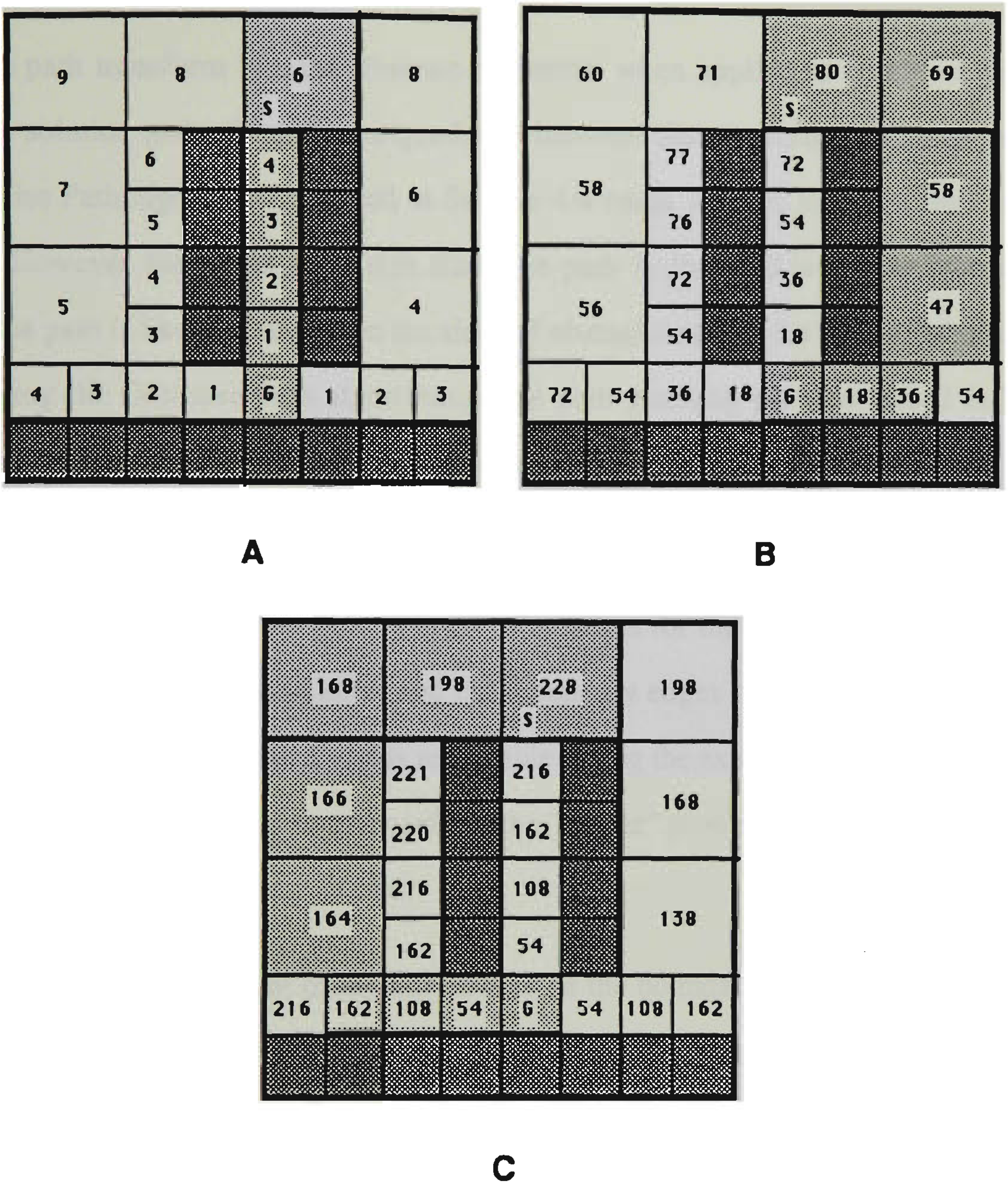


Figure 5.16

The path transform applied to quadtrees with different values of α.

The path transform like the distance transform when applied to quadtrees yields a coarse solution path of free space quadrants between the start and goal locations. The Optimise Path algorithm presented in Section 4.4 could be used to find a fine solution path. However since this algorithm finds the path to the goal which is shortest, the solution path is likely to run down the sides of obstacles or clip the corners of obstacles. Applying the Optimise Path algorithm to the path planning problem posed in Figure 5.16(b) will generate a path which runs along the edges of the obstacle. Using such a strategy defeats the whole purpose of generating the path transform.

An alternate strategy for generating a safe path is for the robot to steer through the middle of the intersection of entry and exit boundary edges of quadrants. This strategy generates relatively safe paths, and is reasonable during the exploration of an environment by the robot. However this strategy suffers the "too far" problem, in situations where the free space quadrants are large in size.

Choosing the mid point of the intersection of the boundary edges of quadrants does not guarantee that this point is the safest point along the intersection edge. The safest point on the intersection edge is found by examining the obstacle transform values of the quadrants which are adjacent and normal to the intersection edge. The obstacle transform values represent the distance from the centre of the quadrant to the nearest obstacle. While it is not possible to calculate the exact distances the intersection edge is from the nearest obstacle, an acceptable approximation can be made. The safest point is located at the point which lies in between the obstacle transform values. Refer to Figure 5.17 for an example of a fine solution path which is generated by using the safest points on the intersection edge between the entry and exit edges of solution path quadrants. The obstacle transform values of each quadrant are displayed, together with the safety values of the safest points. The safest points are marked with the heavy crosses. The obstacle transform values of adjacent quadrants which do not exist are assumed to be (0) zero. This only exists for quadrants which are located on the outside perimeter of the quadtree. This strategy keeps the solution path away from the outside boundaries of the quadtree.

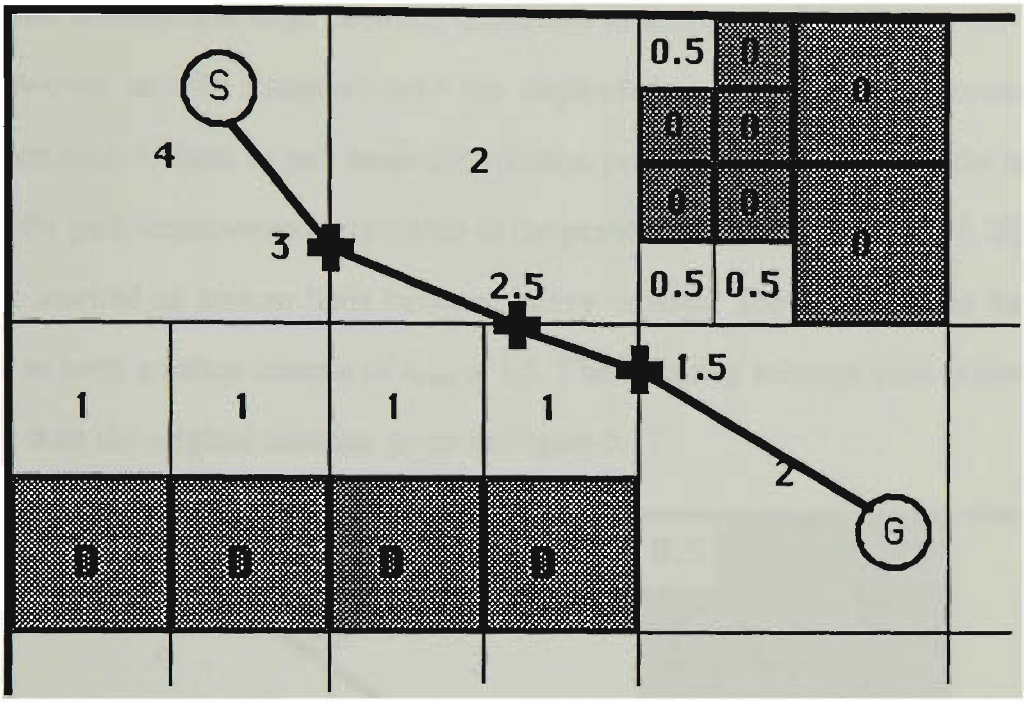


Figure 5.17

The fine solution path generated by using the safest points on the intersection edges.

The fine solution path which is described by a sequence of safest points along the intersection edge of quadrants suffers the "too far" problem, and tends to make unnecessary detours. The solution path is in need of shortening and straightening.

The review of planning "safe" paths in Section 3.3.6 discussed the path improvement technique reported by [Thorpe 84] called "path relaxation". This technique was based upon sliding points of the solution path a small distance, and recomputing the safety cost of the path. This procedure of sliding and recomputing the cost of the solution path was iterated until the change in path cost between iterations converged to a small difference.

The approach used by Thorpe could be applied to this problem. However the computational cost of iterating until a solution emerges is high. Instead, an improved path can be achieved by relaxing the safety criteria of the safest points. The user decides what is the desirable minimum distance s_{min} the solution path can safely approach an obstacle. The safest point is expanded along the intersection edge, such that all points along the expanded safety interval are at least s_{min} from any obstacle. The Optimise Path process (Algorithm 4.7) can now be used to find a fine solution path. Normally this algorithm uses the full intersection edge between quadrants to shorten and straighten the solution path. However in this situation only the expanded safety edge component of the intersection edge is used to pull ~~taught~~ the solution path. Figure 5.18 shows the results of taut applying the path improvement algorithm to the problem posed in Figure 5.17. The safety edges are marked as broken lines between heavy crosses. The safety edges have been specified to have a safety criteria of s_{min} = 1.5. The resulting solution path is shorter and straighter than the original solution given in Figure 5.17.

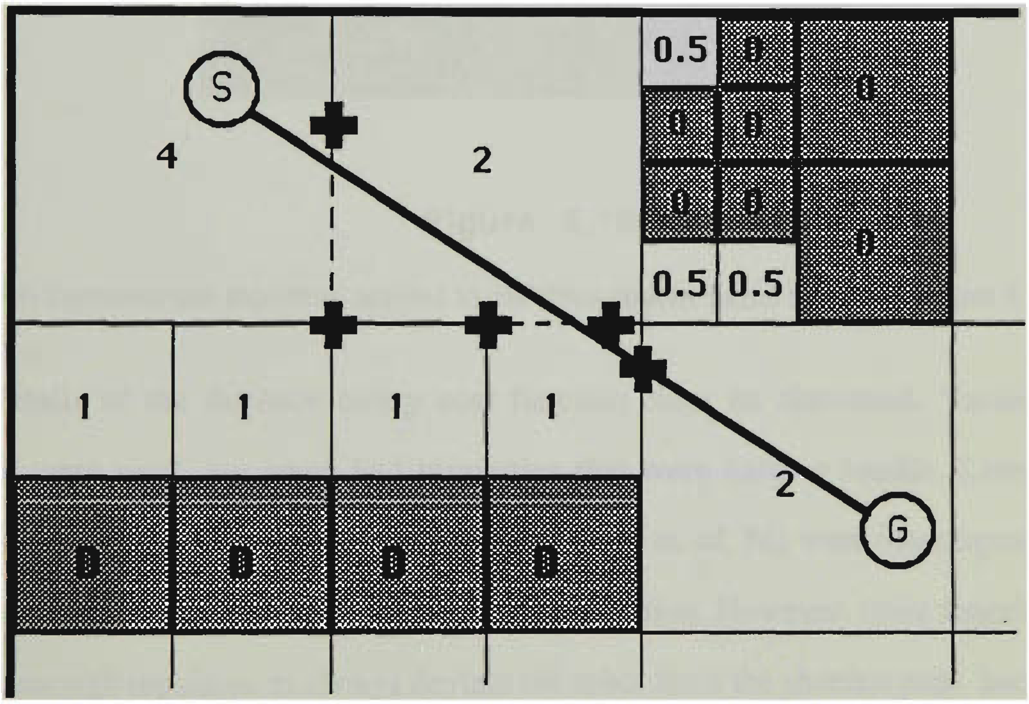


Figure 5.18

The fine solution path generated using the expanded safety edge and the Optimise Path Algorithm.

Applying the path improvement algorithm to the problem of finding "safe" paths posed in Figure 5.16 results in the solutions given in Figure 5.19. This figure illustrates the path improvement algorithm applied to the three ~~course~~ paths generated with α = 0.0, 0.5 and coarse 1.0. The safety edges have been specified to have a safety criteria of s_{min} = 1.0.

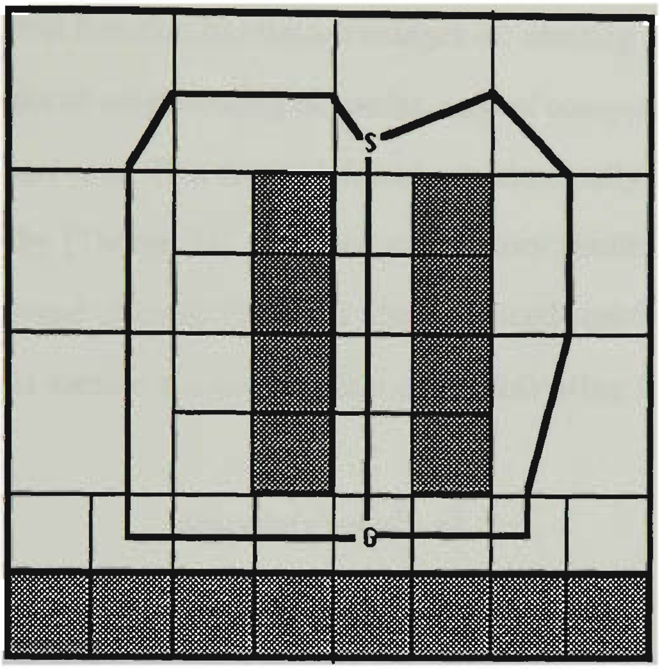


Figure 5.19

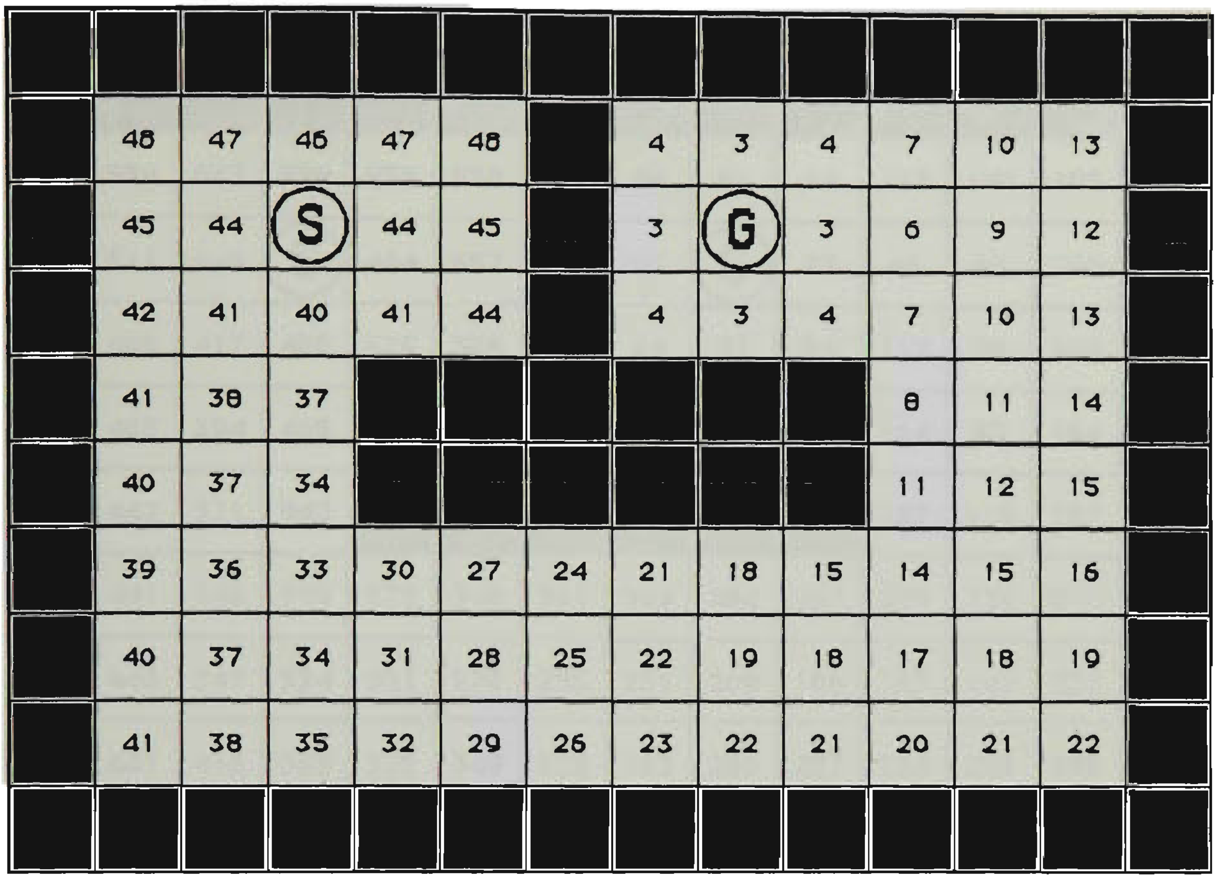
The path improvement algorithm applied to the three ~~course~~ paths shown in Figure 5.16. coarse

The details of the distance safety cost function must be discussed. Various cost functions were used, but some had properties that were hard to handle. Linear cost functions similar to those reported by [Kambhampati *et. al.* 86] were investigated first. Such functions have the advantage of ease of computation. However, these functions did not exert enough repulsion to always deviate the robot from the shortest path. Increasing the slope of the cost function reduced the distance over which the function exerted an influence. To remedy this situation exponential functions were used. These functions had the desired effect of altering the solution path. However, the problem with exponential functions is that they never reach zero, so all the objects have an effect on the solution path, even when the robot is located at significant distances from obstacles. To solve this problem the exponential function was truncated at a fixed distance, but this left furrows or trenches in the path cost function, and the solution paths tended to get caught in these furrows. Similar experiences with these types of cost functions were reported by [Thorpe 84]. Thorpe found that the most effective cost function was a cubic function that ranges from zero at some maximum distance, set by the user, to the obstacles maximum cost at zero distance. Such a cost function has the advantages of: creating a saddle between the repulsion potential peaks of neighbouring obstacles, ease of computation, and having its effects bounded in a local area. This research found experimentally that a cubic function of the type suggested by [Thorpe 84] produced satisfactory results. Experiments using quadratic functions instead of cubic functions also produced satisfactory results. In the results presented in this section a cubic function of the following form was used as the function *obstacle*:

$$\text{obstacle}(x) \ = a^3 - x^3$$

where x is the value of the obstacle transform for a free space quadrant, and a. is the maximum range of the effect of the ***obstacle*** function. In the results reported in this section a was set to (4) four.

An interesting spin-off of the path transform when applied to grids is that it forms a better contour path for a robot to execute the behaviour of "visit all" path planning than the contour path generated by the distance transform. The distance transform forms circular contour patterns which radiate from the goal points. The path transform, on the other hand, forms contour patterns which slope towards the goal, but also follow the shape profile of obstacles in the environment. Figures 5.20 and 5.21 show the distance and path transforms for a path planning problem in an indoor environment. These figures also show the robot execution paths for the "visit all" path planning behaviour for each form of transform. The path transform produces the better execution path, since it produces a path which has less turns and has more path segments that are straight. The distance transform in Figure 5.20 produces a path which requires 37 turns and has an average length of 2.00 units for each path segment. In contrast the path transform in Figure 5.21 produces a path which requires 19 turns and has an average length of 4.05 units for each path segment.



■	■	■	■	■	■	■	■	■	■	■	■	■	■
■	48	47	46	47	48	■	4	3	4	7	10	13	■
■	45	44	S	44	45	■	3	G	3	6	9	12	■
■	42	41	40	41	44	■	4	3	4	7	10	13	■
■	41	38	37	■	■	■	■	■	■	8	11	14	■
■	40	37	34	■	■	■	■	■	■	11	12	15	■
■	39	36	33	30	27	24	21	18	15	14	15	16	■
■	40	37	34	31	28	25	22	19	18	17	18	19	■
■	41	38	35	32	29	26	23	22	21	20	21	22	■
■	■	■	■	■	■	■	■	■	■	■	■	■	■

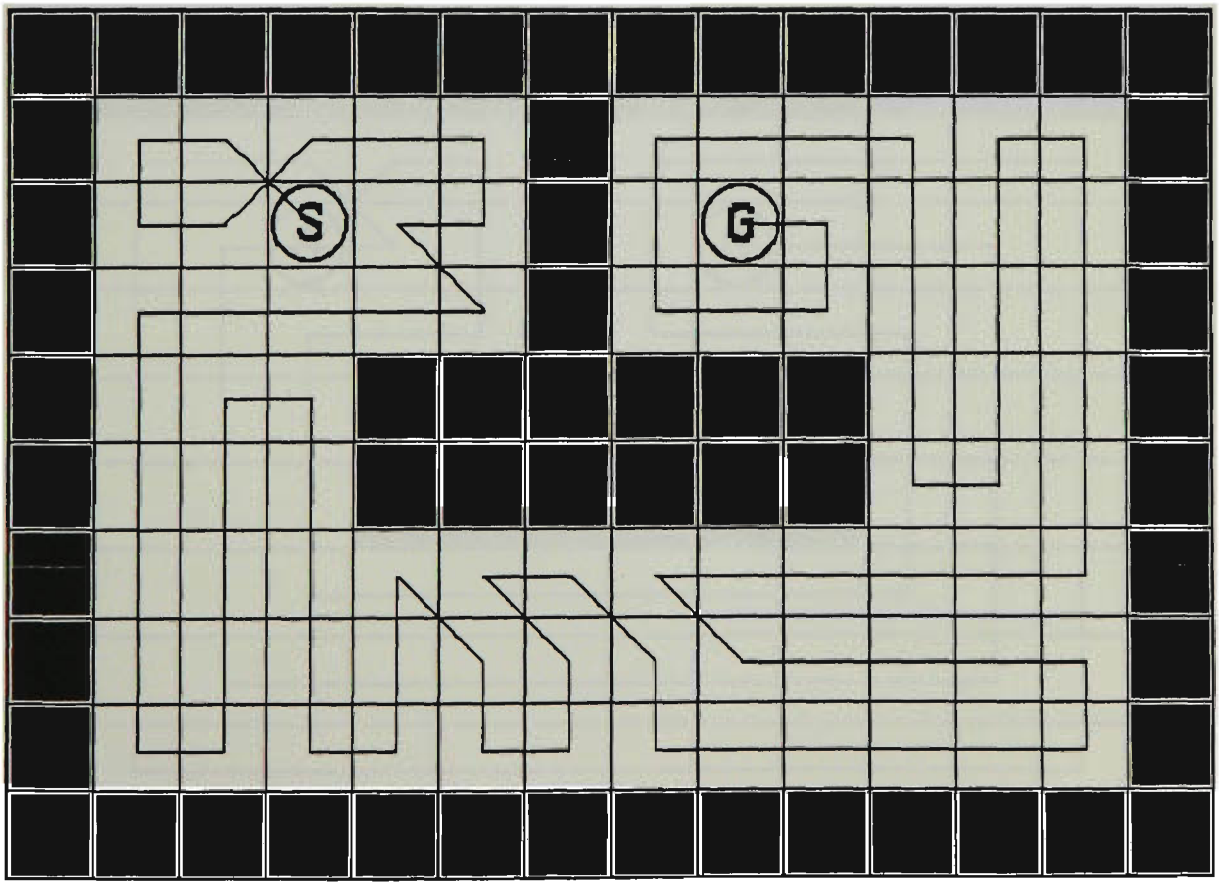
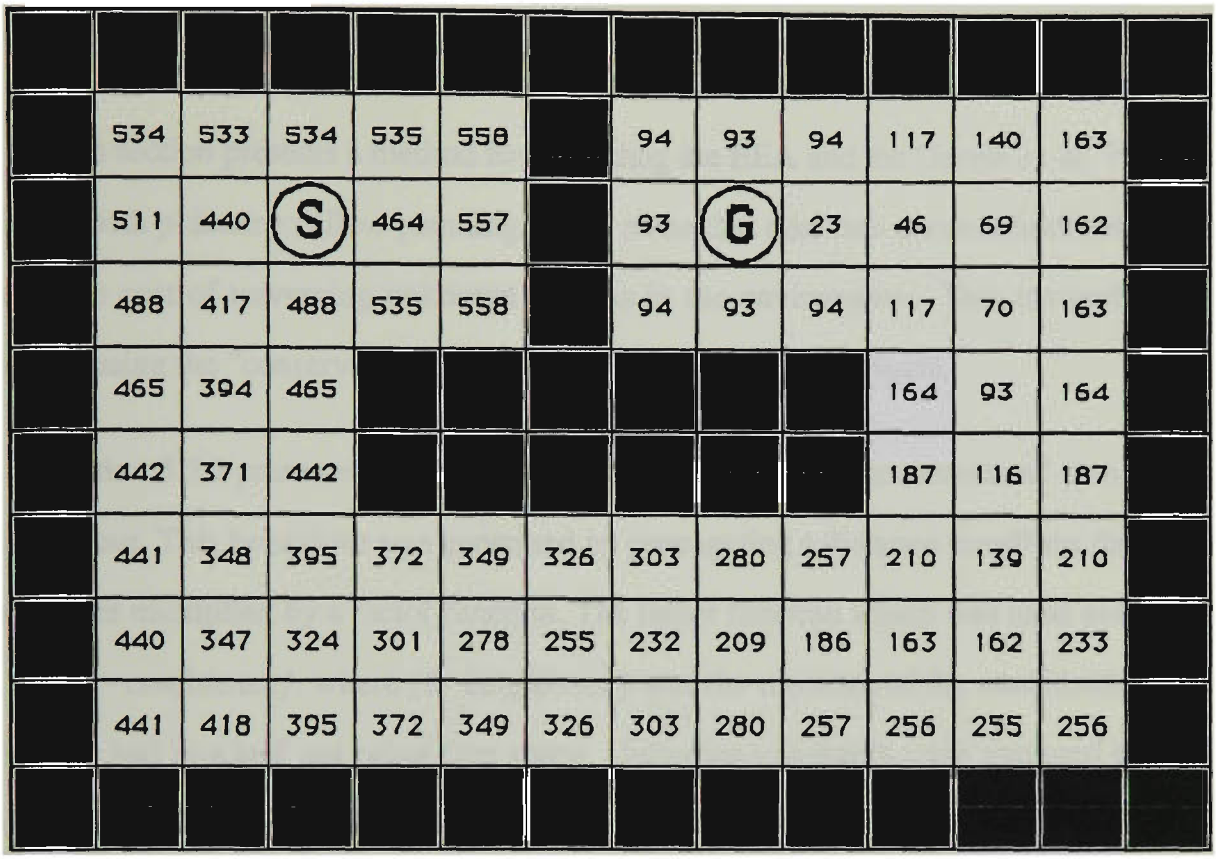


Figure 5.20

Visit all path planning behaviour using the distance transform.



■	■	■	■	■	■	■	■	■	■	■	■	■	■
■	534	533	534	535	558	■	94	93	94	117	140	163	■
■	511	440	S	464	557	■	93	G	23	46	69	162	■
■	488	417	488	535	558	■	94	93	94	117	70	163	■
■	465	394	465	■	■	■	■	■	■	164	93	164	■
■	442	371	442	■	■	■	■	■	■	187	116	187	■
■	441	348	395	372	349	326	303	280	257	210	139	210	■
■	440	347	324	301	278	255	232	209	186	163	162	233	■
■	441	418	395	372	349	326	303	280	257	256	255	256	■
■	■	■	■	■	■	■	■	■	■	■	■	■	■

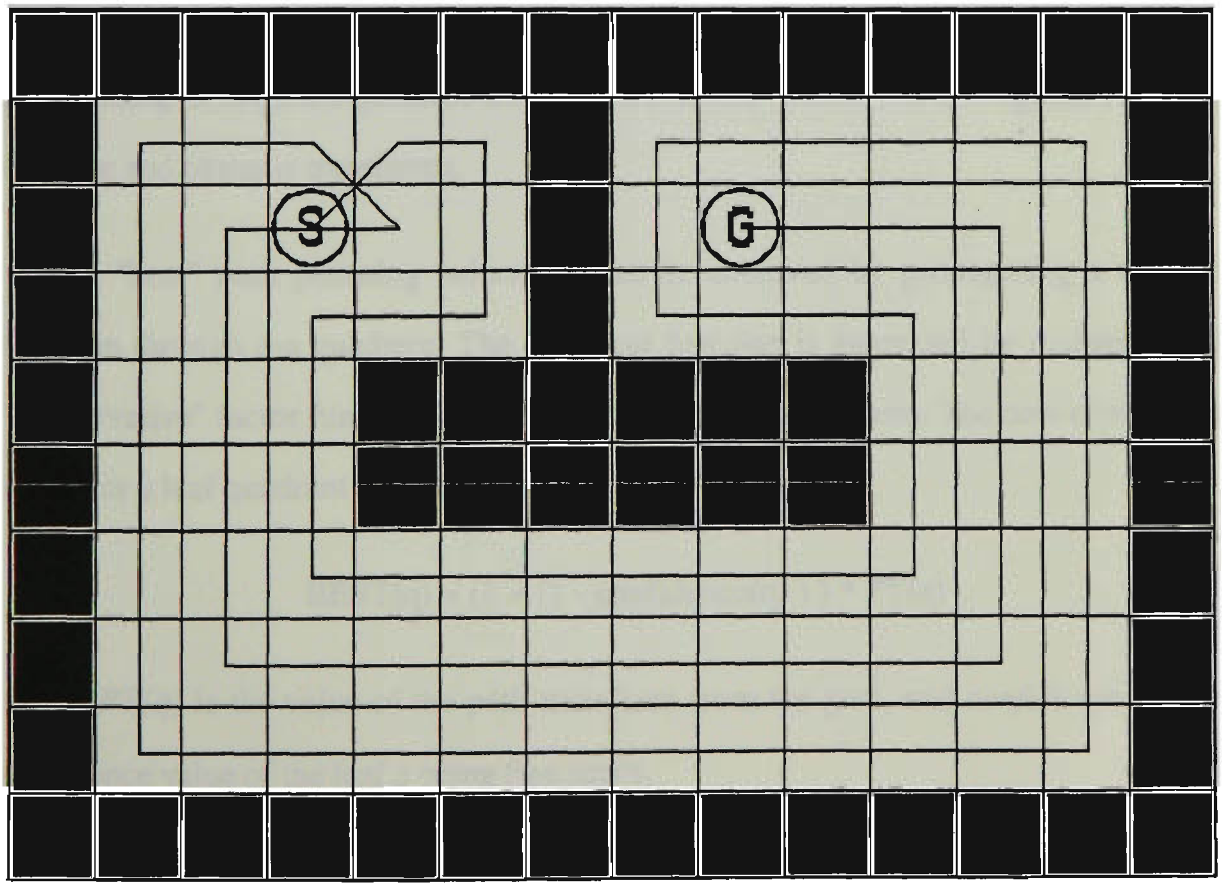


Figure 5.21

Visit all path planning behaviour using the path transform.

5.5 Planning Best Paths

This section presents a method for extending the EEA and the [Jarvis *et. al.* 1986] grid based path planner to allow planning "best" paths that take into account both robot safety and the cost of traversing unknown regions in the environment. This method is based upon fusing the "conservative" and "safe" path planning behaviours.

Section 5.5.2 presented the mechanism for generating the "conservative" path planning behaviour. This behaviour was generated by propagating a distance transform through the quadtree multiplied by a factor function. The factor function which was used evaluated to *1 + [1 - confidence]*, where *[1- confidence]* was the measure of the confidence that the system had in a leaf *not* being free space. Unknown quadrants were assumed to be free spaces, with zero (0) confidence.

Section 5.4 presented the mechanism for planning "safe" paths. This was done by propagating through the quadtree the "path transform" which was a weighted sum of the distance and obstacle transforms.

The "best" path planning behaviour can be achieved by propagating a new cost function through the quadtree. The new cost function is generated by multiplying the "conservative" factor function by the values of the path transform. The new cost function *BEST* for a leaf quadrant *q* is defined as:

$$\mathrm{BEST(q)} = (1 + (1 - \mathrm{confidence(q)}\,)\,) * \mathrm{PT(q)}$$

where *PT(q)* is the value of the path transform from the goal, and *confidence(q)* is the confidence value of the leaf *q* being free space.

Once the transform for "best" path planning has been generated, a fine solution path must be found, using the path improvement algorithm described in Section 5.4. Figure 5.22 shows an example of planning "best" paths between a start (S) and a goal (G). Figure 5.22 (A) shows the confidence values the EEA has in the free space quadrants prior to path planning. Figure 5.22 (B) shows the transform associated with the "best" path planning behaviour using a safety weighting of $\alpha = 0.5$. Figure 5.22 (B) also shows the fine solution path which was found using a safety criteria of $s_{min} = 1.0$ and the zigzag rule (Section 4.4.1).

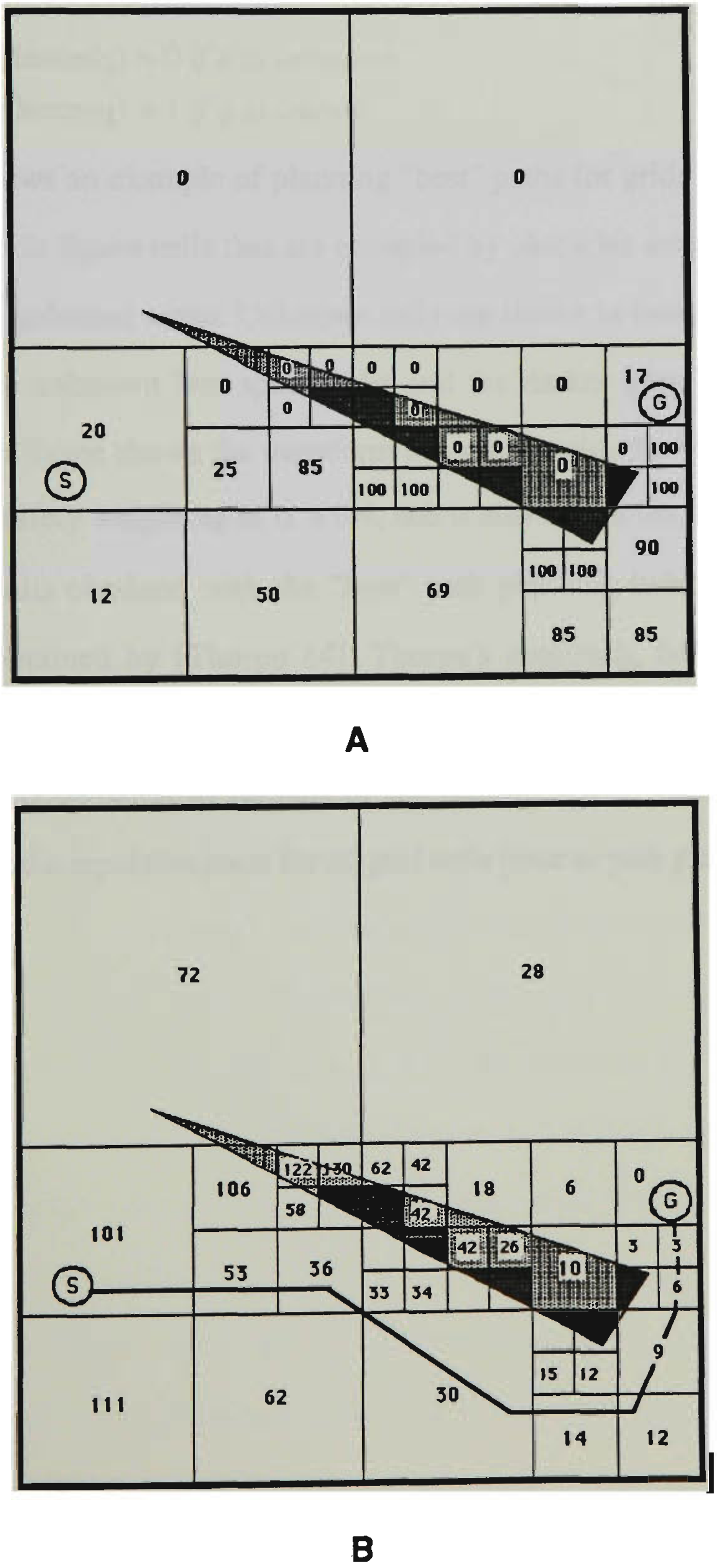


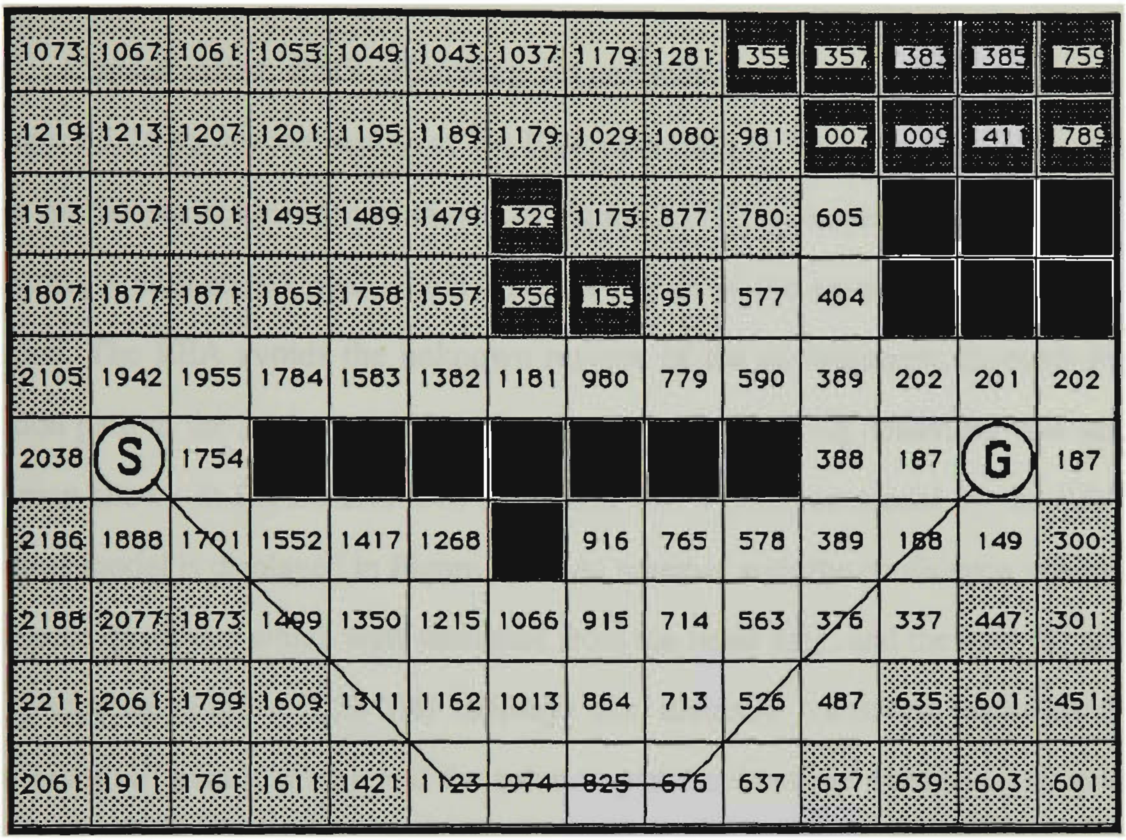
Figure 5.22

Best path planning behaviour with quadtrees.

The cost function BEST which used to compute "best" paths for quadtrees can be used to compute the "best" paths for grids. The function *confidence(q)* is evaluated in a different manner. In the grid data structure grid cells are either known or unknown, therefore the *confidence(q)* is evaluated in the following manner:

confidence(q) = 0 *if q is unknown*
confidence(q) = 1 *if q is known*

Figure 5.23 shows an example of planning "best" paths for grids between a start (S) and a goal (G). In this figure cells that are occupied by obstacles are coloured black and free space cells are coloured white. Unknown cells are shown in two shades of grey. The light grey cells are unknown free space cells and the darker grey cells are unknown obstacle cells. This figure shows the transform associated with the "best" path planning behaviour using a safety weighting of $\alpha = 0.4$, and it also shows the fine solution path to the goal. The results obtained with the "best" path planning behaviour for grids are similar to those obtained by [Thorpe 84]. Thorpe's approach, based on an A* type algorithm, was reviewed in Section 3.3.6. Thorpe searches for the nearest obstacle to the grid cell which is considered to be part of the solution path, while the "best" path approach computes the repulsion costs for all grid cells prior to path planning.



1073	1067	1061	1055	1049	1043	1037	1179	1281	1355	1357	1383	1385	1759
1219	1213	1207	1201	1195	1189	1179	1029	1080	981	1007	1009	1411	1789
1513	1507	1501	1495	1489	1479	1329	1175	877	780	605			
1807	1877	1871	1865	1758	1557	1356	1155	951	577	404			
2105	1942	1955	1784	1583	1382	1181	980	779	590	389	202	201	202
2038	S	1754								388	187	G	187
2186	1888	1701	1552	1417	1268		916	765	578	389	188	149	300
2188	2077	1873	1499	1350	1215	1066	915	714	563	376	337	447	301
2211	2061	1799	1609	1311	1162	1013	864	713	526	487	635	601	451
2061	1911	1761	1611	1421	1123	974	825	676	637	637	639	603	601

Figure 5.23

Best path planning behaviour with grids.

5.6 Experimental Results

The preceding sections of this chapter described eight different path planning behaviours, these being: "optimum", "adventurous", "conservative", "learn all", "visit all", "forgetful", "safest" and "best" path planning behaviours. This section presents one result for each type of path planning behaviour, with the exception of the "optimum" path behaviour for which results were presented in Chapter 4. The seven experimental results are presented in separate subsections. Each experiment uses the sonar data which was collected in Chapter 2. Chapter 2 presented four experimental results of map making (namely Figures 2.17 - 2.20). These four results have been used as input to the extended EEA. The extensions to the navigation algorithm to include path planning behaviours were implemented on a Macintosh II microcomputer.

5.6.1 Conservative Path Planning Experiment

Figure 5.24 shows the results of the EEA operating in "conservative" path planning mode on the environment map shown in Figure 2.17. This experiment shows the EEA favouring a path to the goal which lies in a region of the environment that has been mapped. The EEA avoids the unknown regions of the environment, through which a solution path to the goal passes if the "optimum" path planning behaviour was selected. The map shown in Figure 2.17 was converted into a quadtree model by the EEA. The quadtree model is displayed in Figure 5.24 (A) together with the confidence values of the free space quadrants which were extracted from the sonar map, and the start (S) and goal (G) positions. Figure 5.24 (B) displays the distance transform associated with "conservative" path planning and the solution path between the start and goal positions using FULL path optimisation (Algorithm 4.7) and the zigzag rule (Section 4.4).

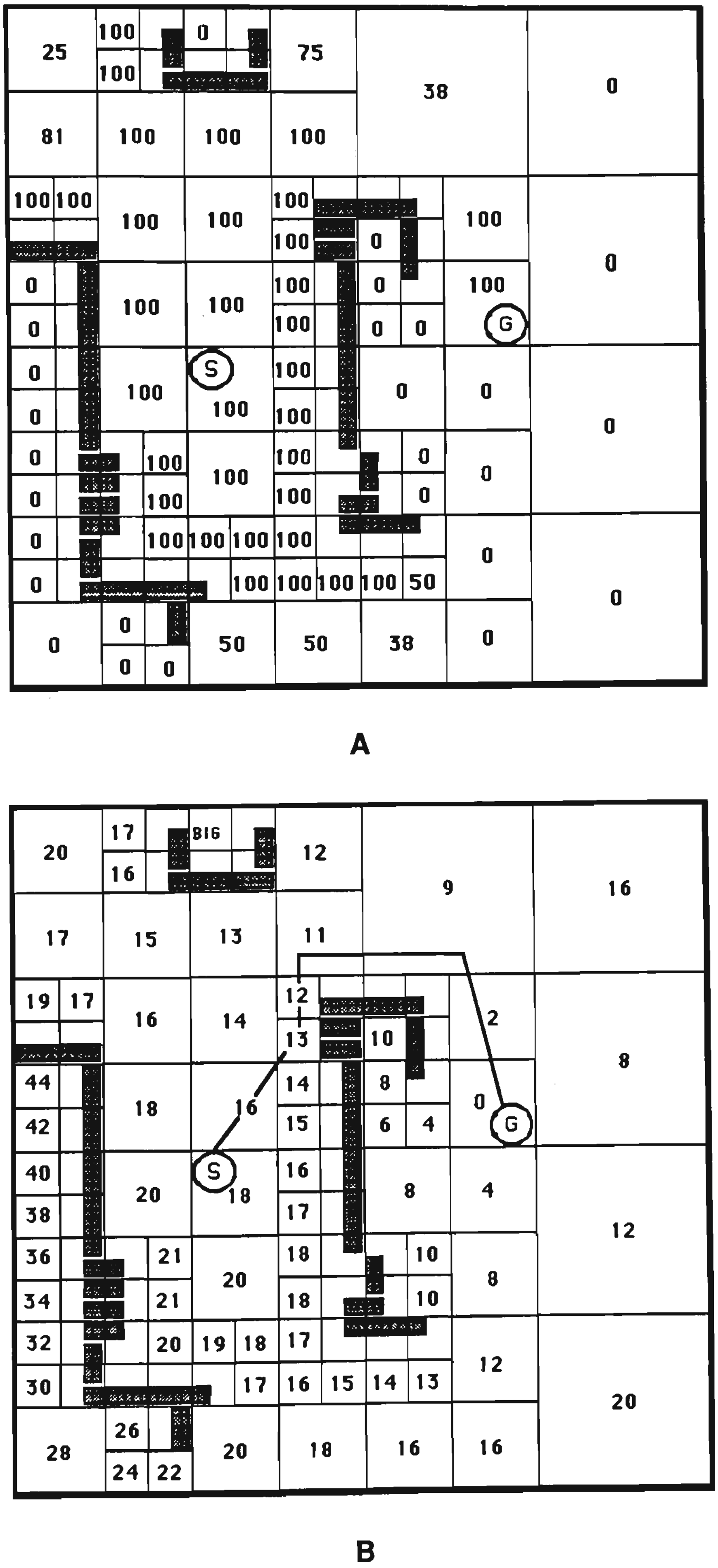


Figure 5.24

Conservative path planning behaviour.

5.6.2 Adventurous Path Planning Experiment

Figures 5.25 and 5.26 show the results of the EEA operating in "adventurous" path planning behaviour mode on the environment map shown in Figure 2.17. This experiment uses the same data the "conservative" path planning experiment used (Section 5.6.1). This experiment shows how the EEA deliberately avoids mapped regions, and instead favours a path to a goal which passes through unknown regions. The quadtree model is displayed in Figure 5.25 (A) together with the confidence values of the free space quadrants which were extracted from the sonar map, and the start (S) and goal (G) locations. Figure 5.25 (B) displays the distance transform associated with "adventurous" path planning and the solution path between the start and goal. The solution path is computed using the using two (2) quadrant REDUCED look ahead path optimisation (Algorithm 4.8) and the zigzag rule.

During the course of path execution to the goal the robot encounters an obstacle at the position R as shown in Figure 5.25 (B). At this point the EEA instructs the mobile robot to collect more sonar range readings and build a new map. From the new map data the quadtree is reconstructed with new confidence values as shown in Figure 5.26 (A). The "adventurous" distance transform is recomputed using the reconstructed quadtree, and a new path is planned from the position R to the goal G as shown in Figure 5.26 (B). The new path is computed using the using the two (2) quadrant REDUCED look ahead path optimisation function and the zigzag rule.

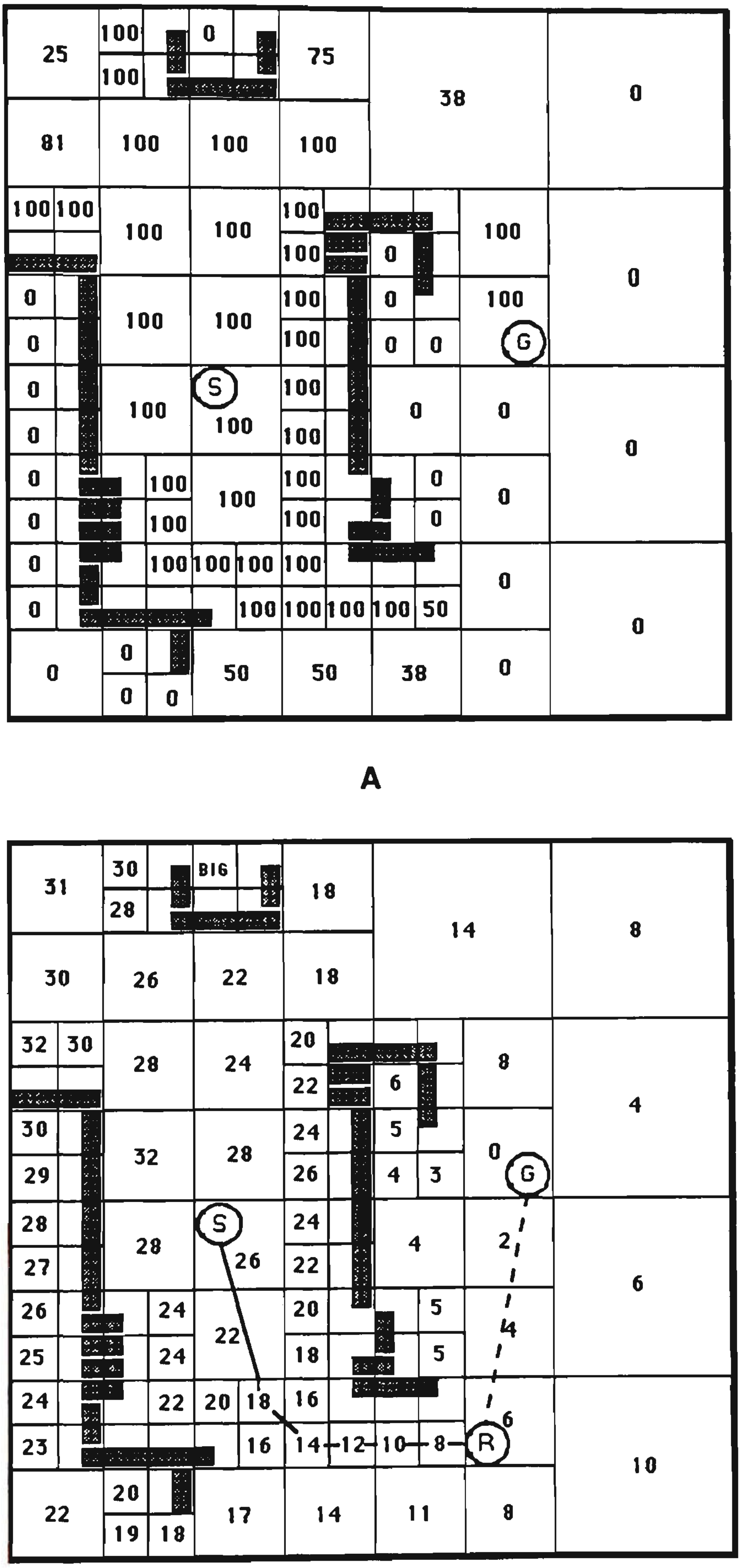


Figure 5.25

Adventurous path planning behaviour.

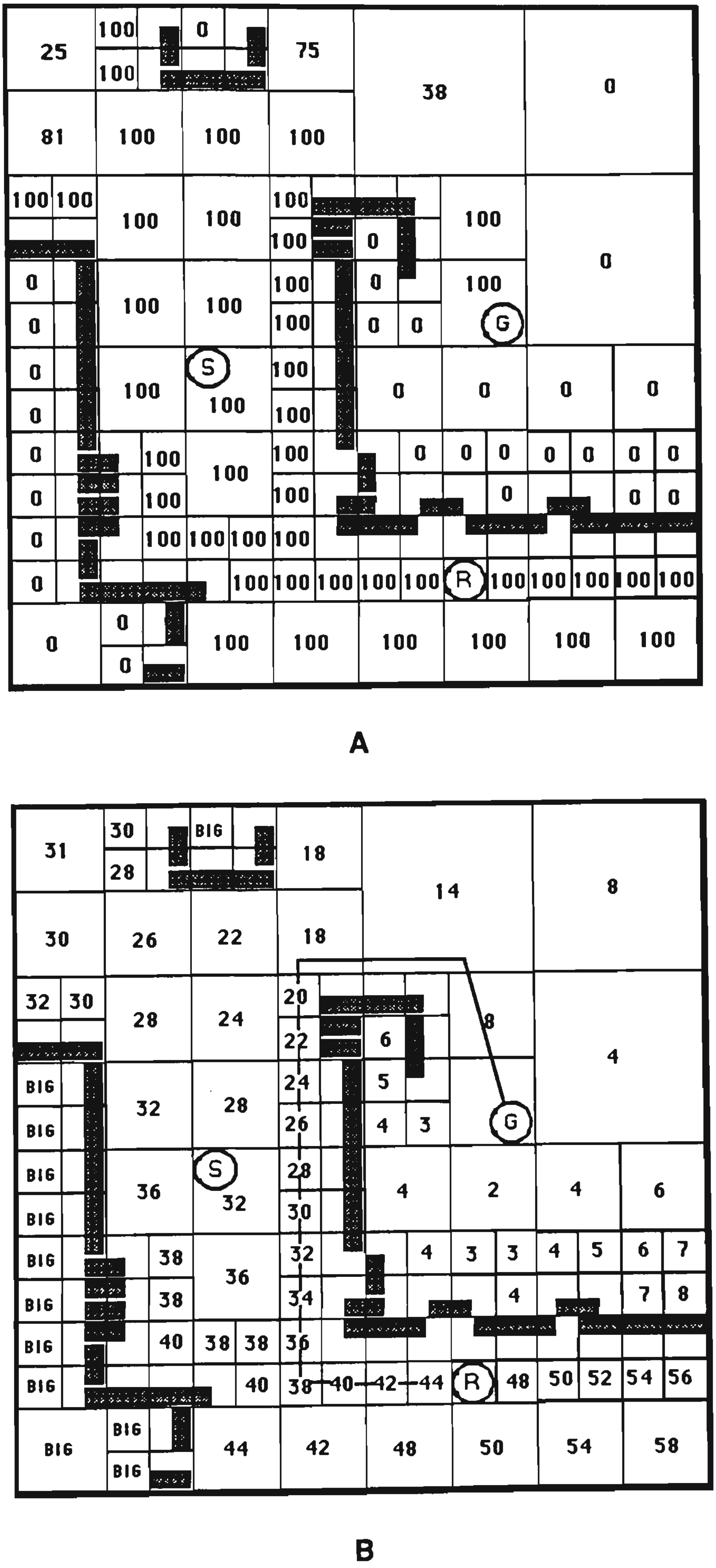


Figure 5.26

Continuation of the Adventurous path planning behaviour example started in Figure 5.25.

5.6.3 Learn All Path Planning Experiment

Figures 5.27 - 5.30 show the results of the EEA operating in "learn all" path planning behaviour mode on the environment map shown in Figure 2.20. Diagram (A) of each figure shows the growing confidence values of quadrants in the quadtree as the environment is progressively learnt. Diagram (B) of each figure shows the "learn all" distance transform for the updated quadtree model. The quadtree model is displayed in Figure 5.27 (A) together with the confidence values of the free space quadrants which were extracted from the sonar map, and the start (S) and goal (G) locations. Figure 5.27 (B) displays the distance transform and the "learn all" path between the start and goal positions.

Once the goal specified in Figure 5.27 has been reached the EEA relocates the goal into the quadrant with the lowest confidence value, and a "learn all" path is generated from the current robot location to the new goal. Figure 5.28 shows the new goal position and the robot's "learn all" path to the new goal. During the course of the "learn all" behaviour the robot encounters an obstacle as it tries to reach the goal specified in Figure 5.29. At this point the EEA instructs the mobile robot to collect more sonar range readings, build a new map and reconstruct the quadtree with new confidence values, as shown in Figure 5.30 (A). Since the goal was located in a position which was occupied by an obstacle, it is assumed that the goal has been reached, and the goal is relocated as shown in Figure 5.30 (A). A new "learn all" path is planned and executed from the current robot location to the relocated goal. Once the robot has reached this goal, it has completely learnt the environment. At this point it proceeds directly to the first goal that was specified in Figure 5.27.

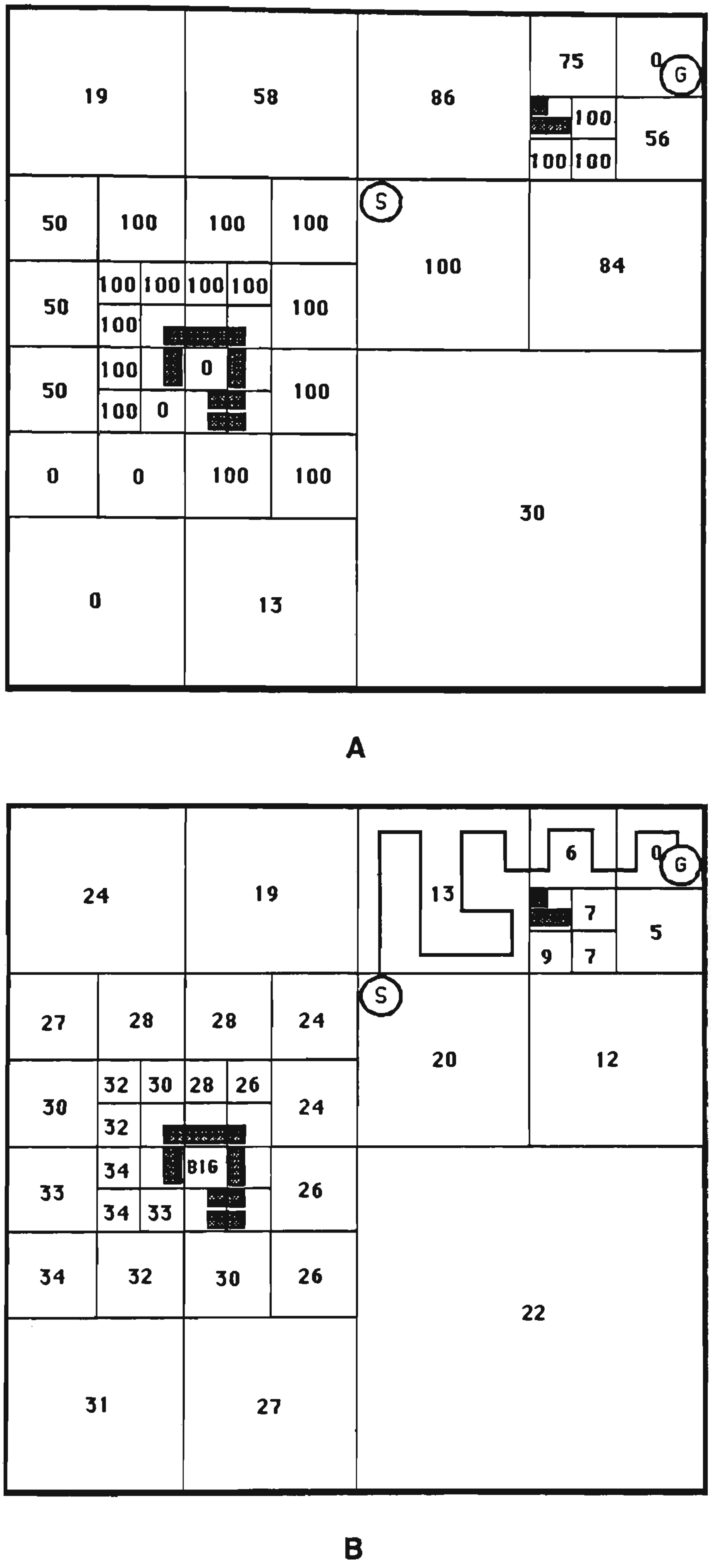


Figure 5.27

Learn all path planning behaviour.

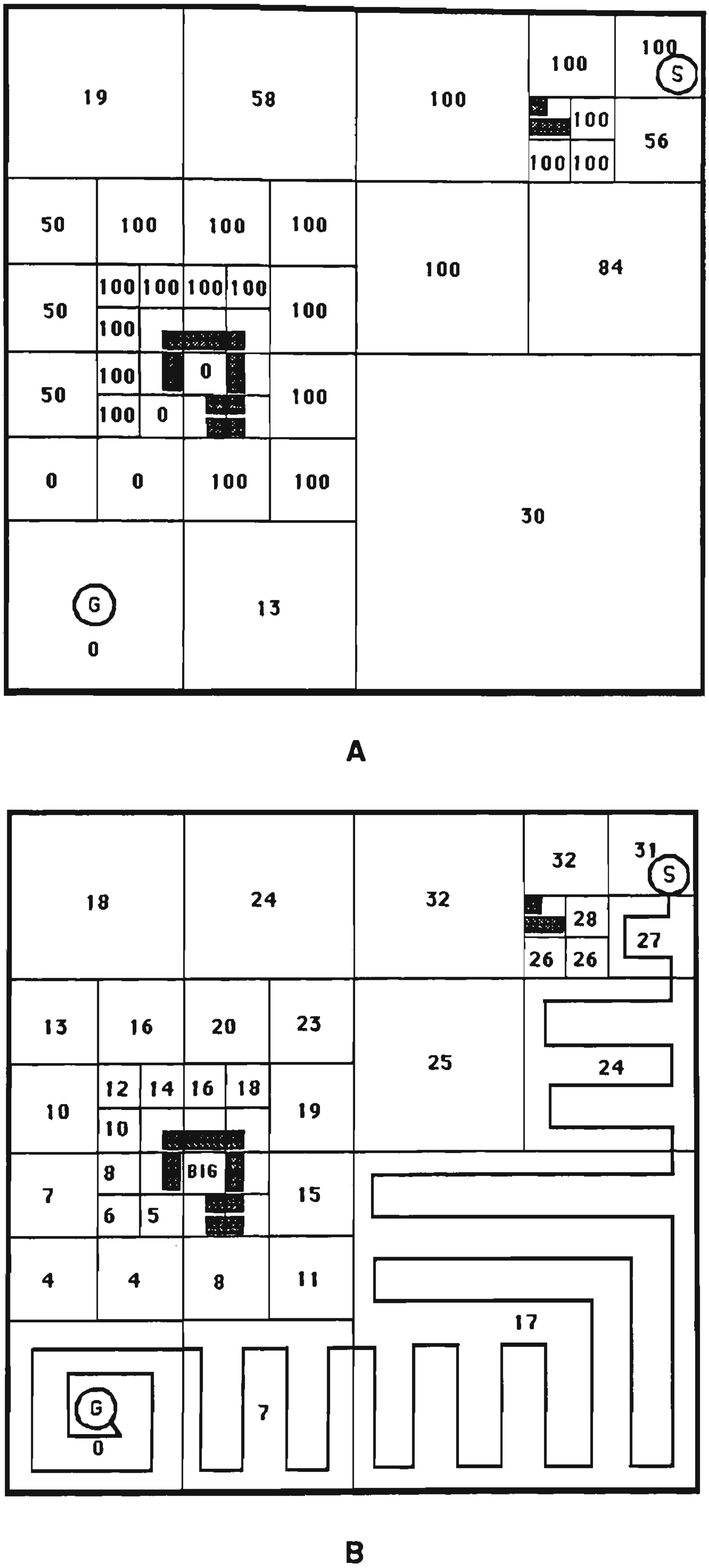


Figure 5.28

This figure continues the experiment started in Figure 5.27.

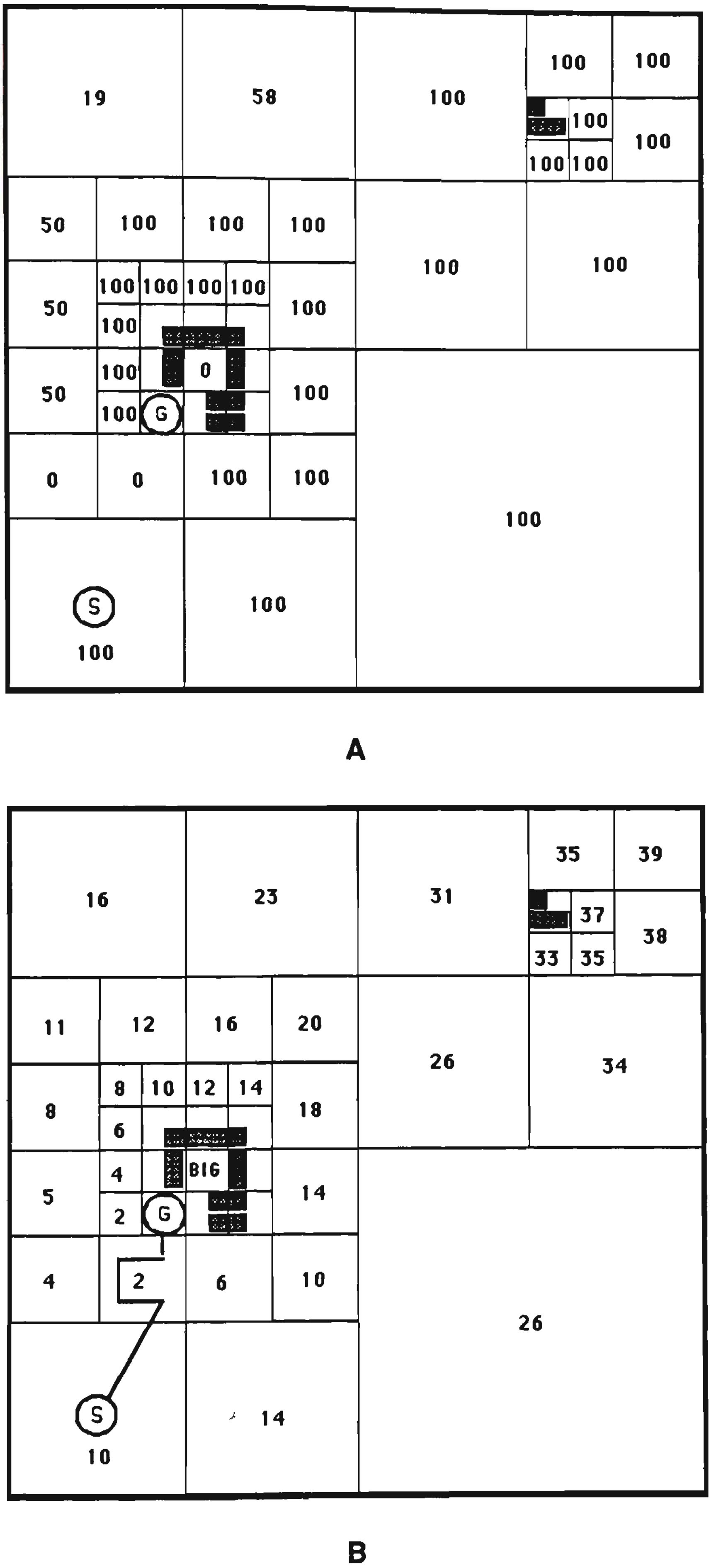


Figure 5.29

This figure continues the experiment started in Figure 5.27.

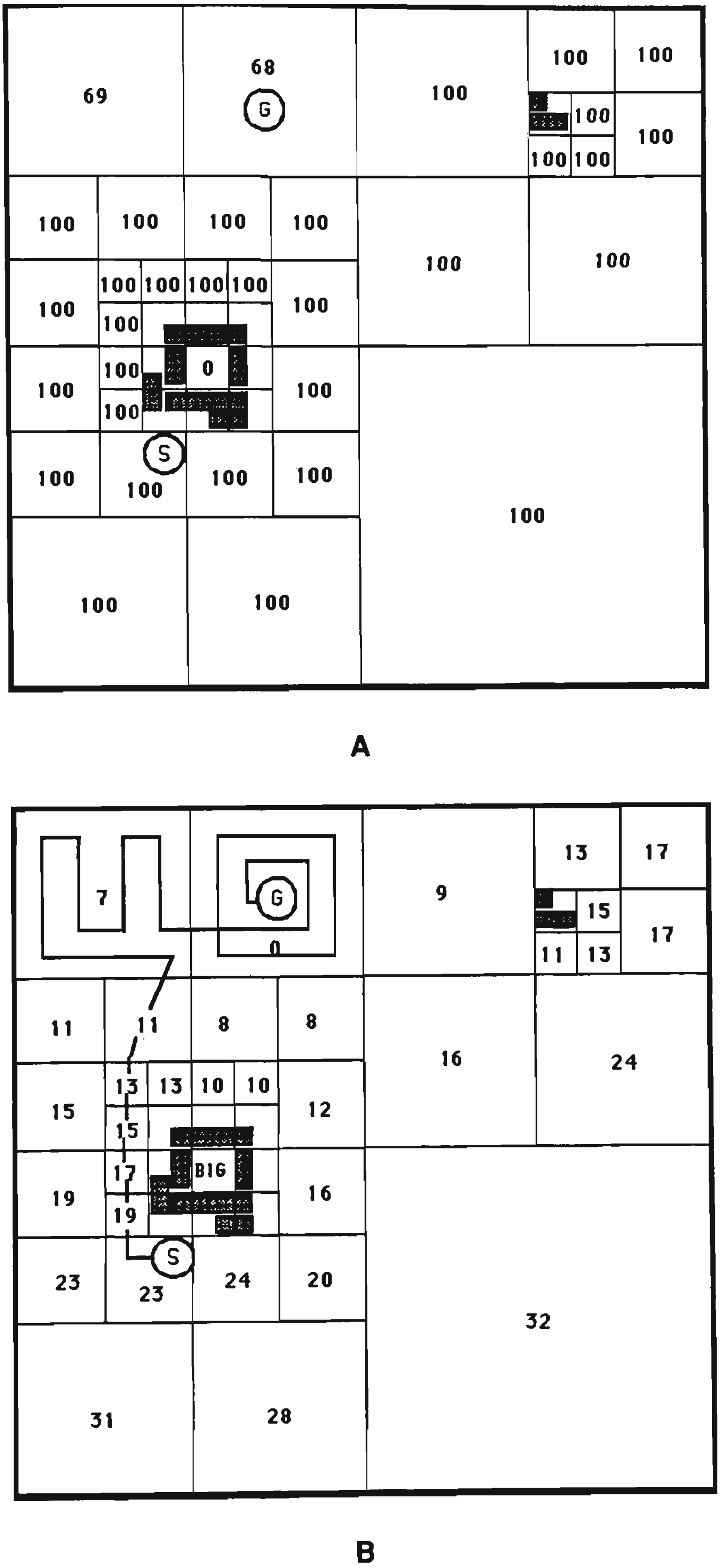


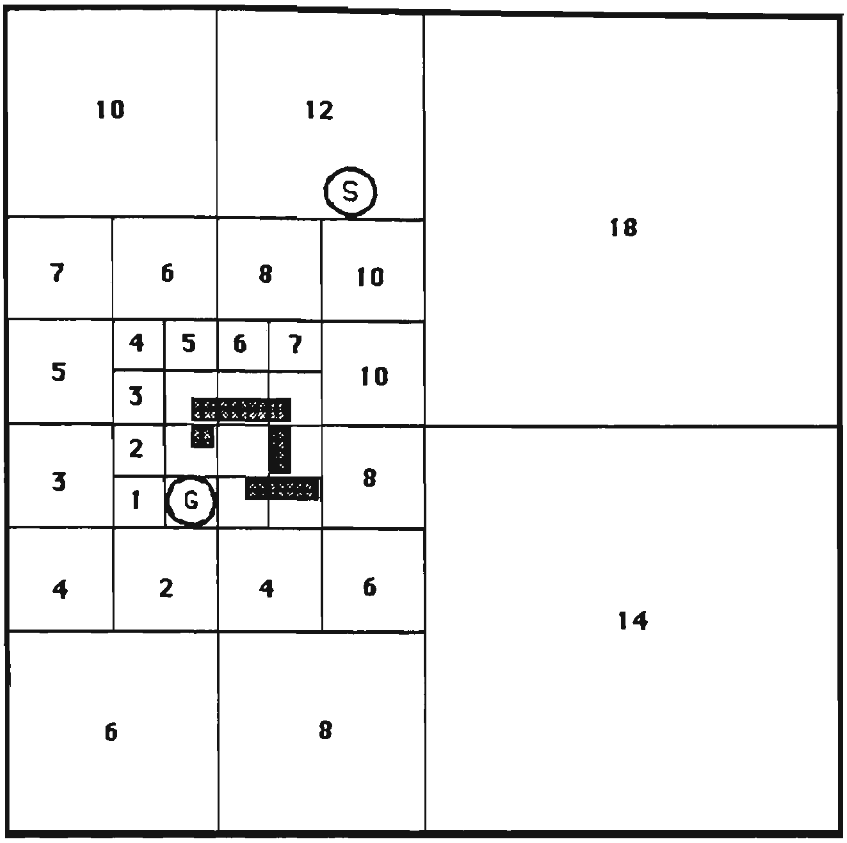
Figure 5.30

This figure continues the experiment started in Figure 5.27.

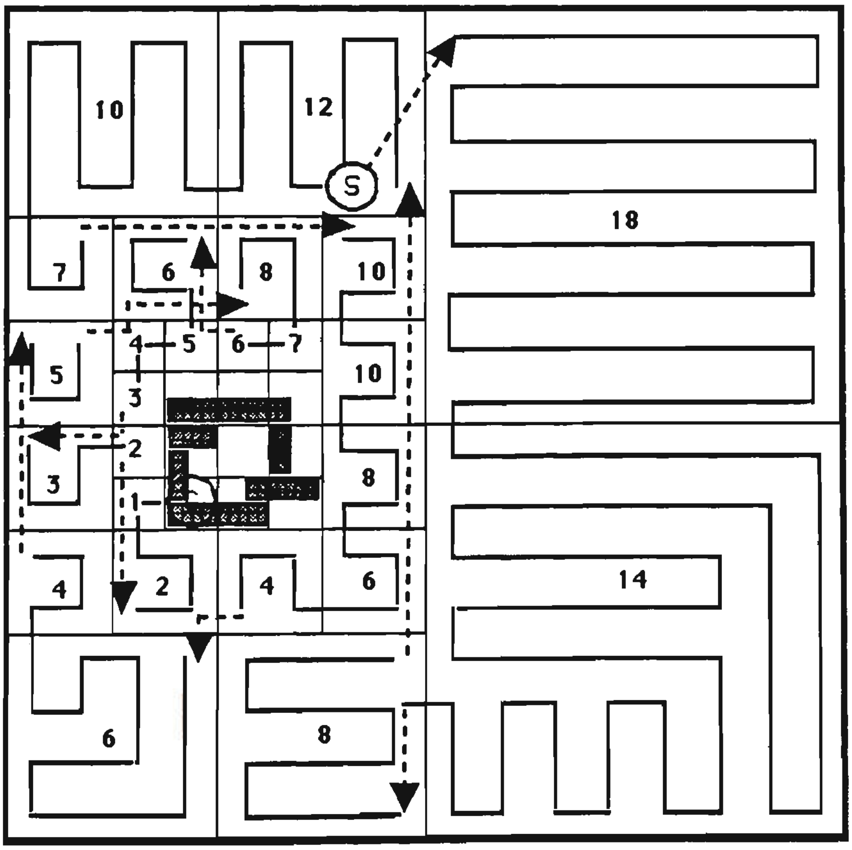
5.6.4 Visit All Path Planning Experiment

Figure 5.31 shows the results of the navigation system operating in "visit all" path planning behaviour mode on the environment map shown in Figure 2.20. Figure 5.31 (A) shows the quadtree model of the environment together with distance transform associated with "visit all" path planning, and the start (S) and goal (G) locations. Figure 5.31 (B) displays the "visit all" solution path between the start and goal positions. To ensure that the robot does not get caught in a corner and have to traverse quadrants which have already been completely swept, the robot moves to the quadrant with the highest distance transform value. A quadrant is only swept if there are no neighbouring quadrants with higher distance transform values. The paths the robot takes when it moves to quadrants with higher distance transform values are shown with broken lines in Figure 5.31 (B). The arrow heads on the broken lines indicate the directions of robot motion.

At the very end of the execution of the "visit all" path the robot encounters an obstacle at the goal position. At this point the EEA instructs the mobile robot to collect more sonar range readings, build a new map and reconstruct the quadtree, as shown in Figure 5.31 (B). The new sonar data only changes the classification of one quadrant in the quadtree from free space to occupied by an obstacle. The structure of the quadtree has not changed and therefore it is not necessary to generate a fresh distance transform, and a new visit all path is not planned.



A



B

Figure 5.31

Visit all path planning behaviour .

5.6.5 Forgetful Path Planning Experiment

Figures 5.32 - 5.34 show the results of the EEA operating in a combination of "optimum" path planning and "forgetful" behaviour modes on the environment map shown in Figure 2.17. The quadtree model of the environment is displayed in Figure 5.32 (B) together with distance transform associated with "optimum" path planning, and the start (S) and goal (G) locations. Once the "optimum" path distance transform has been generated, a fine execution path is planned using the FULL path optimisation function and the zigzag rule. While the robot is executing the "optimum" path, the "forgetful" behaviour is simultaneously decaying the knowledge of the environment by 20% every 30 clock ticks.

Figure 5.32 (B) shows the solution path which has been executed between the start and goal locations, and the associated free space confidence values after 60 clock ticks. Figure 5.32 (A) displays the solution path which has been partially executed between the start and goal positions, and the associated free space confidence values after 120 clock ticks. Figure 5.32 (B) displays the executed path, and the associated free space confidence values after 180 clock ticks. Shortly after the 180 clock tick snap shot, the robot encounters an obstacle. At this point the EEA instructs the mobile robot to collect more sonar range readings and build a new map and reconstruct the quadtree. Figure 5.34 (A) shows the quadtree model with the new confidence values. A new "optimum" path is planned and executed from the current robot position to the goal. This is shown in Figure 5.34 (B).

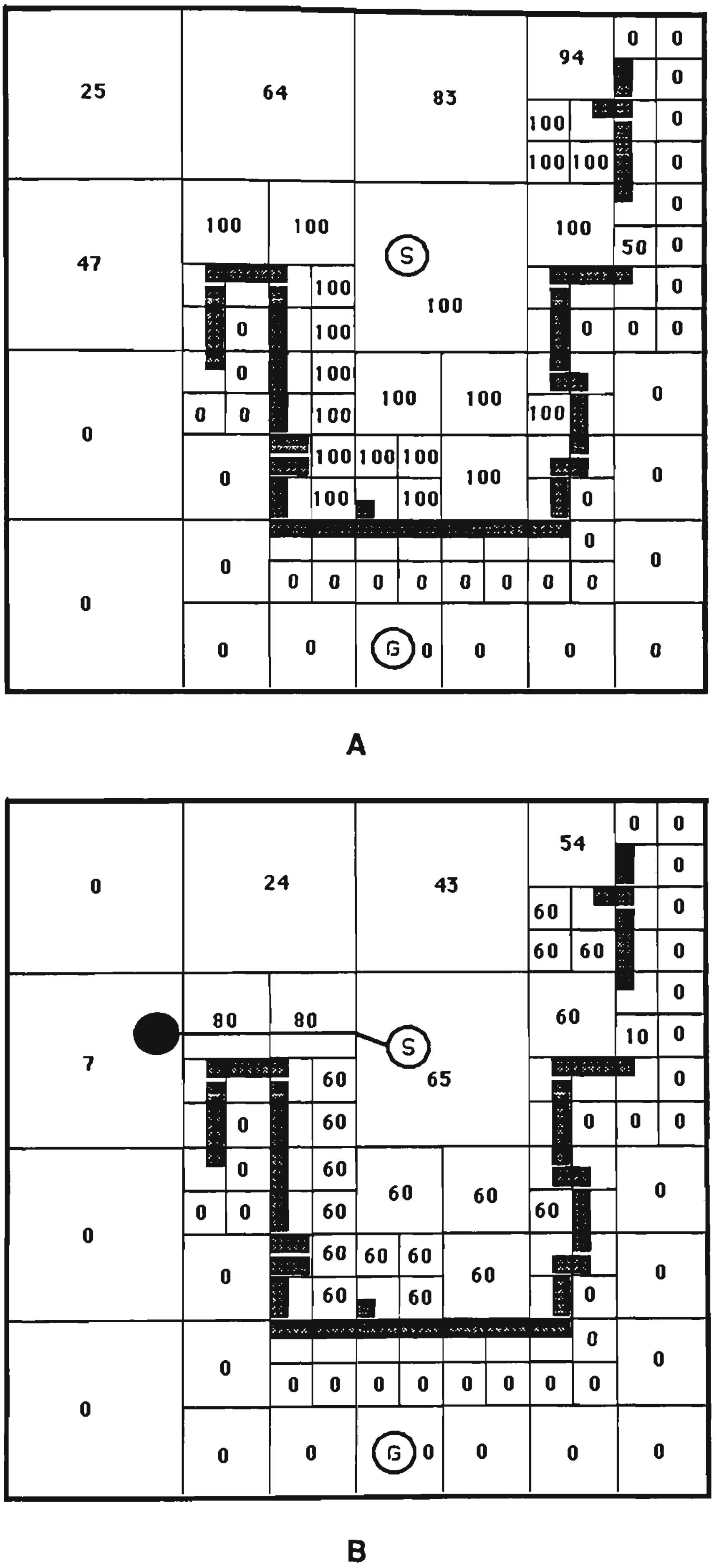


Figure 5.32

Optimum and Forgetful path planning behaviours.

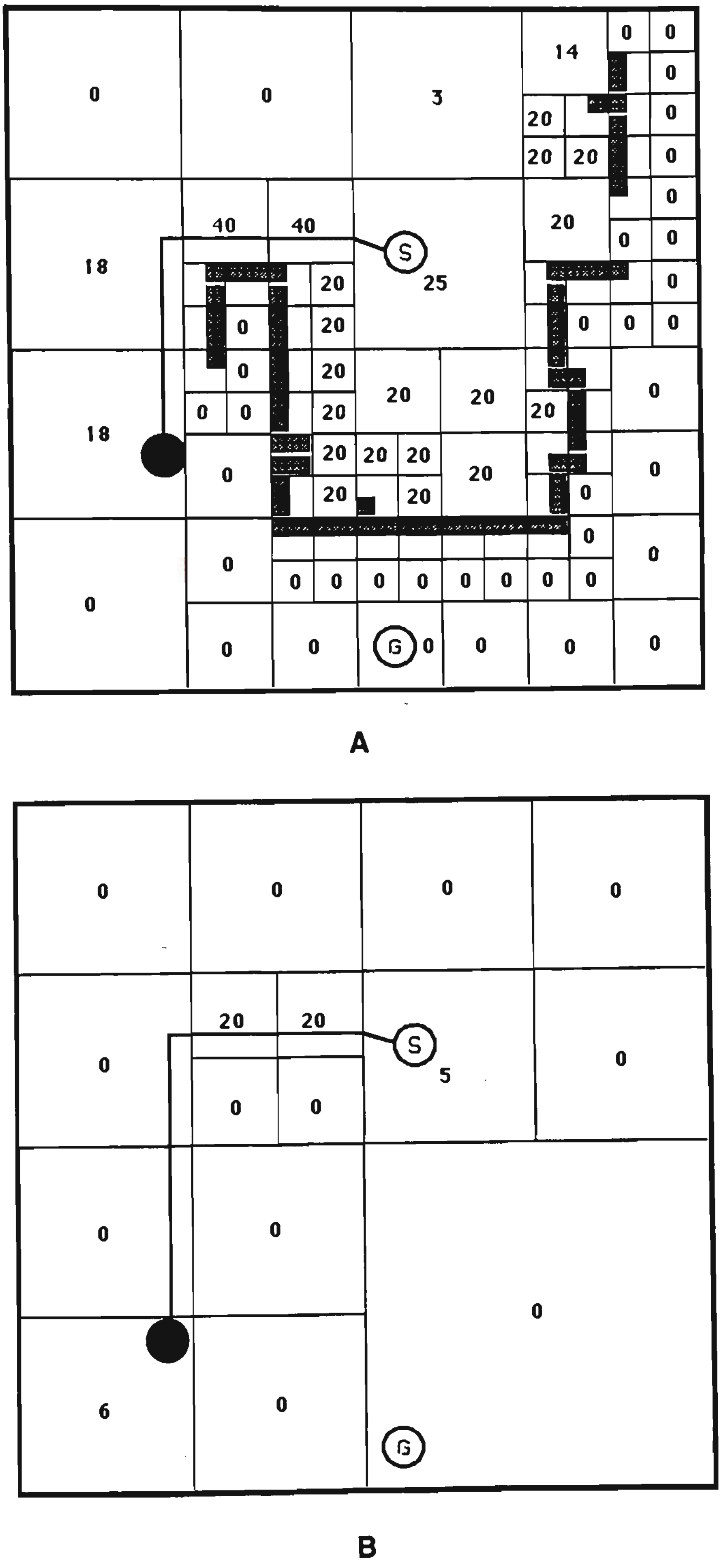
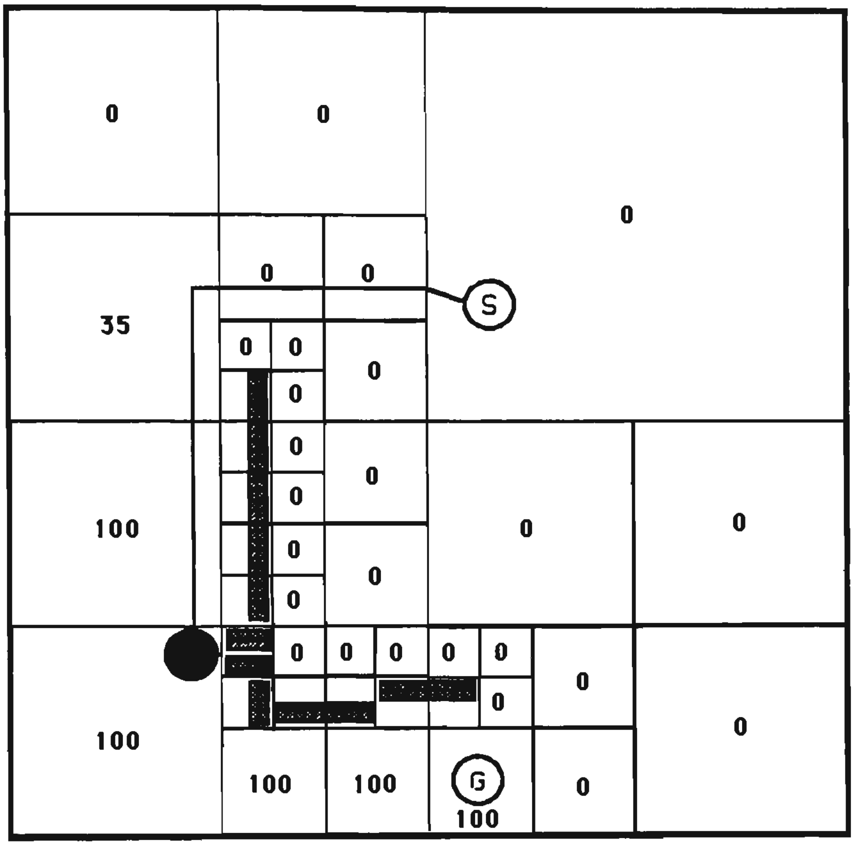
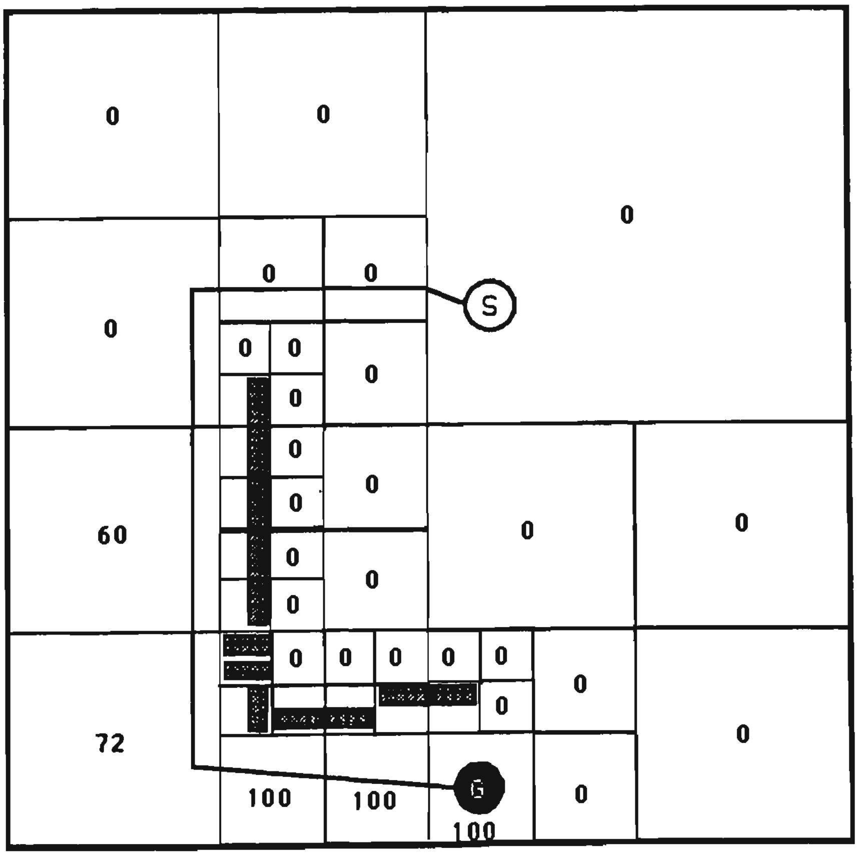


Figure 5.33

This figure continues the experiment started in Figure 5.32.



A



B

Figure 5.34

This figure continues the experiment started in Figure 5.32.

5.6.6 Safest Path Planning Experiment

Figures 5.35 and 5.36 show the results of the EEA operating in the "safest" path behaviour mode on the environment map shown in Figure 2.18. The quadtree model of the environment is displayed in Figure 5.35 (A) together with the start (S) and goal (G) locations. In Figure 5.35 (B) the "safest" path distance transform has been generated using a weighting of $\alpha = 0$, which results in "optimum" path planning behaviour. The fine execution path has been planned using the FULL path optimisation function and the zigzag rule together with a safety clearance of $s_{min} = 2.0$. Since the fine path passes through quadrants of the smallest resolution which are in close proximity to obstacles, the safety clearance feature has no effect on the path. Figure 5.35 (B) shows the "safest" path distance transform has been generated using a weighting of $\alpha = 0.1$. The fine execution path has been planned using the FULL path optimisation function and the zigzag rule, together with a safety clearance of $s_{min} = 2.0$. This fine path is safer than the path generated in Figure 5.35 (A). The safety clearance feature causes the fine path to be generated through the middle of the free space quadrants which lie in the solution path.

Figure 5.36 (A) shows the "safest" path distance transform has been generated using a weighting of $\alpha = 0.3$. The fine execution path has also been planned using the FULL path optimisation function, the zigzag rule, and a safety clearance of $s_{min} = 2.0$. This fine path is safer than the paths shown in Figure 5.35 (A) and (B). Even though some of the quadrants in the solution path are further than 2 units from the nearest obstacle, the safety clearance feature causes the fine path to be generated through the middle of these free space quadrants. This is due to the close proximity of the outside boundary of the quadtree. Figure 5.36 (B) shows the "safest" path distance transform has been generated using a weighting of $\alpha = 0.5$. The fine execution path has also been planned using the FULL path optimisation function, the zigzag rule, and a safety clearance of $s_{min} = 2.0$. This fine path is safer than all other attempts at generating safe paths. Due to the large safety clearance from obstacles, the fine path can be pulled taut through the free space quadrants which form the coarse solution path.

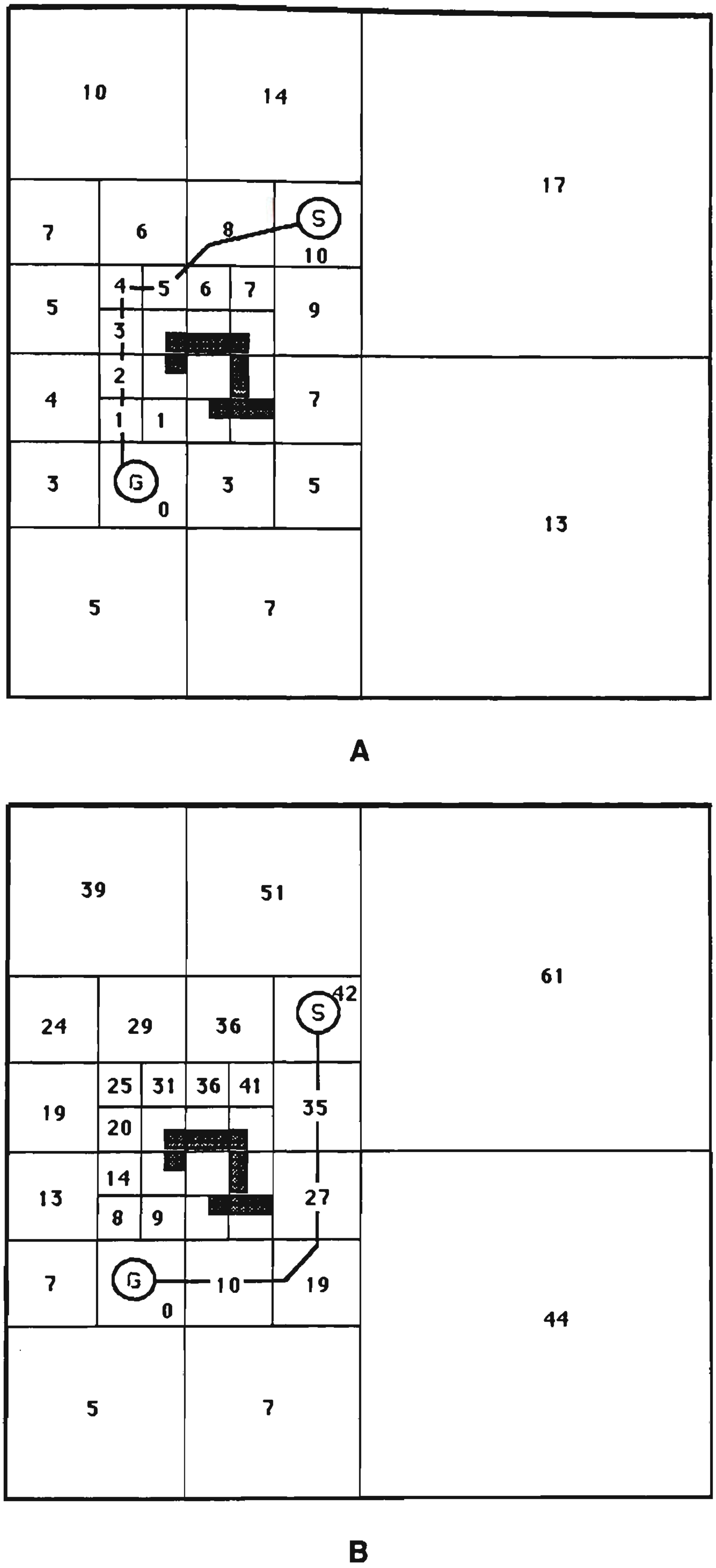


Figure 5.35

Safest path behaviour.

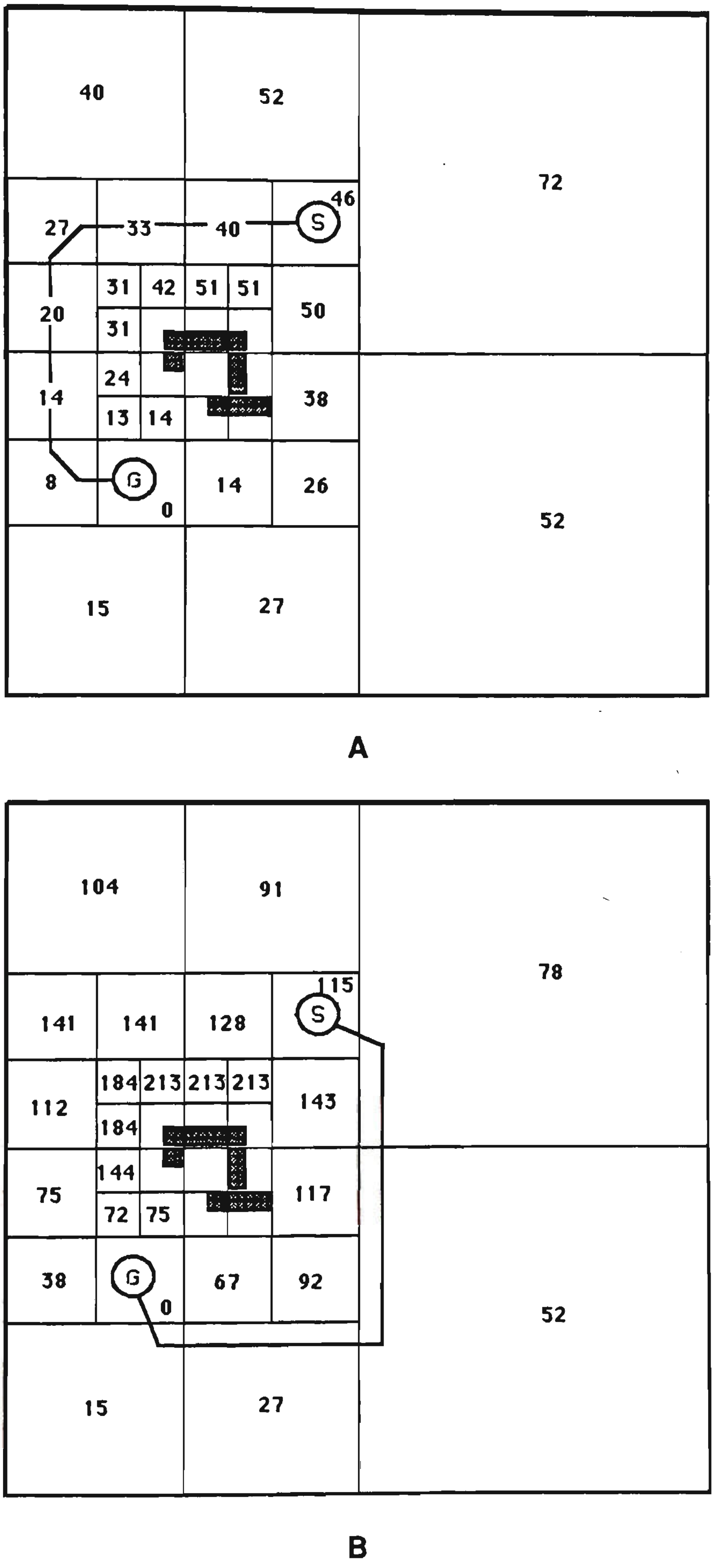


Figure 5.36

This figure continues the experiment started in Figure 5.35.

5.6.7 Best Path Planning Experiment

Figure 5.37 shows the results of the EEA operating in the "best" path behaviour mode on the environment map shown in Figure 2.17. This experiment shows the EEA operating in the same environment shown in Figure 5.24. Figure 5.24 showed the EEA operating in "conservative" path planning mode. The "conservative" planning behaviour kept the solution path in known regions of the environment. However, the path skirted the edges of obstacles. Avoiding the edges of obstacles while favouring known regions of the environment can be achieved by the "best" path planning behaviour. Figure 5.37 shows the robot executing a safe path with greater clearance from obstacles compared to the path shown in Figure 5.24. Figure 5.37 (A) shows the quadtree model of the environment, the confidence values of the free space quadrants which were extracted from the sonar map (Figure 2.17), and the start (S) and goal (G) locations. Figure 5.37 (B) shows the distance transform for the "best" path which was generated using a weighting of $\alpha = 0.4$, together with a fine solution path that uses a safety clearance of $s_{min} = 2.0$. Note: The distance transform values marked with (*) in Figure 5.37 (B) were too large to display in the available space.

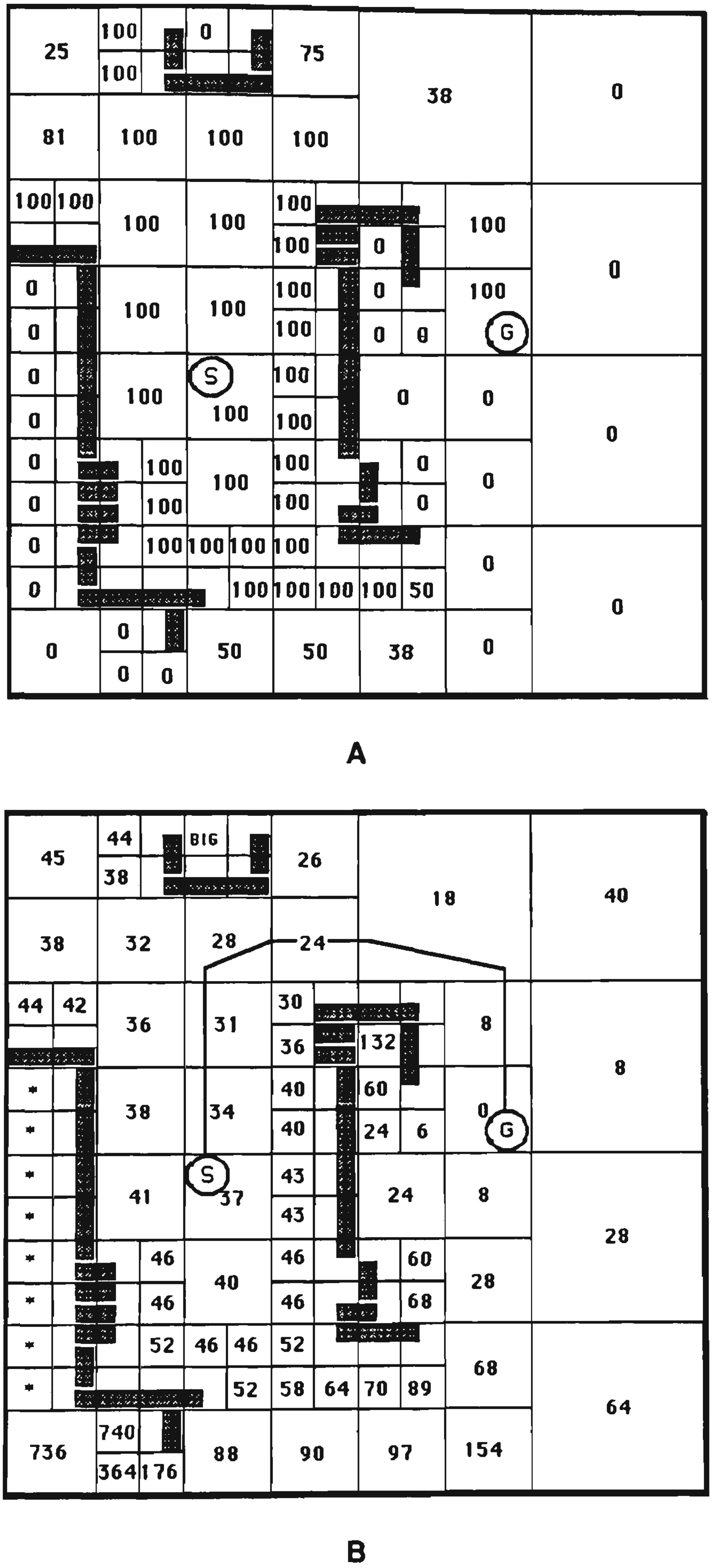


Figure 5.37

Best path planning behaviour.

5.7 Conclusions

This chapter has shown how the Environment Exploration Algorithm (EEA) can be extended to exhibit different types of path planning behaviours, other than the "optimum" path behaviour. It was shown how the "conservative", "adventurous" and "visit all" path planning behaviours which were formulated by [Jarvis *et. al.* 86, 88] for grid based distance transforms could be incorporated to operate within the EEA. This chapter presented four new path planning behaviours, namely: the "learn all", "forgetful", "safe" and "best" path planning behaviours.

Operating the extended EEA in the "learn all" path planning behaviour proved to be an efficient and effective mechanism for systematically mapping all the unknown regions of an environment. It was also shown that the "forgetful" behaviour was a useful behaviour that could coexist with other path planning behaviours in a meaningful way.

This chapter developed a new transform called the "path transform" which is superior to the distance transform since the path transform can simultaneously consider the safety of the robot while finding the shortest path to the goal. The path transform has the desirable properties of potential field path planners without suffering the penalty of local minima. It was shown how the path transform can be applied to the grid and quadtree data structures to produce "safe" path planning behaviour. A new path improvement mechanism for "safe" paths in quadtrees was derived and presented. It was also shown how the path transform could be used with the grid data structure to produce a new "visit all" path. It was shown that the "visit all" path that is generated using path transforms is superior to the "visit all" path that is generated using distance transforms.

A new path planning behaviour called "best" path was derived from fusing the "safe" and "conservative" behaviours. The "best" path planning behaviour was implemented for the grid and quadtree data structures. The grid implementation of "best" path resulted in similar results to those obtained by [Thorpe 84]. Thorpe did not use distance or path transforms to obtain his results.

In this chapter it was shown that the extended EEA can operate with a real mobile robot using live sonar data to navigate autonomously in an unknown environment. Experiments were successfully performed which required the robot to navigate to goals while under the control of a specific path planning behaviour.

Finally, the approach to path planning behaviours described in this chapter is in keeping with the human path planning strategies which were discussed in Section 3.2.

Chapter 6
Comparison of Distance Transform based Path Planners

6.1 Introduction

This chapter presents a comparison of path planners that use distance transforms. This study analyses the performances of the Environment Exploration Algorithm (EEA) which was presented in Chapters 4 and 5, and the [Jarvis *et. al.* 86] grid based algorithm which was reviewed in Chapter 3. It is inappropriate to compare the EEA with other types of path planning algorithms, since distance transforms generate paths from every position in the environment to the nearest goal, while most other path planners plan only a single path from one location to a goal. It therefore follows that distance transforms may be computationally more expensive than other path planners. However distance transforms compensate for their computational burden by supporting multiple robots, multiple goals and different types of path planning behaviours, rather than just the shortest path to the goal. The most suitable comparison to make with the EEA algorithm is the [Jarvis *et. al.* 86] grid based algorithm.

It is intuitively obvious that the EEA algorithm which uses quadtrees will outperform the Jarvis grid algorithm if the environment is free of obstacles. The converse is obvious if the environment is maze like. Environments which are totally free of obstacles, or are heavily cluttered in maze like patterns, occur rarely in practice. It is not obvious which distance transform algorithm is superior in a typical indoor environment i.e. which one has the best average case performance. Figure 6.1 shows three different types of environments in which path planning can take place to navigate a robot from a start (S) to a goal (G). Figure 6.1 (A) shows an environment which is clear of obstacles. Figure 6.1 (B) shows a maze like environment. Figure 6.1 (C) shows a typical indoor environment. The purpose of this research is to determine which distance transform algorithm is best suited to a specific environment given the size of the environment and the degree of clutter caused by obstacles in the environment.

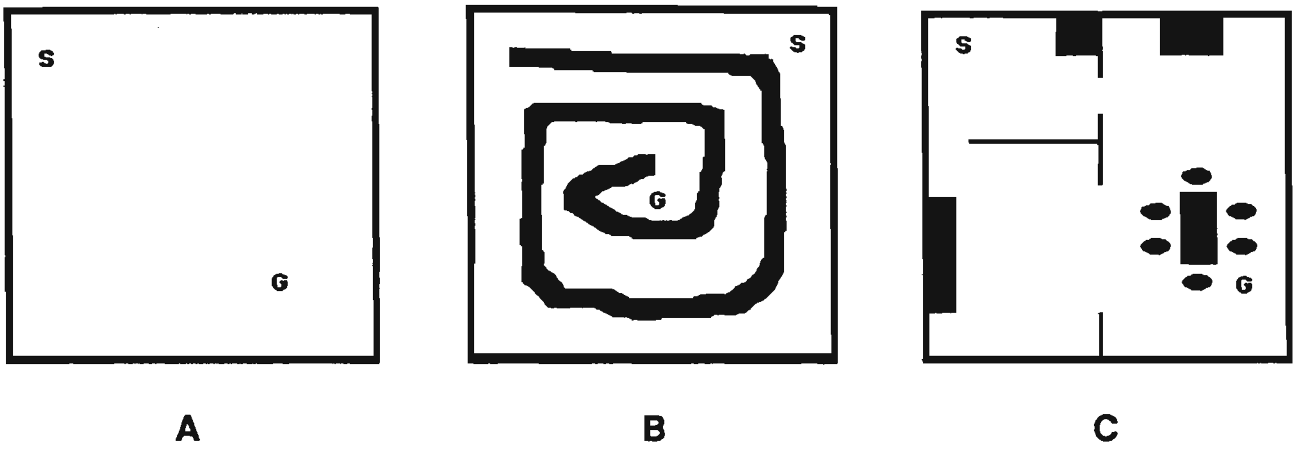


Figure 6.1

Three types of path planning environments.

[Samet 84] presents a theorem which states that the number of quadrants in a quadtree is proportional to the the perimeter of the regions contained in an environment map. Samet also states that increasing the map resolution leads to a linear growth in the number of quadrants in the quadtree. Therefore the cost of computing the distance transform in quadtrees will increase linearly as the resolution of the map grows. In grids, the cost of computing the distance transform does not increase linearly, due to the quadrupling of the number of cells in the grid every time the map resolution is doubled. This research will use Samet's findings to find which distance transform algorithm is best suited to a specific environment given the size of the environment and the degree of clutter caused by obstacles in the environment.

The time to compute the distance transform for quadtrees is heavily dependent on how quickly the neighbours of a free space quadrant can be found. In the grid data structure finding the neighbouring grid cell is a trivial exercise. Finding the neighbours in a quadtree is not a straightforward matter. [Samet 82] proposed an algorithm for neighbour finding in a quadtree. The basic idea is to ascend the quadtree until a common ancestor is located and then descend back down the quadtree in search of the neighbouring quadrant. An alternate approach to searching for neighbours is to a build a list of neighbours for each quadrant leaf. This list is built after the quadtree has been constructed. The overhead for keeping a list of neighbours for each quadrant leaf is the extra memory required to store the list and the processing time required to create the list.

To evaluate the cost of this additional storage this study also compares the generation of distance transforms for quadtrees using neighbour lists and not using neighbour lists. The purpose of the comparison is to see whether the memory overhead for storing neighbour lists results in a significant improvement in the computation time of the distance transform. Another overhead which should be considered is the computational cost of building a quadtree. It is assumed that the environment in which the mobile robot will operate is modelled as a 2 dimensional array. Therefore there is no work to be done to obtain a grid representation of the environment. However this array representation must be converted into a quadtree, which is done using the [Samet 81] quadtree construction algorithm. The quadtree is constructed only once if a robot is operating in a completely known environment. In partially known environments the quadtree continually grows as the robot acquires fresh sensor data. In such situations it is difficult to estimate how much the quadtree will grow each time sensor data is included in the environment model. The time to build a complete quadtree can be regarded as the worst case. However, in practice, the worst case scenario will not arise since a quadtree can be extended in a manner that prevents complete reconstruction using Algorithm 4.10 (Model Update).

In the first stage of path planning for both grid and quadtree data structures, the distance transform is propagated through the regions of free space. The second stage is to find a "fine" path from the start location to the goal location. Finding a "fine" path with grids is a straight forward exercise. However with quadtrees finding a "fine" path requires the construction of a visibility graph. There is obviously less computational effort associated with finding a "fine" path with grids, compared to quadtrees. To compute the "fine" paths for both data structures is an insignificant effort, when compared to the work needed to compute the distance transforms. This fact is highlighted in the experimental results presented in Table 6.1.

Map Size	% Grid Fine Path	% Quadtree Fine Path
8 × 8	–	–
16 × 16	–	–
32 × 32	–	1.980
64 × 64	0.353	1.838
128 × 128	0.186	1.376
256 × 256	0.099	1.212
512 × 512	0.049	0.818

Table 6.1

Fine Path Planning statistics.

This table shows the average percentages that the computation of the "fine" path is of the whole process of path planning, for grids and quadtrees of various map sizes. The statistics in this table were derived from averaging the results of four (4) different environments. The diagrams which describe each environment and the associated path planning statistics are presented in Appendix A. In Table 6.1 the minus (-) signs mean that no statistics could be calculated because the time to compute the "fine" path was too small to measure. Even though the computation of the "fine" path for quadtrees is considerably slower than for grids, this time is not a significant component of the overall path planning computational effort. The real effort in path planning resides in the computation of the distance transform. This study concentrates on the analysis of what configurations and concentrations of obstacles in an environment affect the computation of the distance transform.

This Chapter has been organised in the following manner. Section 6.2 presents the results of using random data to compare the performance of the quadtree based distance transform with the grid based distance transform. Section 6.3 presents the results of comparing the performance of the two distance transform based path planners using spiral and maze data. Section 6.4 presents the results of using various obstacle shapes in five (5) different proportions to compare the performance of the two path planners. Section 6.5 presents the results of comparing the performance of the two path planners using three (3) different indoor environments.

Finally in Section 6.6 the conclusions that were reached and the insights that were gained from comparing the two distance transform based path planners are presented.

All the experimental work reported in this chapter was done using a Macintosh II microcomputer. All timings reported in this chapter are in clock ticks. A clock tick represents one sixtieth (1/60) of one second.

6.2 Random Data

This section reports on the experimental results which were obtained using random data with different percentage concentrations of blocked cells on various environment map sizes. The aim of these experiments was to observe what effect different concentrations of random data had on the speed of computation of the distance transform, and the memory requirements for the grid and quadtree data structures.

Experiments were conducted for eight (8) different concentrations of random data. The concentration of random data is defined to be the ratio of obstacle cells to total number of cells in the environment. For example a 30% concentration of random data means that 30% of the cells in the environment map are obstacles i.e. not free space. The concentrations of random data used in this research were 0%, 5%, 10%, 20%, 30%, 40%, 50% and 60%. The random data experiments were conducted for seven (7) different sized maps. The map sizes which were used were 8x8, 16x16, 32x32, 64x64, 128x128, 256x256 and 512x512. The figures which describe the experimental environments and the associated path planning statistics are presented in Appendix B.

Figures 6.2 and 6.3 summarise the results of this experiment. Figure 6.2 shows eight (8) graphs which correspond to the computation time for each different concentration of random data. Figure 6.3 shows eight (8) graphs which correspond to the memory requirements for each different concentration of random data. Higher resolution versions of the graphs shown in both figures are presented in Appendix B.

Examining the results presented in Figures 6.2 and 6.3 led to the following conclusions. As expected, the distance transform applied to quadtrees in a random concentration of 0% out performed the distance transform applied to grids both in computation time and memory requirements as shown in Figures 6.2 (A) and 6.3 (A). The computation time varied by a single order of magnitude. The time to compute the distance transform in a grid data structure is dependent on the number of cells in the grid, and thus the computation time quadruples as the resolution of the map doubles. The time to compute the distance transform in a quadtree includes the time required to build the quadtree. The time required to construct the quadtree is dependent upon the number of cells in the map. Hence the computation time of the distance transform for quadtrees follows a similar trend to computation time in grids. The memory advantages of the quadtree over the grid are significant for 0% concentration. The memory requirements for the quadtree remain static for all sizes of the environment map, while the memory requirements of the grid quadruple as the size resolution of the map doubles.

Examining the results also shows that the computation time for grids and quadtrees increases until the random concentration reaches 30%, for higher concentrations of random data the computation time decreases. This is due to the fact that as the number of obstacle cells increases the number and length of free space solution paths in the environment decreases. The amount of memory required for building a quadtree diminishes as the random data concentration increases. This is due to the fact that a quadtree is less fragmented because of the increased likelihood that more random obstacle cells will be neighbours to other obstacle cells, and hence can be consolidated into larger obstacles.

For all concentrations of random data other than 0% the grid out performs the quadtree in both computation time and memory requirements. Obstacle cells which are randomly distributed cause severe fragmentation of the quadtree. Even low concentrations of obstacle cells cause undesirable fragmentation of the quadtree. The computation time and memory requirements of the distance transform for quadtrees without neighbour lists is an order of magnitude greater than the corresponding measures for grids. Computing the distance transform for quadtrees with neighbour lists, results in computational savings of approximately 30% at the expense of a two fold increase in memory requirements.

It can be concluded that quadtrees are completely unsuitable in environments where the obstacles are randomly distributed as single blocked cells.

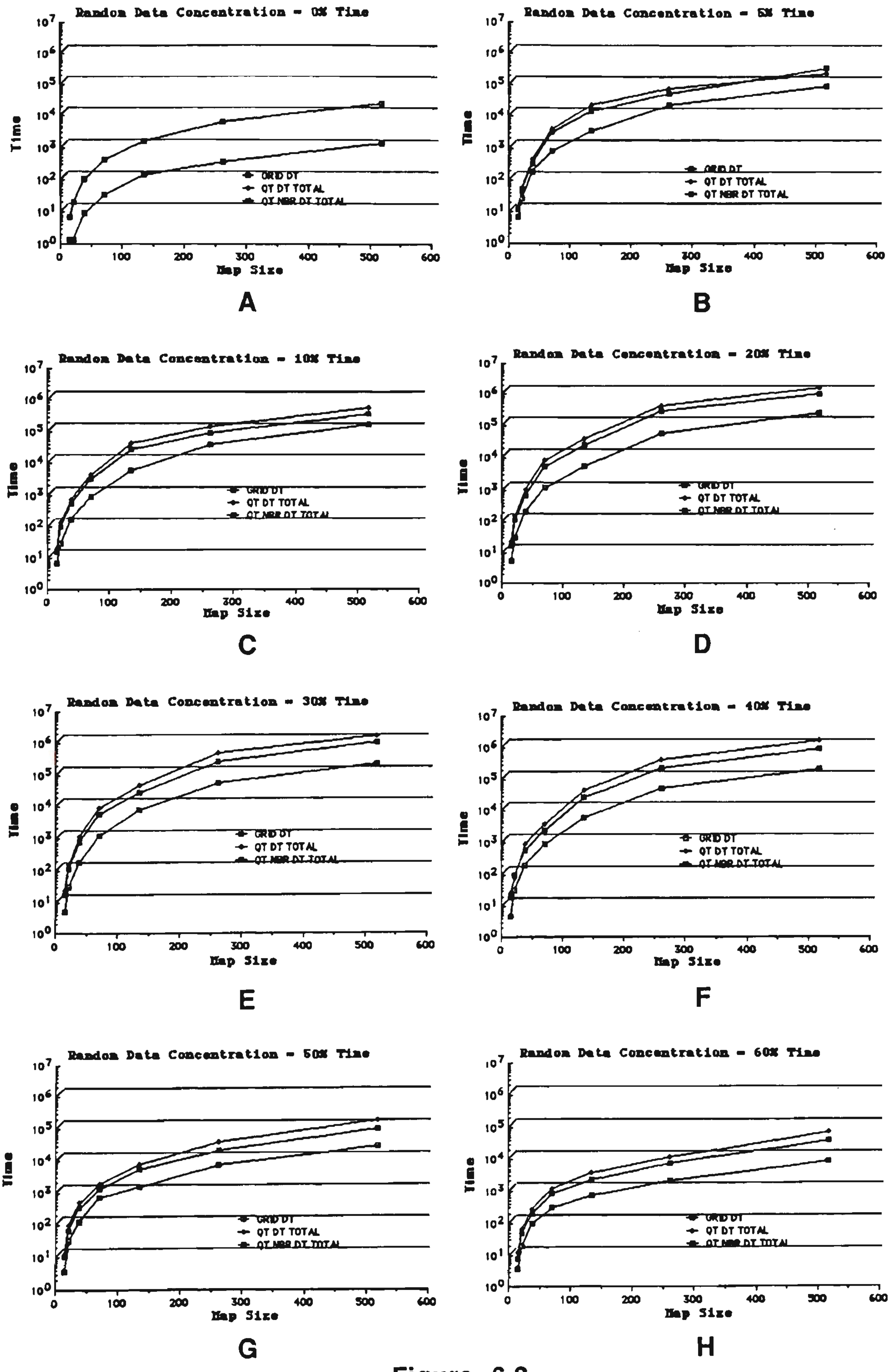


Figure 6.2

Computation Times for Path Planning in Random Environments.

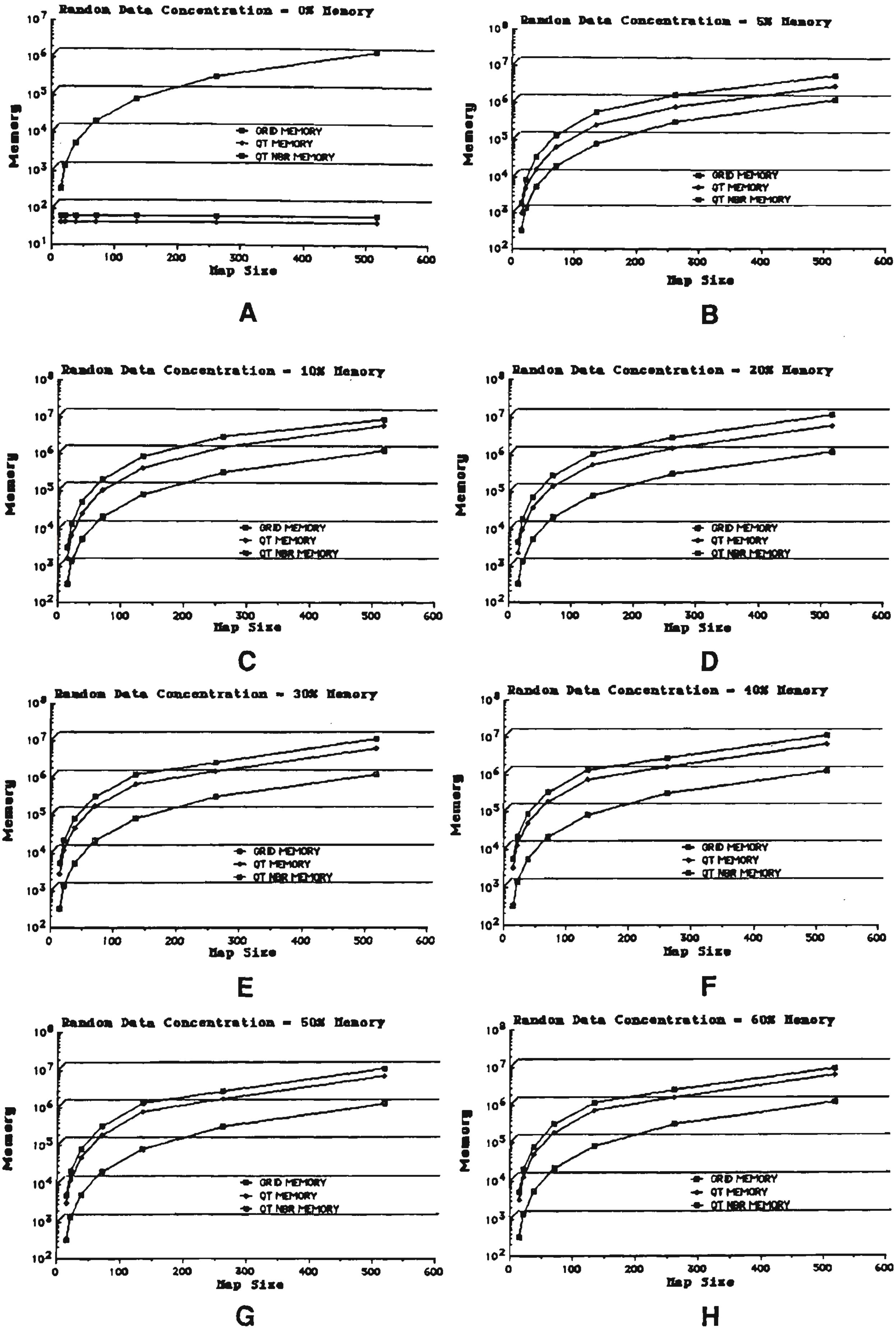


Figure 6.3

Memory Requirements for Path Planning in Random Environments.

6.3 Spiral and Maze Data

This section reports on the experimental results which were obtained using spiral and maze configurations on various environment map sizes. The aim of these experiments was to observe what effect such configurations of the environment had upon the speed of computation of the distance transform, and the memory requirements for the grid and quadtree data structures.

Experiments were conducted for the two environment configurations shown in Figure 6.4. Figure 6.4 (A) shows a spiral map configuration with a goal at the centre of the spiral. Figure 6.4 (B) shows a maze map configuration with a goal at the centre of the maze. The spiral and maze were selected to be circular in shape. Circular shapes are the worst case type of obstacle to represent with a quadtree; this shape ensures that the quadtree that represents this shape is not neatly aligned to the boundaries of the obstacle. Every time the resolution of the map is increased the quadrants along the perimeter of the circular obstacle will divide into smaller quadrants.The experiments were conducted for seven (7) different sized maps. The map sizes which were used were 8x8, 16x16, 32x32, 64x64, 128x128, 256x256 and 512x512. The path planning statistics for this experiment are presented in Appendix C.

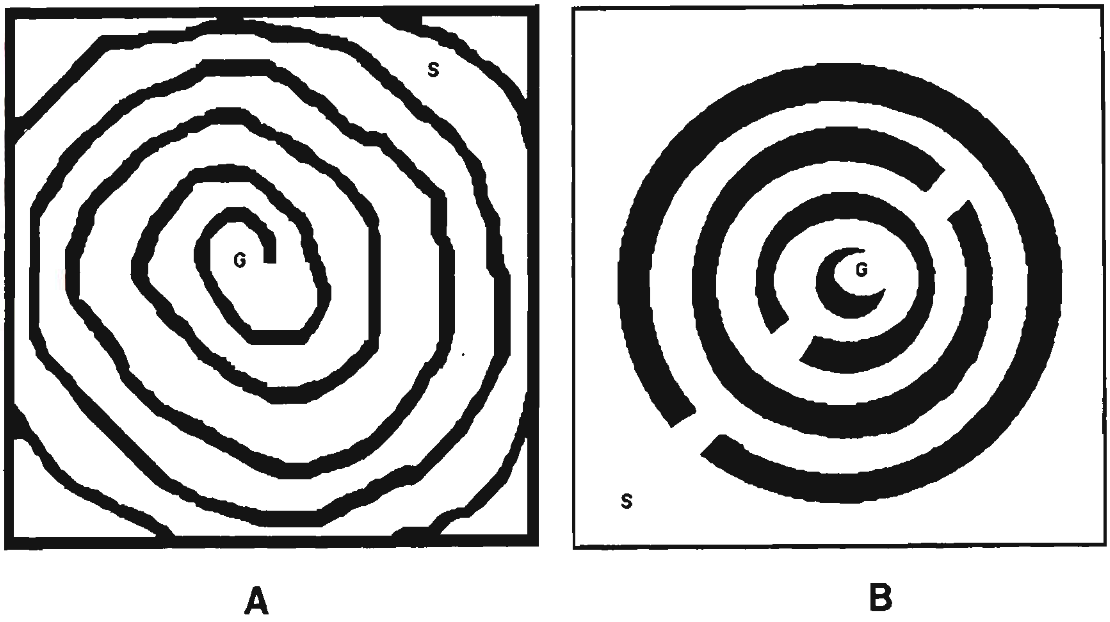


Figure 6.4

Spiral and Maze Path Planning Environments.

Figures 6.5 and 6.6 each show two (2) graphs which summarise the results of this experiment. In both figures graph (A) shows the computation time results and graph (B) shows the memory requirements results. For this experiment the computation time should only be analysed for map sizes greater than 32x32 cells, since no solution path can be found for smaller map sizes. Path planning was performed on the smaller maps to determine the memory requirements.

Examining the results for the spiral experiment presented in Figure 6.5 led to the following conclusions. As expected the distance transform applied to grids in a spiral environment outperformed the distance transform applied to quadtrees both in computation time and memory requirements. The results are in line with Samet's theorem due to the long perimeter of the spiral.

The computation time for quadtrees and grids varied significantly, for a map size of 64x64 cells. However as the resolution of the map doubled, the computation time for quadtrees doubled while with grids the computation time quadrupled. For a 512x512 sized map the computation time for the quadtrees was a factor of three (3) times greater than the computation time for grids. For quadtrees with neighbour lists the computation time was only a factor of two (2) times greater than the computation time for grids. If this trend continued then for larger sized maps quadtrees would outperform grids. However this hypothesis could not be proved due to the memory constraints of the experimental test bed.

A similar trend to the pattern of computation times in the spiral map is evident in the memory requirements of the spiral map. For a 512x512 sized map the memory requirements for quadtrees are of the same order as the memory requirements for grids. If this trend continued for larger sized maps, quadtrees would significantly outperform grids. However this hypothesis could not be verified.

Computing the distance transform for quadtrees with neighbour lists, resulted in similar figures to those obtained using random data. Neighbour lists yield average computational savings of approximately 35% at the expense of a two fold increase in memory requirements.

It can be concluded that quadtrees are not as good as grids for solving path planning problems in spiral map environments for all sized maps up to and including 512x512 sized maps. Quadtrees could be better than grids for solving path planning problems in spiral map environments for maps which are greater than 512x512 cells in size.

Examining the results for the maze experiment presented in Figure 6.6 led to the following conclusions. The distance transform applied to grids in a maze environment out performed the distance transform applied to quadtrees both in computation time and memory requirements for all sized environment maps up to and including 256x256 sized maps. Quadtrees out performed grids for 512x512 sized maps in all respects with the exception of the memory requirements for quadtrees with neighbour lists. The results is also in line with Samet's theorem, due to the reduced length of the perimeter of the maze compared to the perimeter of the spiral.

Computing the distance transform for quadtrees with neighbour lists, resulted in similar results to those obtained for the spiral and random maps i.e. neighbour lists have average computational savings of approximately 32% at the expense of a two fold increase in memory requirements.

It can be concluded that quadtrees are better than grids for solving path planning problems in maze map environments for environment maps which are greater than 256x256 cells in size.

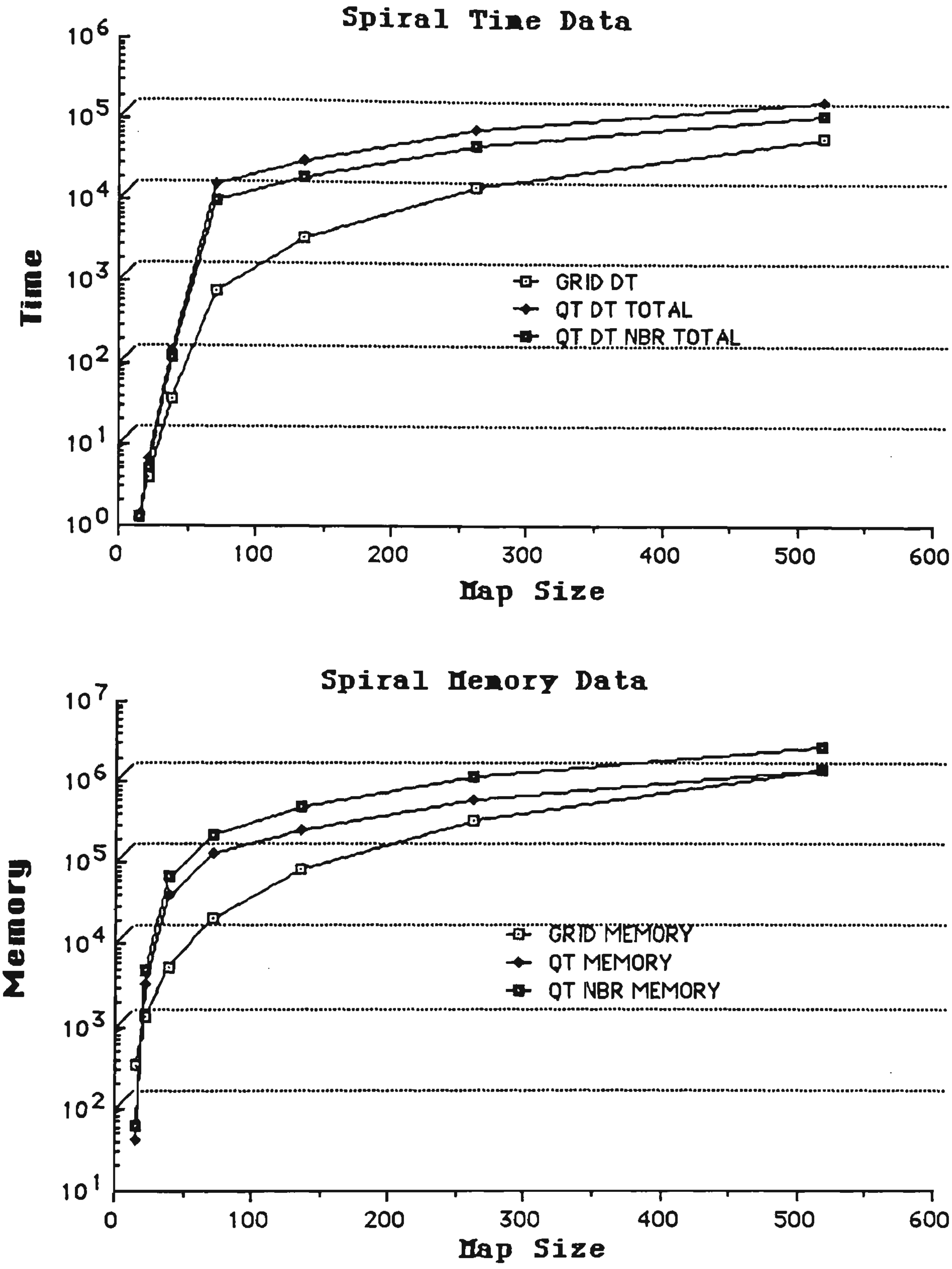
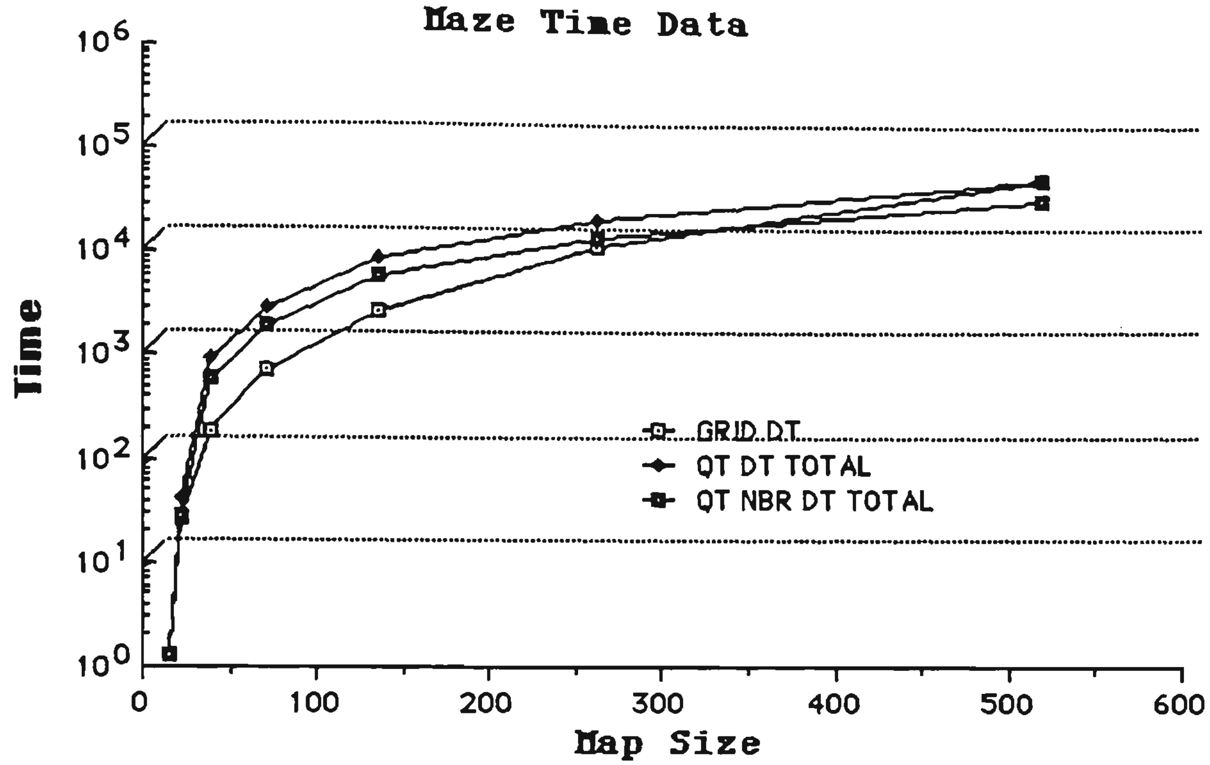


Figure 6.5

Computation Time and Memory Requirements for Path Planning in a Spiral environment.



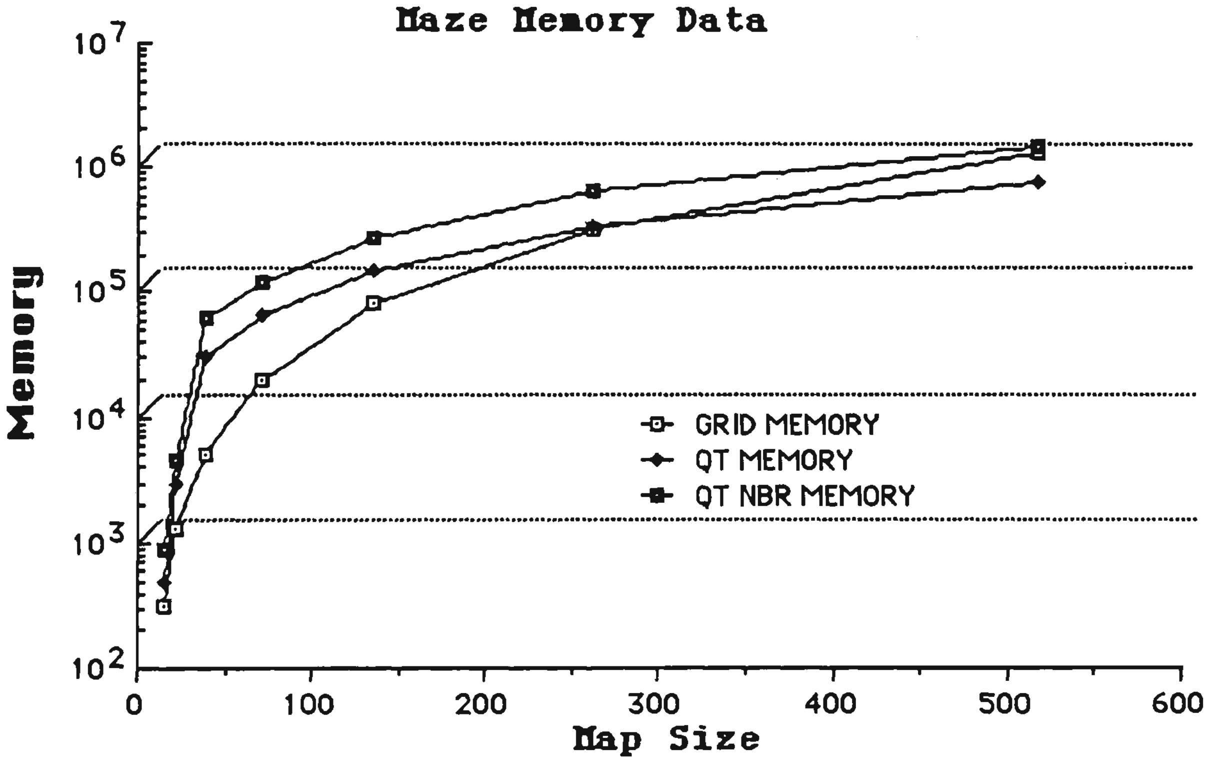


Figure 6.6

Computation Time and Memory Requirements for Path Planning in a Maze environment.

6.4 Obstacle Data

In Sections 6.2 and 6.3 experiments were performed on the worst case situations that can arise using distance transform path planners. Environment set ups of this kind are not likely to occur in practice. It is far more likely that a smaller set of reasonably sized obstacles which are uniformly distributed will occur. This section reports on the experimental results which were obtained using various configurations of obstacles with different environment map sizes. The aim of these experiments was to observe what effect such configurations of the environment had upon the speed of computation of the distance transform, and the memory requirements for the grid and quadtree data structures.

Experiments were conducted for five (5) obstacle configurations. The first obstacle configuration which was tested is shown in Figure 6.7. This figure shows a circular obstacle at the centre of the map. Placing a circular obstacle at the centre of a map causes the greatest fragmentation of the quadtree. The other four (4) obstacle configurations had two, three, four and five obstacles respectively. The figures which describe these four (4) obstacle configurations are presented in Appendix D. These obstacle configurations also included objects with straight line edges. However these obstacles were orientated to ensure that the edges of the obstacles did not align with any quadrant boundaries in the quadtree. The experiments were conducted for seven (7) different sized maps. The map sizes which were used were 8x8, 16x16, 32x32, 64x64, 128x128, 256x256 and 512x512. The path planning statistics for this experiment are presented in Appendix D.

Figures 6.8 - 6.10 summarise the results of this experiment. Figure 6.8 contains three graphs which show the computation time results, the memory requirements, and the relationship between perimeters of obstacles and computation time for the path planning experiment depicted in Figure 6.7. Figures 6.9 and 6.10 each show six (6) graphs which correspond to the path planning statistics for the other four obstacle configurations (there are 3 graphs for each obstacle configuration). Higher resolution versions of the graphs shown in Figures 6.9 and 6.10 are presented in Appendix D.

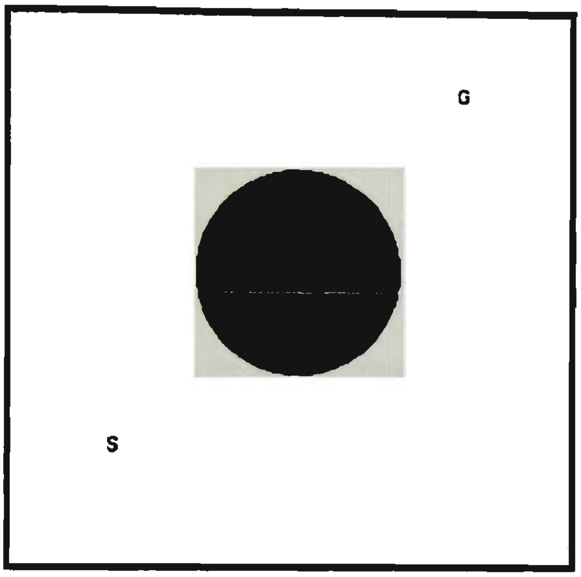


Figure 6.7

Path Planning in a One Obstacle environment.

Examining the results presented in Figure 6.8 led to the following conclusions. The distance transform applied to quadtrees out performed the distance transform applied to grids both in computation time and memory requirements for environment maps which are greater than or equal to 32x32 cells in size. The results confirm Samet's theorem and show that there is a linear relationship between the perimeter length of the obstacle and time of computation of the distance transform in a quadtree, as shown by the Perimeter Data graph in Figure 6.9. The distance transform in quadtrees is dependent only on the perimeter of obstacles in the environment, while the grid distance transform in contrast is dependent on the number of cells in the grid.

Computing the distance transform for quadtrees with neighbour lists, resulted in similar results to those obtained in Sections 6.2 and 6.3. Using quadtrees with neighbour lists has average computational savings of approximately 18% at the expense of a two fold increase in memory requirements.

Examining the results presented in Figures 6.9 and 6.10 led to the following conclusions. As the degree of clutter from obstacles in the environment increased the performance of the grid based distance transform improved. For the two obstacle environment the distance transform applied to quadtrees outperformed the distance transform applied to grids both in computation time and memory requirements for environment maps which are greater than or equal to 64x64 cells in size. Similarly for three obstacle environments the distance transform applied to quadtrees out performed the distance transform applied to grids for environment maps which are greater than or equal to 128x128 cells in size. The degradation of performance of quadtree distance transform is shown in the Perimeter Data graphs by the increased slope of the line which represents the perimeter length versus the computation time. The slope of this line depends on both the total perimeter length of all the obstacles in the environment, and the complexity of the environment i.e. the number of obstacles in the environment. To perform accurate path planning the shapes of the obstacles shown in the experimental environments must be modelled with reasonable accuracy. This requires a map resolution of at least 128x128 cells. The quadtree distance transform performs well at this level at resolution for all the experiment maps with the exception of the highly cluttered 5 obstacle environment.

It can be concluded that quadtrees are better than grids for solving path planning problems in uncluttered obstacle environments. The map resolution of the environment is dependent on the number of obstacles. If only one obstacle needs to be represented the resolution can be as low as 32x32 cells. Higher map resolutions are needed as the number of obstacles increases. Generally for environments with four or less obstacles a resolution of at least 128x128 is needed. Quadtrees are better than grids for cluttered environments if it is a requirement that the obstacles in the environment be accurately modelled. At high map resolutions in cluttered environments quadtrees offer substantial memory savings in addition to the computational savings.

Computing the distance transform for quadtrees with neighbour lists for the four (4) obstacle environments resulted in average computational savings of approximately 25% for the two obstacle environment, 16% for the three and four obstacle environments, and

18% for the five obstacle environment. For all four obstacle environments the computational savings were offset by a two fold increase in memory requirements.

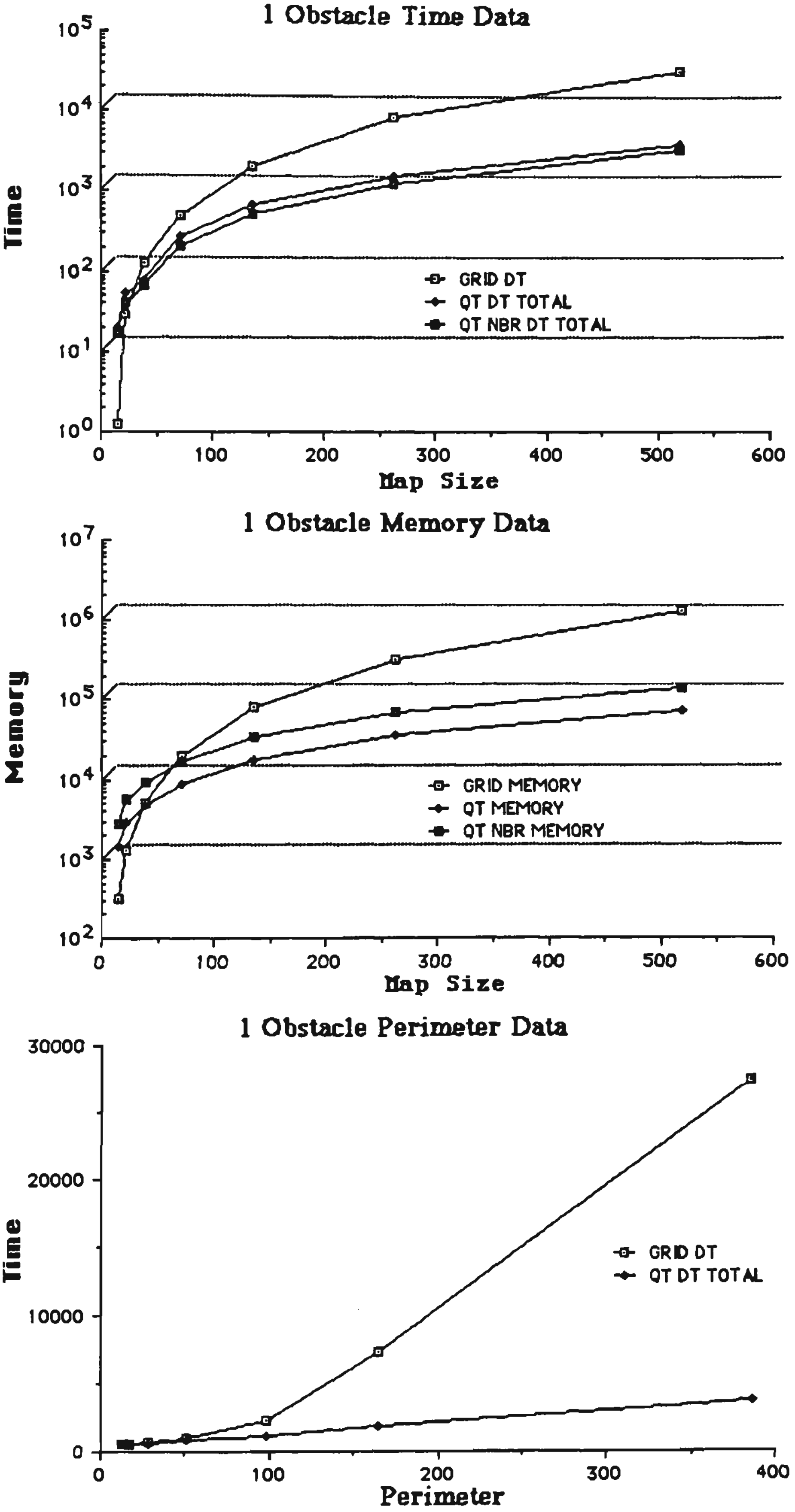


Figure 6.8

Path Planning Statistics for a One Obstacle environment.

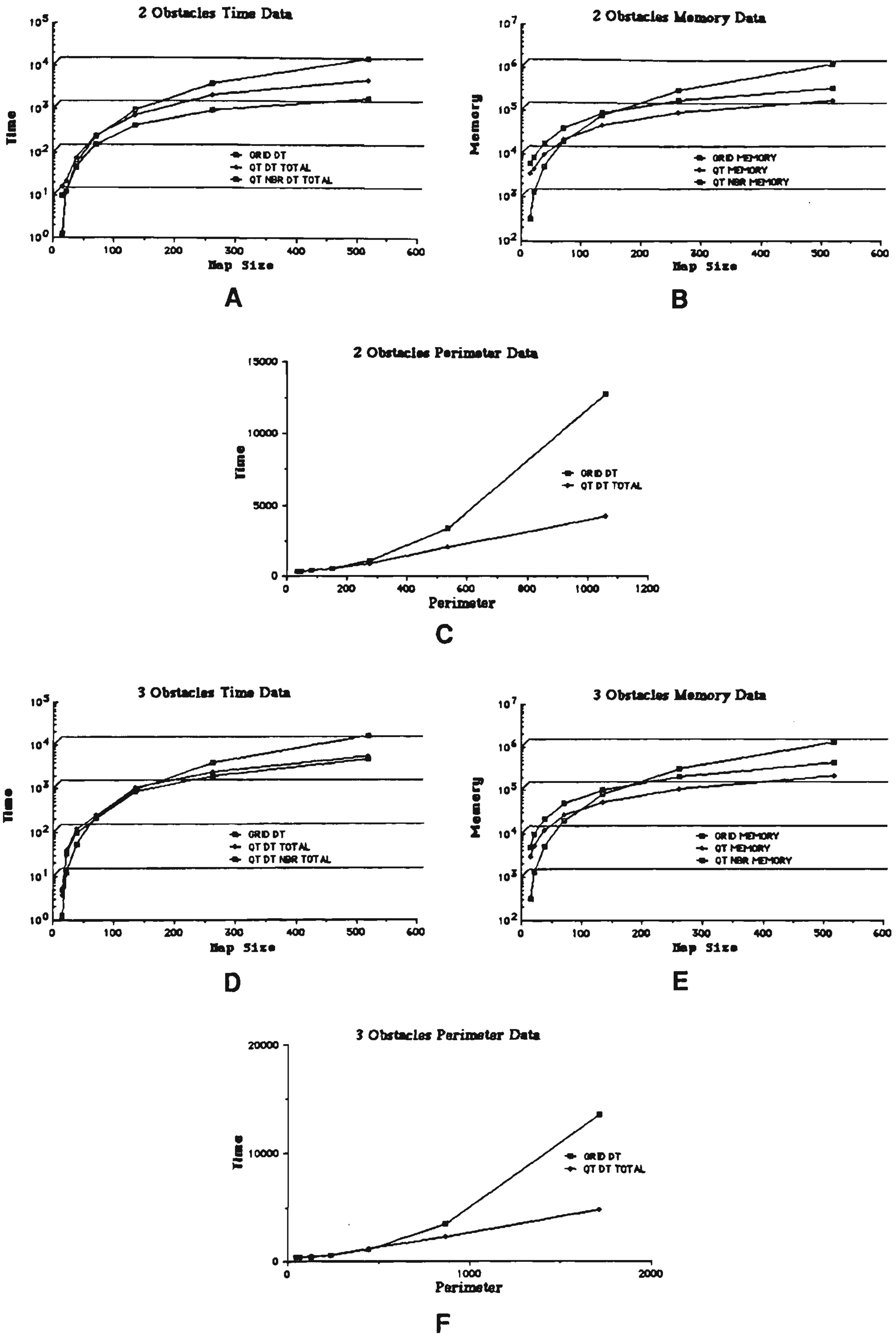


Figure 6.9

Path Planning Statistics for a Two and Three Obstacle environments.

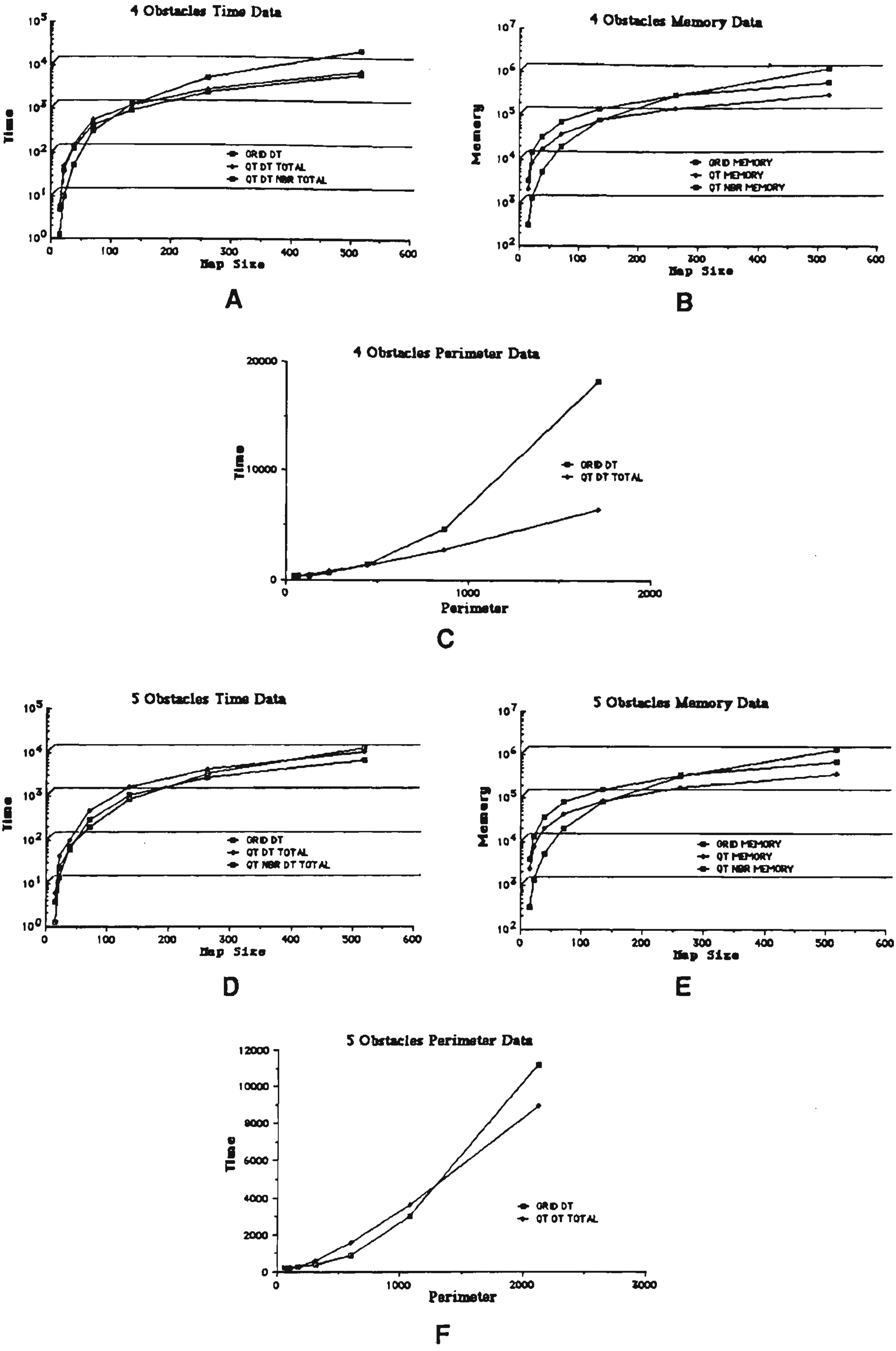


Figure 6.10

Path Planning Statistics for Four and Five Obstacle environments.

6.5 Indoor Environment Data

In Section 6.4 path planning experiments were performed in environments which were cluttered with obstacles. All the obstacles in that experiment were made up of shapes that caused the greatest possible fragmentation of the quadtree. In practice. environments of this kind are extremely rare. It is far more likely that an environment will contain a variety of obstacle shapes. These shapes will range from shapes that cause minimal fragmentation of the quadtree e.g. walls in an indoor environment to complex polygonal shapes which cause high fragmentation of the quadtree. This section reports on the experimental results which were obtained for indoor environments with different environment map sizes. The aim of these experiments was to observe what effect such configurations of the environment had upon the speed of computation of the distance transform, and the memory requirements for the grid and quadtree data structures.

Experiments were conducted for three (3) indoor environments. The environments which were tested are shown in Figure 6.11. Figure 6.11 (A) shows a map of a computer laboratory. The laboratory contains a mixture of shapes ranging from straight line walls to circular tables. Figure 6.11 (B) shows the maze like map of a horse stable. Figure 6.11 (C) shows the the map of two rooms in a house.The experiments were conducted for seven (7) different sized maps. The map sizes which were used were 8x8, 16x16, 32x32, 64x64, 128x128, 256x256 and 512x512. The path planning statistics for this experiment are presented in Appendix E.

Figures 6.12 - 6.14 summarise the results of this experiment. Each of these figures contains two graphs which show the computation time and the memory requirement results for the path planning environments shown in Figure 6.11.

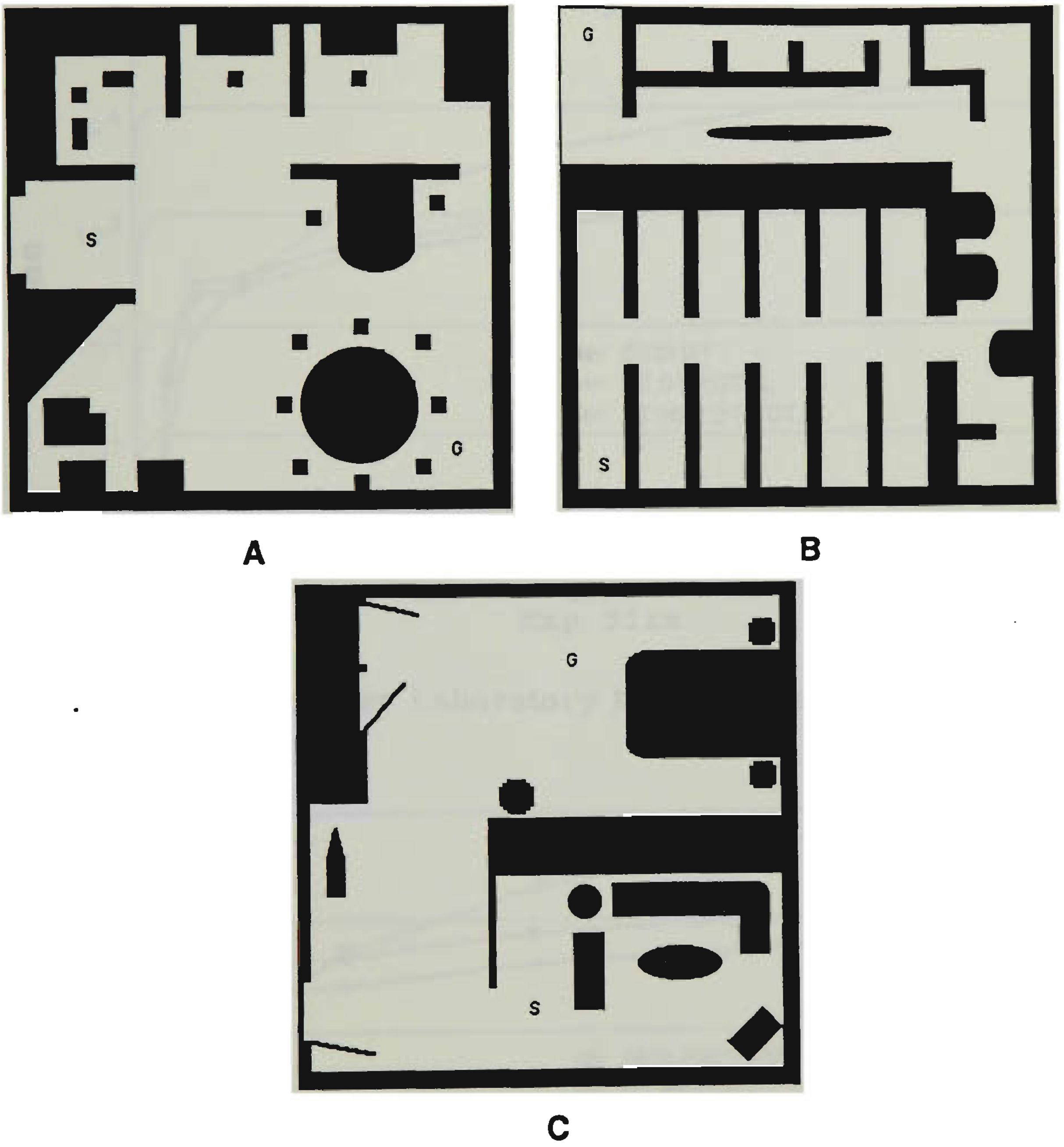
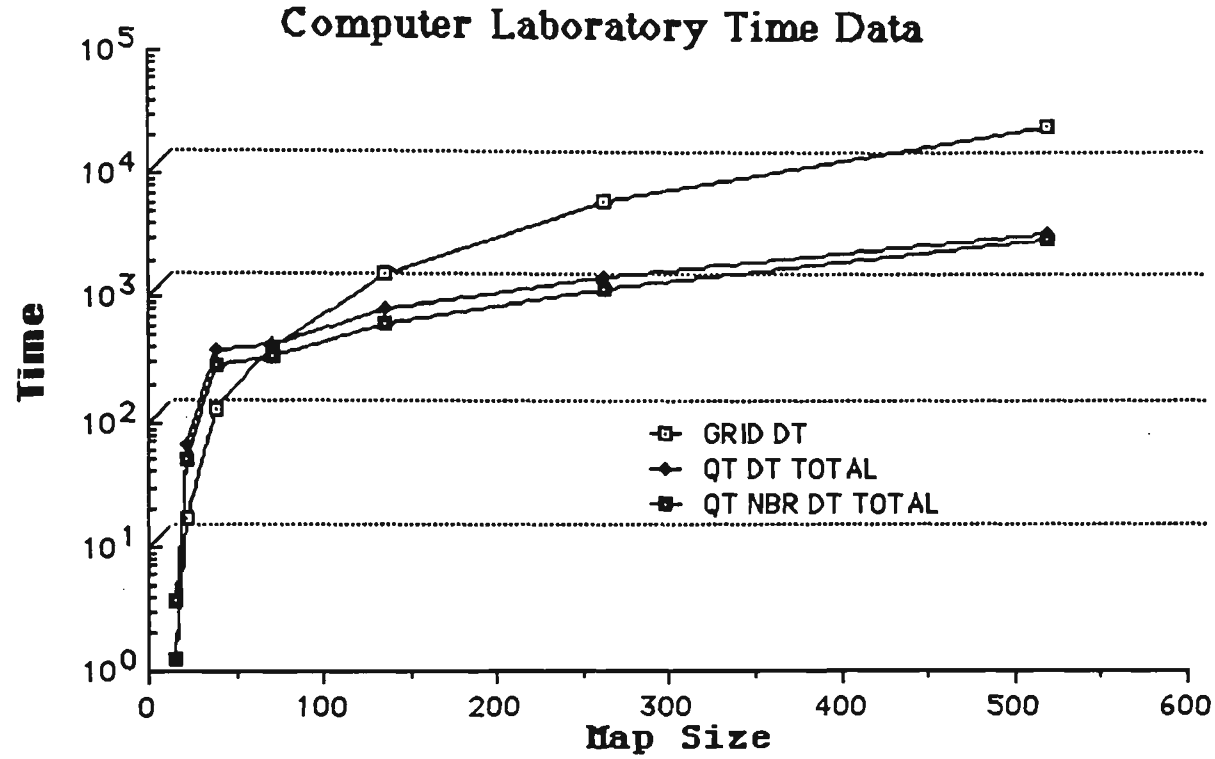


Figure 6.11

Path Planning in three indoor environments.



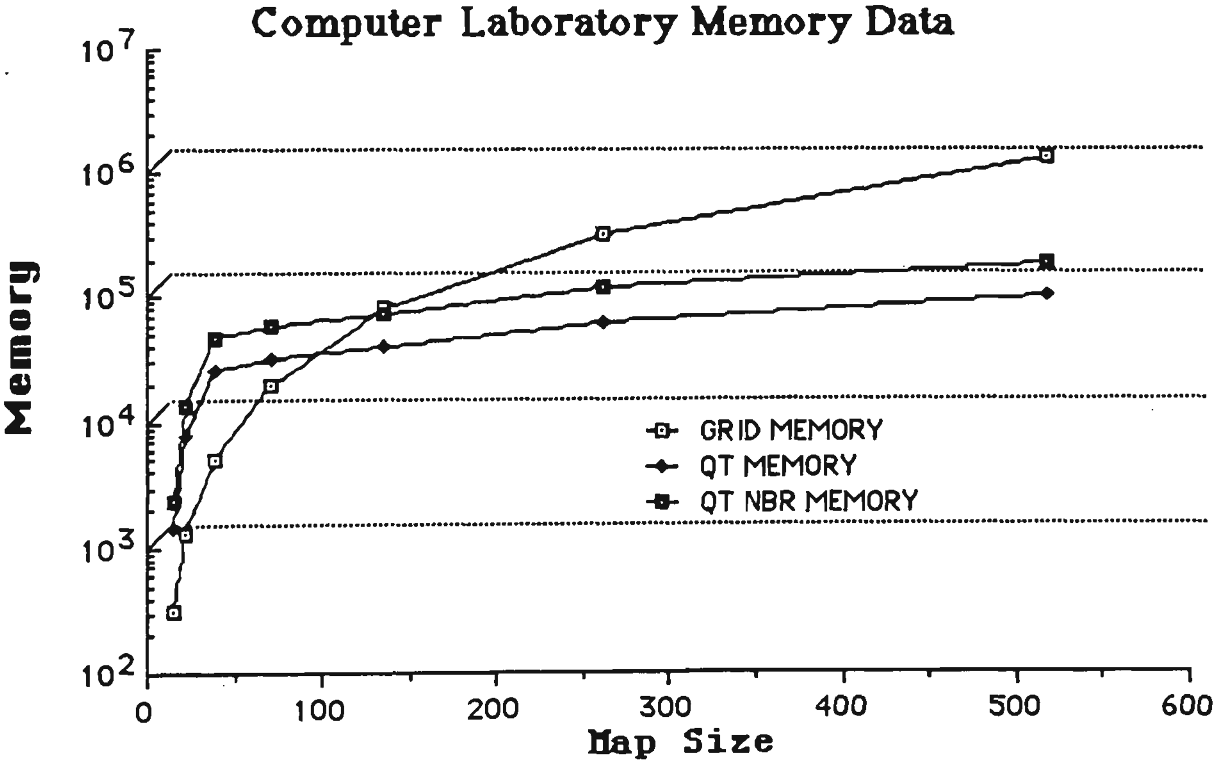
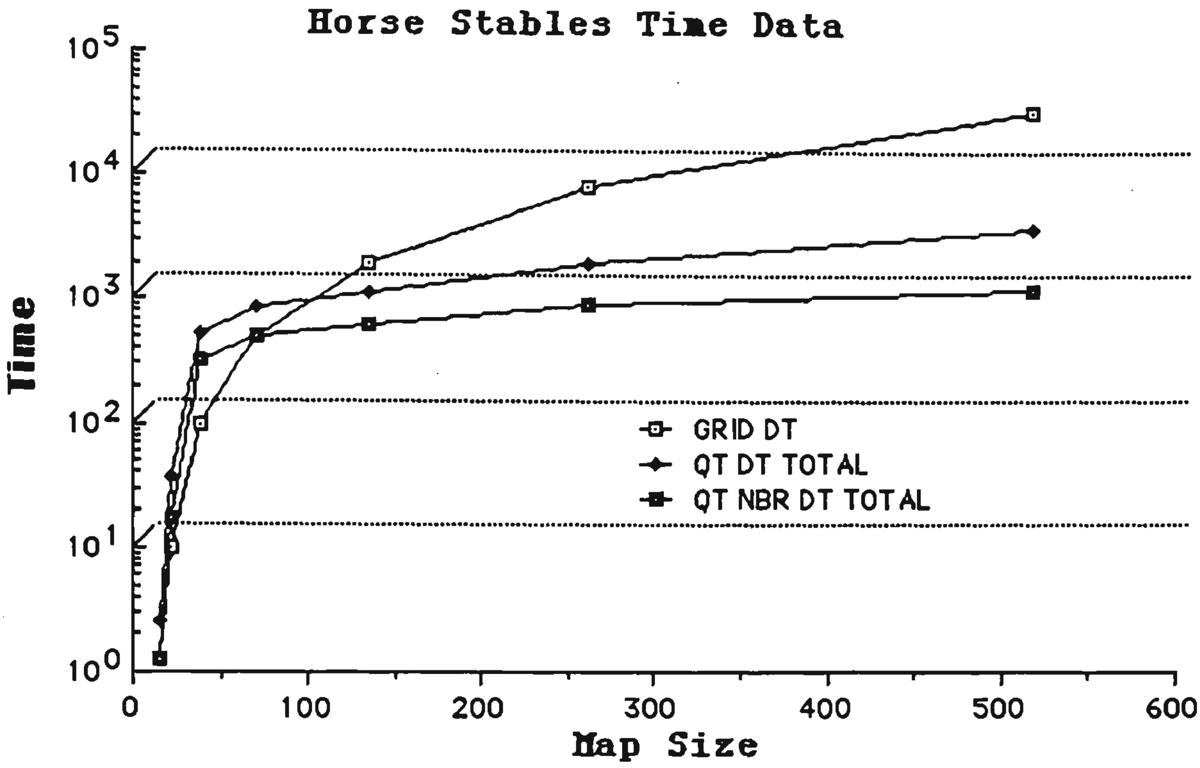


Figure 6.12

Path Planning statistics for the Computer Laboratory.



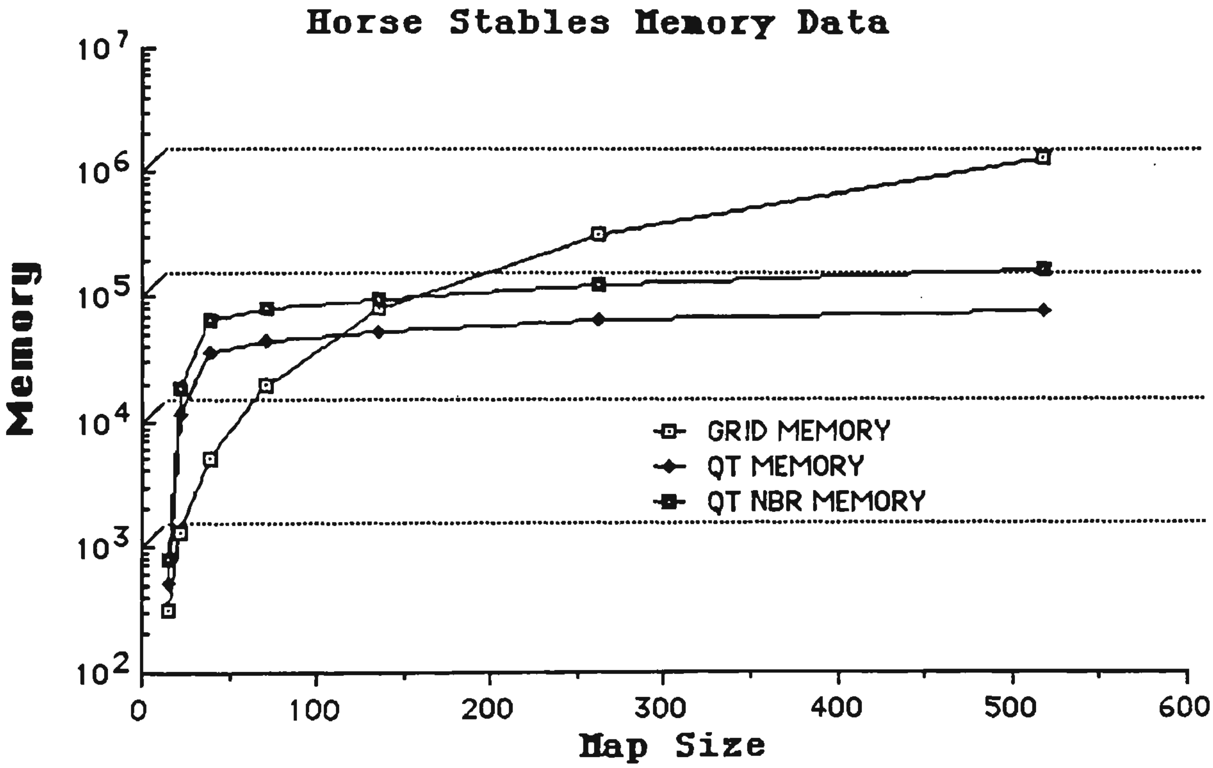
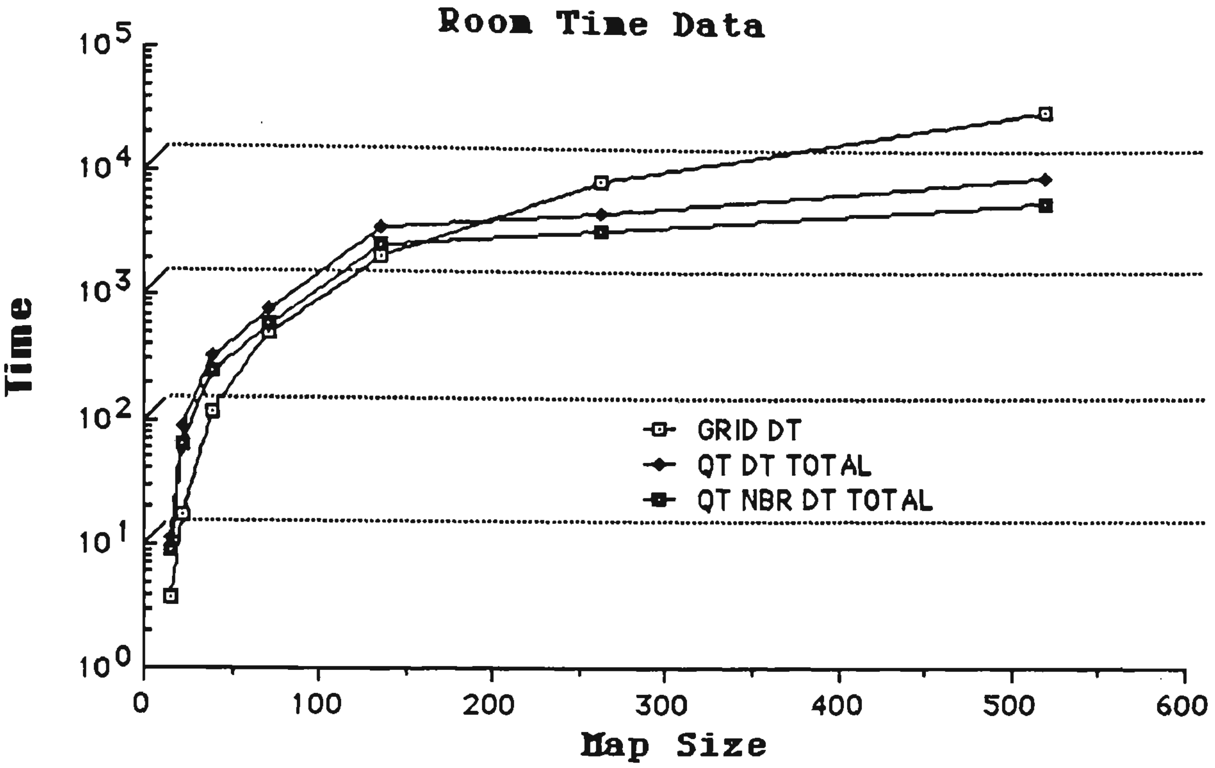


Figure 6.13

Path Planning statistics for the Horse Stables.



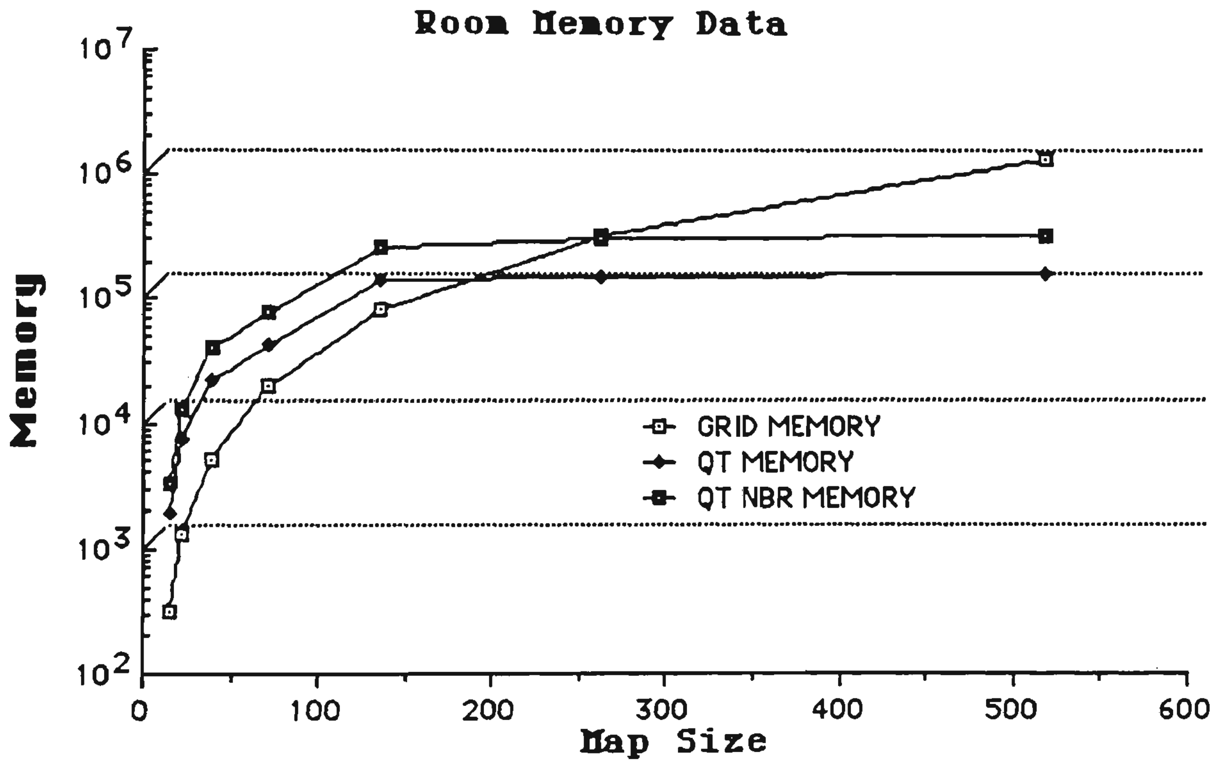


Figure 6.14

Path Planning statistics for the House with 2 Rooms.

Examining the results presented in Figures 6.12 - 6.14 led to the following conclusions. The memory requirements for all three environments begins to level out and grows at a small rate for environments greater than 64x64 cells in size. This is because the number of complex obstacles in all three environments is relatively small. The simple obstacles, such as walls, align to low resolution quadrants and therefore do not require any additional memory at higher map resolutions. Since the distance transform applied to quadtrees is dependent on the number of quadrants in the quadtree, the computation performance of the quadtree distance transform is significantly superior to the grid distance transform as the map resolution increases. The quadtree distance transform outperformed the distance transform applied to grids both in computation time and memory requirements for environment maps which are greater than or equal to 128x128 cells in size, with the exception of the room environment where the grid was superior in map sizes smaller than 256x256. If the path planning environments need to be precisely modelled then quadtrees offer significant savings both in memory and computational requirements for maps which are greater than 128x128 cells in size.

Computing the distance transform for quadtrees with neighbour lists for the three (3) indoor environments resulted in average computational savings of approximately 16% for the computer laboratory, 18% for the horse stables, and 26% for the room environment. For all three environments the computational savings were offset by a two fold increase in memory requirements.

6.6 Conclusions

This chapter compared the performances of the quadtree and grid distance transform path planners. Experiments were performed using a wide ranging variety of test environments on different map resolution sizes. From these experiments the following insights were gained.

Apart from the trivial case where an environment contains no obstacles quadtrees should not be considered unless the map resolution size is at least 32x32 cells. Grids are the most appropriate data structure for smaller map sizes i.e 8x8 and 16x16. If the required map resolution is suitable for quadtrees then the shape and distribution of the obstacles in the environment must be considered to decide whether or not the quadtree is the best data structure to model the environment.

It was shown that if the environment is made up of small pixel sized obstacles randomly distributed, the quadtree is an unsuitable data structure to model the environment. Similarly, if path planning is to be done in an environment containing a circular spiral quadtrees are also unsuitable. Grids should be used instead. Quadtrees are unsuitable in a maze environment unless the map resolution is at least 512x512 cells.

Clearly the above obstacle configurations are worst case situations and are not likely to occur in practice. If an environment contains a small number of obstacles the quadtree is the most efficient data structure. One obstacle environments can be represented in a quadtree with a map resolution as low as 32x32 cells. As the number of obstacles increases the required map resolution for the quadtree increases. It was shown that in environments containing up to three obstacles the quadtree was the most efficient data structure for map resolutions of 128x128 and higher. In environments which have more than three obstacles quadtrees are the preferred data structure if the map resolution is 256x256 and higher. The obstacle shapes in these experiments caused heavy fragmentation to the quadtree.

Further investigations were carried out using indoor environments with a mixture of obstacle shapes. Some obstacles caused minimal fragmentation of the quadtree while others caused heavier fragmentation. The results showed that quadtrees were the preferable data structure in map resolution sizes of 128x128 and higher in the majority of test cases. Quadtrees were the best suited data structure for all the test cases for map resolution sizes greater than 128x128 cells.

Quadtrees are a highly suitable data structure if it is a requirement that the obstacles in the environment be accurately modelled e.g for object recognition. At high map resolutions in cluttered environments quadtrees offer substantial memory savings in addition to the computational savings.

If path planning is to take place in a completely known environment, then the extra computational overhead of building neighbour lists can be justified. It was found in all the experiments that computing the distance transform with neighbour lists produced an average saving in computation time of at least 16%. As the environments become more complex the savings in time increased. Throughout all the experiments it was found that the penalty for the use of neighbour lists was a two fold increase in memory requirements.

The experiments that were performed in this chapter assumed that the environment was known and that the cost of constructing the quadtree was included in the path planning statistics. The Environment Exploration Algorithm (EEA) described in Chapter 4 operates in unknown environments. This algorithm progressively builds the quadtree while the environment is being explored and therefore the costs of constructing the quadtree are small. Also since the EEA assumes that unknown areas are free space, the cost of path planning will initially be low. As the knowledge of the environment increases the cost of path planning will steadily become greater. Path planning with grids will be expensive from the outset. It can therefore by safely assumed that the EEA will perform satisfactorily in learning environments of the type that were experimented with in this chapter.

Chapter 7
Path Planning for Mobile Robots with 3 DOF

7.1 Introduction

As reported in Chapter 3 much of the research effort into mobile robot path planning has concentrated on the problem of finding paths from a start position to a goal position by translation of the robot's body only. The problem of finding paths which require the rotation of the robot's body have been largely ignored. This chapter will present a new path planning algorithm for mobile robots which have 3 degrees of freedom (DOF) of movement, operating in cluttered environments. This algorithm (referred to as "3DOFA" for short) has the desirable property of potential field path planners, that of taking information about clearance from obstacles into account when planning paths. However this method does not have the potential field drawback of suffering from local minima problems. In addition the 3DOFA does not have the computational burden which is normally associated with 3 DOF path planners [Brooks *et. al.* 85, Lozano-Perez 83, Schwartz *et. al.* 83]. The 3DOFA is guaranteed to find a solution path if one exists. The 3DOFA is based on an extension to the path planning methodology of distance transforms. The remainder of this section will discuss what features are necessary and desirable for a 3 DOF mobile robot path planning algorithm. Section 7.2 presents the detail workings of the 3DOFA. Path planning results of the new algorithm are presented in Section 7.3. Finally in Section 7.4 a summary of conclusions about the new algorithm are presented.

Section 3.3.5 reviewed 3 DOF path planning. From this review a number of observations were made about the shortcomings of past approaches to this problem. The global approach [Brooks *et. al.* 85, Lozano-Perez 83, Schwartz *et. al.* 83] of building 3 dimensional graphs is extremely expensive in computational effort. Another drawback of the global approach is that it suffers from the "too close" problem since it assumes that

that the shortest path is the best path. The alternative to a global approach is a local approach [Donald 87] which uses heuristics to guide the search. A heuristic guided search greatly improves the execution time of path planning. However, heuristic search is prone to failure and the resulting solution path may be neither the shortest nor the safest. It is simply a negotiable path from the start configuration to the goal configuration.

The [Ilari *et. al.* 90] approach to 3 DOF path planning seeks to find the shortest global path between the start and goal locations which passes through the middle of the free space between obstacles. This path is then searched for a fine path of legal robot orientations using heuristics. A major drawback of the [Ilari *et. al.* 90] approach is that it suffers from the "too far" problem.

A similar idea to the [Ilari *et. al.* 90] approach has been presented by [Noborio *et. al.* 89] which is based on quadtrees. This method finds a coarse solution path of free space quadrants between the start and goal locations, such that the minimum width of the robot can pass through the solution path quadrants. The ~~course~~ path is refined using heuristics. coarse Since this method does not take clearance from obstacles information into account, and searches for the shortest negotiable path, this method can at times suffer the "too close" problem.

Both the [Ilari *et. al.* 90] and the [Noborio *et. al.* 89] approaches are susceptible to failure since they are based on heuristic searches. However the probability of this occurring is lower than for other heuristic methods, since the heuristics are being used to refine a global path which is likely to yield a solution.

In Section 5.3 an extension to the distance transform called the "path transform" was presented. Instead of propagating a distance from the goal wave front through free space, a new wave front is propagated which is a weighted sum of the distance from the goal together with a measure of discomfort from moving too close to obstacles. This has the effect of producing a distance transform which has the desirable properties of potential field path planners i.e. it avoids the "too close" and "too far" clearance from obstacle problems, without suffering from local minima problems.

The path transform offers an elegant and straight forward approach to finding a global path between a start and a goal compared to the [Ilari *et. al.* 90] search for a global path. The [Ilari *et. al.* 90] approach to finding a global path is based upon constructing a Voronoi diagram for all the free space regions in the environment and then processing the Voronoi diagram to remove branches in the diagram which are not relevant to path planning. Figure 7.1 (A) shows an example of the Illari method of building a Voronoi diagram. Figure 7.1 (B) shows the path planning version of the Voronoi diagram after it has been processed to remove the irrelevant branches in the diagram. This example highlights a deficiency of Voronoi diagrams. Voronoi diagrams are sensitive to noise on the boundaries of obstacles. The small triangular obtrusion on the boundary causes an unnecessary deviation in the global path. There seems to be no mechanism available to limit the effect of distant small obstacles.

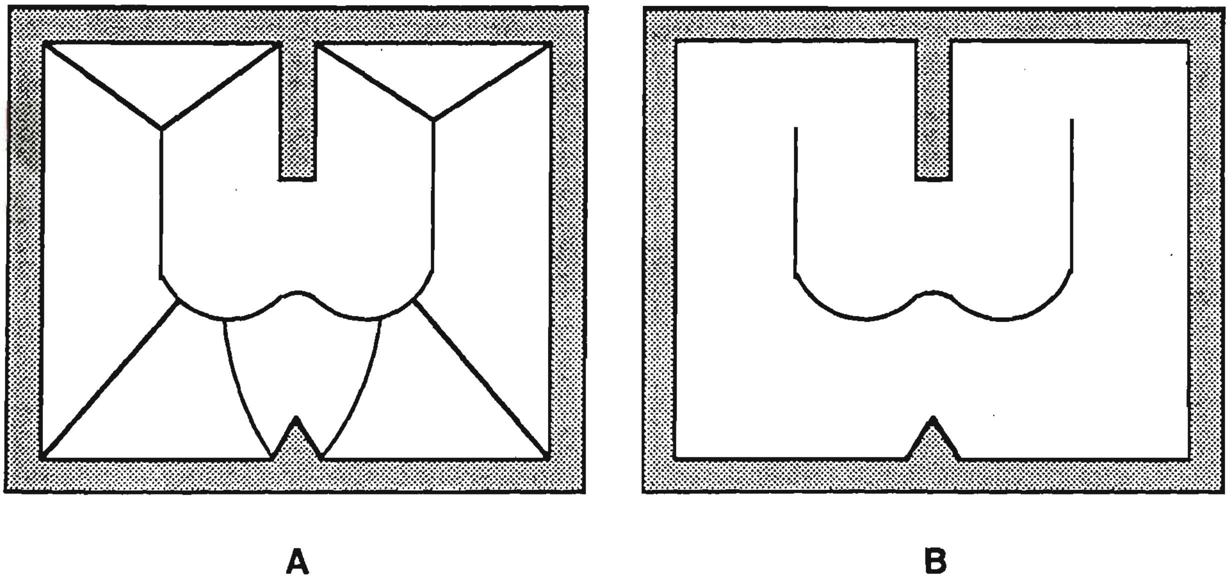


Figure 7.1

Ilari et. al. Global Path Planning.

The next step is to add the start and the goal locations to the global path network. Ilari joins the start and goal locations to the global path by computing the shortest straight line to the global path network which does not intersect with an obstacle. This practice has the nasty side affect of possibly creating non-optimal paths between the start and goal locations. Illari then searches the global path network using the A* search for a path which maximises clearance from obstacles while minimising the length of the path to the goal.

The Ilari approach to constructing the global path network is cumbersome and the resulting path network has serious limitations. The path transform can achieve the same objectives sought by Ilari without the drawbacks of Ilari's approach. The path transform is based upon combining the "obstacle transform" described in Section 5.4 with the distance transform. The obstacle transform represents the distance from each free space cell to the nearest obstacle. Joining the highest values in the obstacle transform with line segements will yield a Voronoi diagram. Instead of generating a Voronoi diagram a cost function is applied to the obstacle transform which varies inversely with the distance to the nearest obstacle. The cost function exerts its influence over a limited distance. Once the path transform has been generated, a solution path to the goal is known for each grid cell. This solution path minimises the distance to the goal while keeping a safe distance from obstacles.

The second stage of Ilari's path planner refines the global path to produce a 3 DOF path by using heuristics. Ilari notes the drawback of using a heuristic which is based solely on minimising the distance to the goal. Ilari proposes two types of heuristics to refine the global path. The first class of heuristic tries to select points in free space that keep the robot close to the goal while not straying from the global path. This heuristic is computed along the whole length of the global path prior to the search for a 3 DOF path. The path transform provides the same facility as this heuristic without the need for preprocessing. The path transform can be regarded as a heuristic which pushes a robot away from obstacles in the direction of the goal. In contrast the Ilari path planner pushes the robot toward the goal by the need to keep the robot on the global path, rather than the need to keep away from obstacles.

The second class of heuristic designed by Ilari has the purpose of keeping the body of the mobile robot aligned closely to the global path. This heuristic requires that every point in free space must be assigned a most suitable orientation value prior to path planning. A suitable orientation is an orientation which has the most chance of overcoming, with the fewest reorientations, future bottlenecks along the proposed path. The path transform does not provide an equivalent to this class of heuristic.

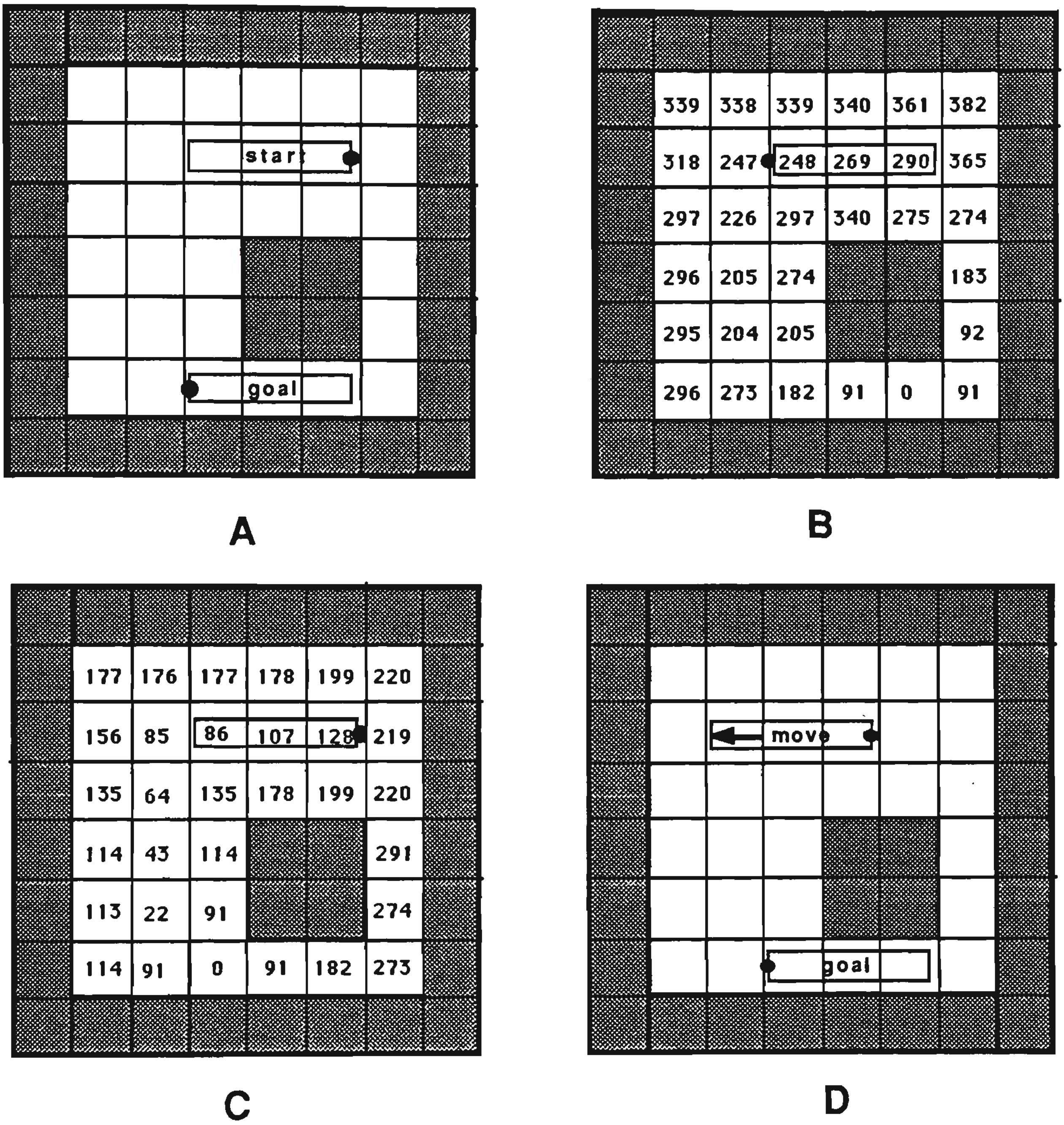
Section 7.2 presents a straight forward extension to the path transform which allows it to perform in an equivalent manner to the Ilari orientation heuristic. The extended path transform can then be used as the basis for a new algorithm (3DOFA) for planning paths for mobile robots with 3 DOF. The new algorithm does not have the computational burden which is normally associated with 3 DOF path planners and is guaranteed to find a solution path if one exists.

Section 7.3 presents the experimental results of implementing the 3DOFA. Finally in Section 7.4 the conclusions that were reached and the insights that were gained from investigating the problem of 3 DOF path planning are presented.

7.2 A New 3 DOF Path Planning Algorithm

The objective in path planning is for the robot to move from a start configuration to a goal configuration. The path transform can only guide a point robot to a point goal configuration. The single point path transform specifies a robot with two degrees of freedom (translation only along the x and y axis). Selecting one control point on the rectangular robot does not provide the path transform with enough information to determine whether or not the robot has reached the goal configuration. If we consider the robot to have two control points, one at each end, we have enough information to determine whether the goal configuration has been reached. The problem remains of how to extend the path transform to consider the movement of more than one control point on the robot.

A path transform is computed for each control point on the robot. The path transform will describe a solution path from the start configuration to the goal configuration for each control point. Figure 7.2 shows a robot with two control points and the path transforms which correspond with each control point. Figure 7.2 (A) shows a rectangular robot, start and goal configurations (the heavy dot discriminates between the two ends of the robot). The two control points which have been selected on the robot correspond to the mid points of the two ends of the robot. Figures 7.2 (B) and (C) show the path transforms for each control point. The movement of the individual control points is not independent. Only movements of the control points that do not violate the geometry of the robot are allowed. Consequently situations will arise where all the path transforms drive the robot into "conflict" situations, where all the control points are working against each other. A conflict resolution function is needed to handle this problem. This function selects a single control point, and the path transform values of this control point are used to guide the robot to the next grid cell. The conflict resolution function selects the control point with the largest path transform value, because this is the control point which is furthest from its goal configuration. The chosen control point is then moved closer to the goal. This is done by moving down the steepest descent gradient of the path transform (i.e. the neighbouring cell with the smallest path transform value). This algorithm uses a 4 connected grid i.e. diagonal neighbours are not considered, since diagonal paths tend to clip obstacles. The remaining control point is moved to a grid cell which does not violate the geometry of the robot, and does not cause a collision with obstacles in the environment. In Figure 7.2 the conflict resolution function selects the control point in Figure 7.2 (A). The robot moves towards the goal as shown in Figure 7.2 (D). Once a move has been successfully completed, the conflict resolution function selects the next control point to guide the next movement of the robot. This procedure is repeated until all the control points on the robot reach their respective goal configurations.



B

339	338	339	340	361	382
318	247	248	269	290	365
297	226	297	340	275	274
296	205	274			183
295	204	205			92
296	273	182	91	0	91

C

177	176	177	178	199	220
156	85	86	107	128	219
135	64	135	178	199	220
114	43	114			291
113	22	91			274
114	91	0	91	182	273

Figure 7.2

3 DOF Path Planning with Path Transforms.

Checking if a proposed move produces an illegal geometric configuration of the robot can be done in a straightforward manner. The control points must always be separated from each other by the same fixed distance. After each proposed move the distances between all the control points are calculated, and compared to the distances between the control points of the initial robot configuration.

A collision check must be made every time the robot is moved, because the obstacles in the environment were not expanded by the radius of the cylinder approximating the robot. This is not a trivial task and may seem to be an expensive computation. However a hierarchical collision testing procedure is used to implement collision testing based on the idea of distance space bubbles [Verwer 90]. Under the Verwer approach the robot is inside a space bubble which is of sufficient radius to encapsulate the entire robot. The distance to the closest obstacle is measured from the centre of the bubble. If this distance is greater than the radius of the bubble then no collision has occurred. If on the other hand this distance is less than the radius of the bubble, then a collision is possible and further testing is necessary. The bubble encapsulating the robot is burst and the robot is then enclosed with two smaller bubbles, and the test for collision is repeated once more. This process continues until it can be confirmed that no collision has occurred, or the bubbles reach a resolution, beyond which no further testing is done. In this case the conclusion is made that a collision has occurred. Refer to Figure 7.3 for an example of collision detection using the bubbles hierarchy. This figure shows from left to right the series of tests for collision between a rectangular robot and an obstacle. The first figure shows that a collision has occurred with the enclosing bubble. The robot is split along its longest side, giving two smaller rectangles. These rectangles are enclosed by bubbles, and these bubbles tested for collision.

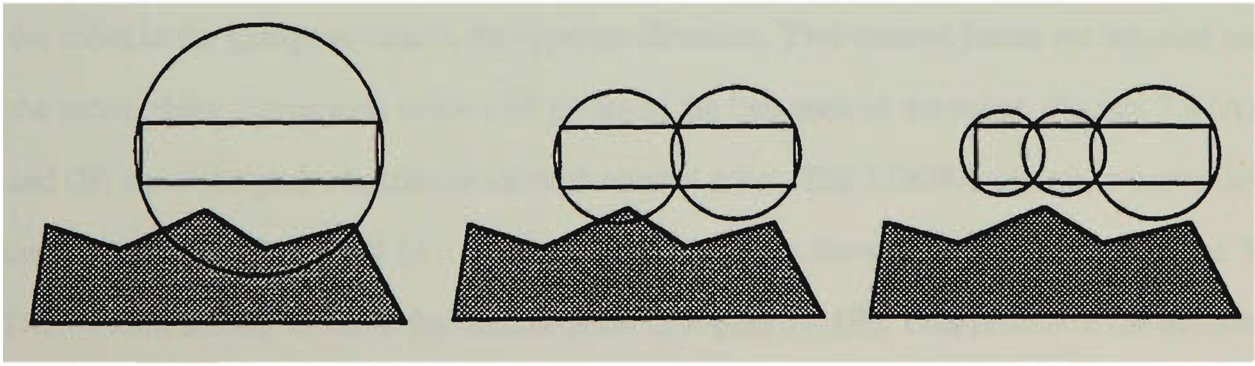


Figure 7.3

Distance Space Bubble Hierarchy.

This hierarchical collision checking procedure suits this path planning method for the following reasons. During the computation process of the path transform, the obstacle transform is calculated. The obstacle transform gives for each free space cell the distance to the closest obstacle. Thus the information necessary for checking the distance of a space bubble from its nearest obstacle is already available. Secondly this path planner deliberately steers the robot away from obstacles, and it only approaches obstacles if it is absolutely necessary. Thus the bubble hierarchy is very suitable to this path planning method, since checking for collisions will be done primarily at the highest levels. This will give a cheap and efficient collision detection mechanism. The bubble hierarchy is computed prior to path planning and is stored as a binary tree. The bubble hierarchy can generate superfluous bubbles i.e. bubbles which only cover interior parts of the robot. It is only necessary to store the distance space bubbles which enclose the outside surfaces of the robot. When the bubble hierarchy is being computed the superfluous bubbles are omitted and not stored in the binary tree.

Following the steepest descent path for control points with the largest path transform values will not necessarily guarantee that the robot will reach its desired goal configuration. A "deadlock" situation can arise where the robot is caught in the situation where one control point moves the robot in one direction, and on the very next move another control point pushes the robot back in the opposite direction. This produces an undesirable oscillating behaviour. Figure 7.4 shows an example of this behaviour. Figure 7.4 (A) shows a rectangular robot parked in a garage. The path planning task is to repark the robot in the garage to face in the opposite direction. Two control points are selected on the robot; these correspond to the mid points of the two ends of the robot. Figures 7.4 (A) and (B) show the path transforms for each control point. The 3 DOFA selects to move the control point in Figure 7.4 (A). The resulting motion is shown in Figure 7.4 (C). The 3 DOFA then selects to move the control point in Figure 7.4 (B). This results in the motion shown in Figure 7.4 (D).

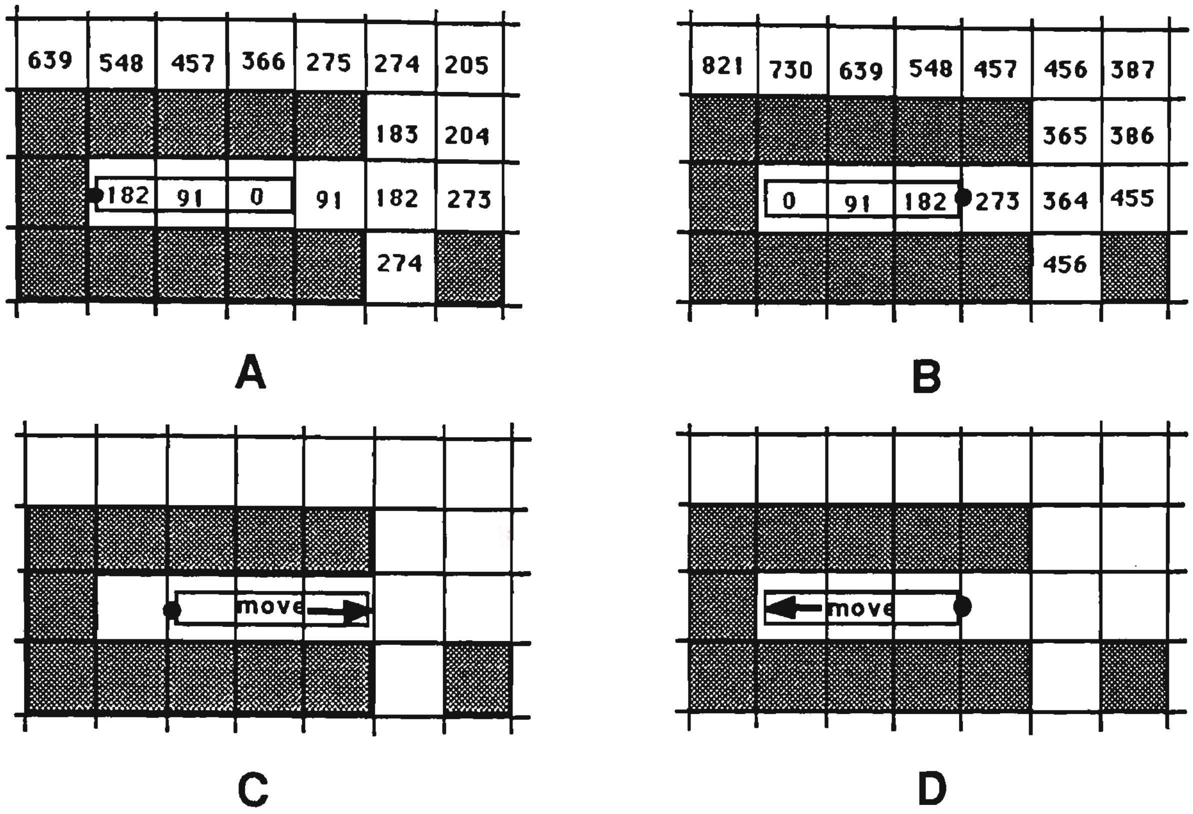


Figure 7.4

Deadlock situations during path planning.

The solution to this problem is to keep a record of all the moves made by the robot. This information is kept in an "open" list; each entry in the list records the grid cells that all the control points are located in. Before a robot move is attempted this list is checked for the proposed robot configuration. If the proposed move has already been performed earlier, then this move is abandoned and another move is selected. The next move that is selected for a control point is the one with the smallest path transform value, other than the grid cell which has been rejected. This mechanism solves the deadlock situation. At the end of path planning the open list shows all the intermediate moves of the control points between the start and goal configurations.

The 3DOFA is a similar to the A* [Hart *et. al.* 68] algorithm because it uses an open list and backtracking to select other grid cells as candidates for the next move. The A* algorithm uses the straight line distance to the goal, as the heuristic to guide the robot to the goal. The 3DOFA uses the path transform values as the heuristic to guide the robot to the goal. The path transform values are a much more powerful heuristic than the straight line distance to the goal, and therefore will guide the search through configuration space much more efficiently. The path transform produces a potential field valley between obstacles, through which the robot moves. By choosing suitable control points on the robot, the robot will naturally orientate itself with the potential contours of the path transform. Thus using the path transform with two control points creates an orientation heuristic similar to the [Ilari *et. al.* 90] orientation heuristic but without the extra work.

Since the path transform acts as a heuristic to guide the robot to the goal, the robot can be guided into situations where no solution path exists. For example consider the case of moving a long rectangular robot down a narrow corridor, and around a sharp bend (as shown in Figure 7.5). If the robot is too long, there is no solution path, even though the path transform indicates there is a path to the goal for a point robot. The robot is trapped.

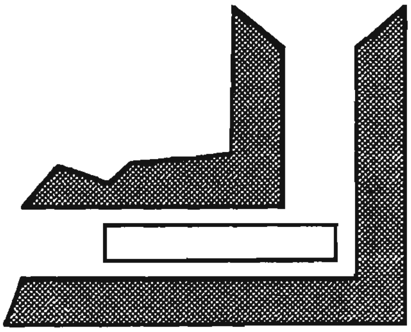


Figure 7.5

Move a long rectangular robot around a corner in a narrow corridor.

The only option is to backtrack along the open list, and move the invalid configurations from the open list to a "closed" list. The elements contained in the closed list are configurations of the robot, which have been tried and are known not to belong to the solution path. This idea is similar to the way the A* search escapes dead end situations. Figure 7.6 shows an example of the A* algorithm backtracking. Figures 7.6 (A) - (D) show the A* search algorithm backtracking out of a dead end. Once the robot has backed out the the dead end and then moves along the correct path to the goal, the open list will contain only the moves from the start to the goal which bypass the dead end. The closed list will contain all the moves which were tried to escape the dead end.

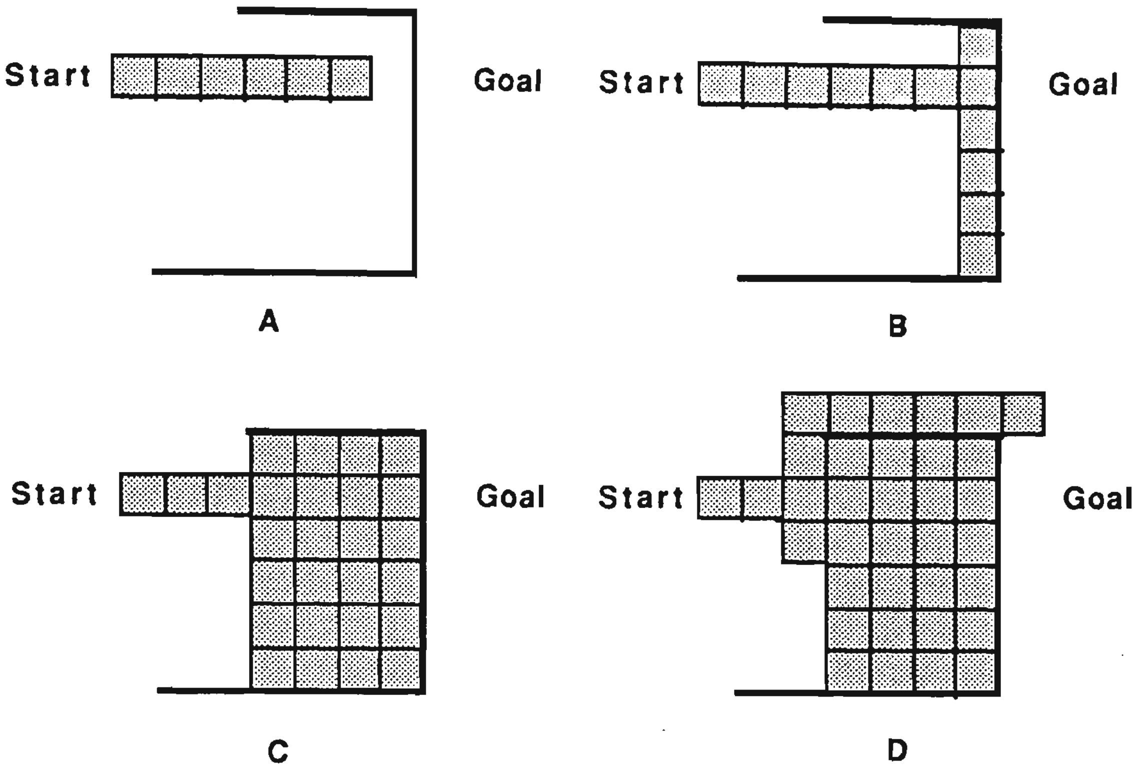


Figure 7.6

The A* search algorithm escaping from a dead end.

If a solution path for the mobile robot to move between the start and goal configurations does not exist, for example if the robot is surrounded by impassable narrow corridors, it should be understood that this path planning algorithm will search the entire solution space. Although this is a drawback, a path planner is usually invoked only when it is known that a solution path between the start and goal configurations exists. Due to the exhaustive search properties of this algorithm it should be clear that the algorithm will find the solution to a path planning problem if it exists.

The situation can arise that a solution path can be unnecessarily long. The open list is checked for any configuration of the robot earlier on in the solution path which could directly reach the current robot configuration. If the solution path can be shortened, the robot configurations which have been by-passed are deleted from the open list and are added to the closed list. Figure 7.7 shows an example of a solution path being shortened. Figure 7.7 (A) shows the 3DOFA rotating the robot about one control point. After the robot rotated to the location shown with the broken line in Figure 7.7 (A), the 3DOFA deduced that this location could also be reached with less effort by rotating in the opposite direction. This is shown in Figure 7.7 (B).

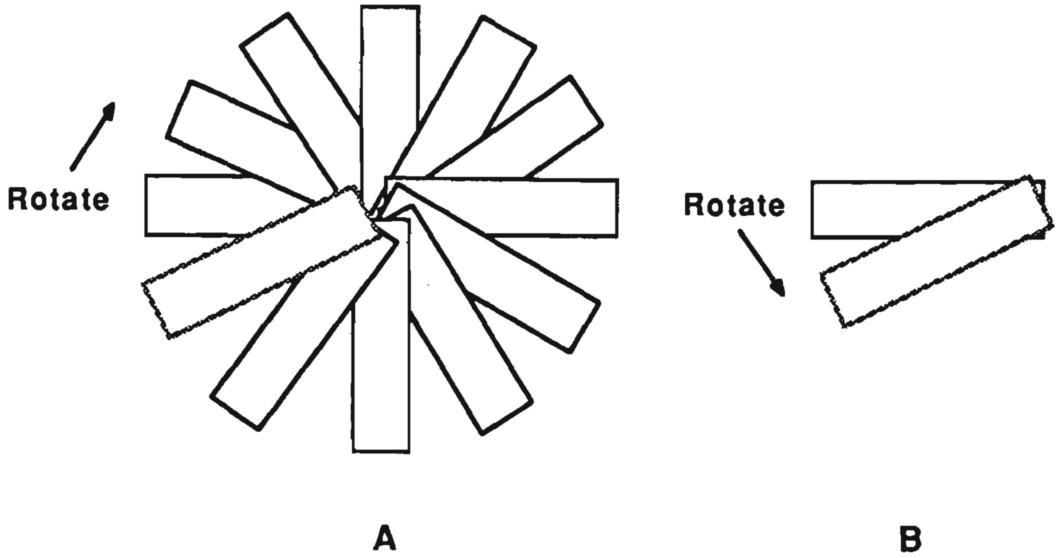


Figure 7.7
Shortening the solution path.

The detailed algorithms for this 3 DOF path planning method are as follows. The pseudo code algorithm for the 3DOFA is presented in the function PATH_PLANNING which is shown in Algorithm 7.1. The function PATH_PLANNING takes as input the environment grid map and the start and goal configurations of two control points. This function returns a list of moves which contains the solution path between the start and goal configurations.

The PATH_PLANNING function contains two calls to the function PATH_TRANSFORM. The PATH_TRANSFORM function computes the path transform that guides a control point to its goal configuration. The path transform is computed for both control points. The pseudo code details of the PATH_TRANSFORM function are not provided since these details were presented in Section 5.4.

In the next phase of planning the PATH_PLANNING function must decide which control point to move. The job of deciding which control point to move is done by the function MAX_MIN. The pseudo code details of this function are shown in Algorithm 7.2.

```
function PATH_PLANNING( map, bubbles, $start_{p1}$, $start_{p2}$, $goal_{p1}$, $goal_{p2}$ )
    open = NIL
    closed = NIL
    $dt_{p1}$ = PATH_TRANSFORM( map, $start_{p1}$, $goal_{p1}$ )
    $dt_{p2}$ = PATH_TRANSFORM( map, $start_{p2}$, $goal_{p2}$ )
    RECORD( $start_{p1}$, $start_{p2}$, open )
    while ( $dt_{p1}[start_{p1}] \neq 0$ and $dt_{p2}[start_{p2}] \neq 0$ ) do
        MAX_MIN( $dt_{p1}$, $start_{p1}$, $dt_{p2}$, $start_{p2}$, min, max )
        valid = FALSE
        $neighbours_{max}$ = FIND_NEIGHBOURS( $dt_{max}$, max )
        while ( $neighbours_{max} \neq$ NIL and not ( VALID ) ) do
            $move_{max}$ = NEXT( $neighbours_{max}$ )
            $neighbours_{min}$ = FIND_NEIGHBOURS( $dt_{min}$, min )
            while ($neighbours_{min} \neq$ NIL and not ( VALID ) ) do
                $move_{min}$ = NEXT( $neighbours_{min}$ )
                if ( VALID_MOVE( $move_{max}$, $move_{min}$, map, bubbles, open, closed
                    RECORD( $move_{max}$, $move_{min}$, open )
                    SHORTEN( open, closed)
                    valid = TRUE
                end if
            end do
        end do
        if not ( valid ) then
            RECORD( $start_{p1}$, $start_{p2}$, closed )
            BACKTRACK( open, $start_{p1}$, $start_{p2}$ )
        else
            SET( $move_{max}$, $move_{min}$, $start_{p1}$, $start_{p2}$ )
    end do
    return ( open )
end function
```

Algorithm 7.1

3DOF Path Planning Algorithm.

```
procedure MAX_MIN( $dt_{p1}$, $start_{p1}$, $dt_{p2}$, $start_{p2}$, min, max )
    if ( $dt_{p1}[start_{p1}] > dt_{p2}[start_{p2}]$ ) then
        max = $start_{p1}$
        min = $start_{p2}$
    else
        max = $start_{p2}$
        min = $start_{p1}$
    end if
end procedure
```

Algorithm 7.2

Select the Control Point that needs to be moved.

The next step of path planning is to decide where to move the selected control point. The FIND_NEIGHBOURS function which is described in Algorithm 7.3, finds all the possible moves for a control point, and then ranks these moves in descending path transform value order.

```
function FIND_NEIGHBOURS( dt, start )
  neighbours = NIL
  neighbours = ADD( neighbours, start )
  for ( i=1 to 4 )
       neighbours = ADD( neighbours, GET_NEIGHBOUR( i, start ) )
  neighbours = SORT( neighbours )
  return ( neighbours )
end function
```

Algorithm 7.3

Find all the Possible Moves for a Control Point and Rank them.

The next stage of planning performed by the PATH_PLANNING is to step through the list of desirable moves returned by the FIND_NEIGHBOURS function. The NEXT function is responsible for selecting the next best move which should be attempted. The PATH_PLANNING function attempts to move both the control points to the position selected by the NEXT function. The implementation particulars of the NEXT function are straightforward and are therefore not presented. Once the next move has been selected the VALID_MOVE function which is described in Algorithm 7.4 tests if the proposed move is legal. The VALID_MOVE function evaluates four conditions to test whether or not a proposed move is valid. The function tests that the proposed move does not violate the geometry of the robot, the move does not cause a collision with any obstacle and the move is absent from the open and closed lists.

```
function VALID_MOVE( $move_{max}$, $move_{min}$, map, bubbles, open, closed )
  if    ( VALID_GEOMETRY( $move_{max}$, $move_{min}$ ) ) and
        ( NO_COLLISION( $move_{max}$, $move_{min}$, map, bubbles ) ) and
        ( ABSENT( $move_{max}$, $move_{min}$, closed ) ) and
        ( ABSENT( $move_{max}$, $move_{min}$, open ) ) then
              valid = TRUE
  else
       valid = FALSE
  end if
  return ( valid )
end function
```

Algorithm 7.4

Check if a proposed move is valid.

The VALID_MOVE function checks that the proposed move does not violate the geometry of the robot by insuring that after the move the control points are still separated by a fixed distance. This check is performed by the function VALID_GEOMETRY, which is described in Algorithm 7.5. This function calculates the distance between the new locations of the control points and compares this against a precomputed constant called CONTROL_POINT_DISTANCE. This constant represents the distance between the two control points on the robot.

```
function VALID_GEOMETRY( $move_{max}$, $move_{min}$ )
   d = DISTANCE( $move_{max}$, $move_{min}$ )
   if ( d = CONTROL_POINT_DISTANCE ) then
      valid = TRUE
   else
      valid = FALSE
   end if
   return ( valid )
end function
```

Algorithm 7.5

Check if the separation of control points is valid.

The next check done by VALID_MOVE is to verify that the proposed move does not cause a collision with any obstacles in the environment. This check is done by the function NO_COLLISION which is described in Algorithm 7.6 This function uses distance space bubbles to check for collisions. The distance space bubbles are precomputed prior to path planning and are stored in a binary tree. The function performs operations on the distance space bubble which is stored at the root of the binary tree. The first operation performed on the distance space bubble is to locate the bubble's centre to a grid cell on the obstacle transform map. The next operation checks if the proposed robot move is collision free. This is done by checking that the radius of the distance space bubble is less than the obstacle transform value stored in the grid cell to which the centre of the bubble was located. If the radius of the distance space bubble is greater than the obstacle transform value stored in the grid cell then further collision checking is necessary. Further collision checking is done by splitting the current distance space bubble into two smaller bubbles, and then recursively calling the NO_COLLISIONS function to check the two smaller distance space bubbles. Splitting a distance space bubble is straightforward and is done by finding the two descendant children of the current distance space bubble in the precomputed binary tree.

```
function NO_COLLISIONS( $move_{max}$, $move_{min}$, map, bubbles )
   if ( EXIST( bubbles ) ) then
      pos = FIND(bubbles, $move_{max}$, $move_{min}$ )
      if ( bubbles.radius < map[pos].ot ) then
         valid = TRUE
      else
         pos = SPLIT(bubbles, bubble1, bubble2 )
         valid = NO_COLLISIONS( $move_{max}$, $move_{min}$, map, bubble1 )
         if ( valid ) then
            valid = NO_COLLISIONS( $move_{max}$, $move_{min}$, map, bubble2 )
         end if
      end if
   else
      valid = FALSE
   end if
   return ( valid )
end function
```

Algorithm 7.6

Check for collisions using distance space bubbles.

The last two checks in the VALID_MOVE function ensure that the proposed move has not been tried previously. The first check ensures that the proposed move is not present on the closed list of moves. The closed list contains moves which are known not to belong to the solution path. The final check ensures that the proposed move is not present on the open list of moves. The open list contains moves which are known to belong to the solution path. The checking of the open and closed lists is done by the ABSENT function. The implementation particulars of the ABSENT function are straight forward and are therefore not presented.

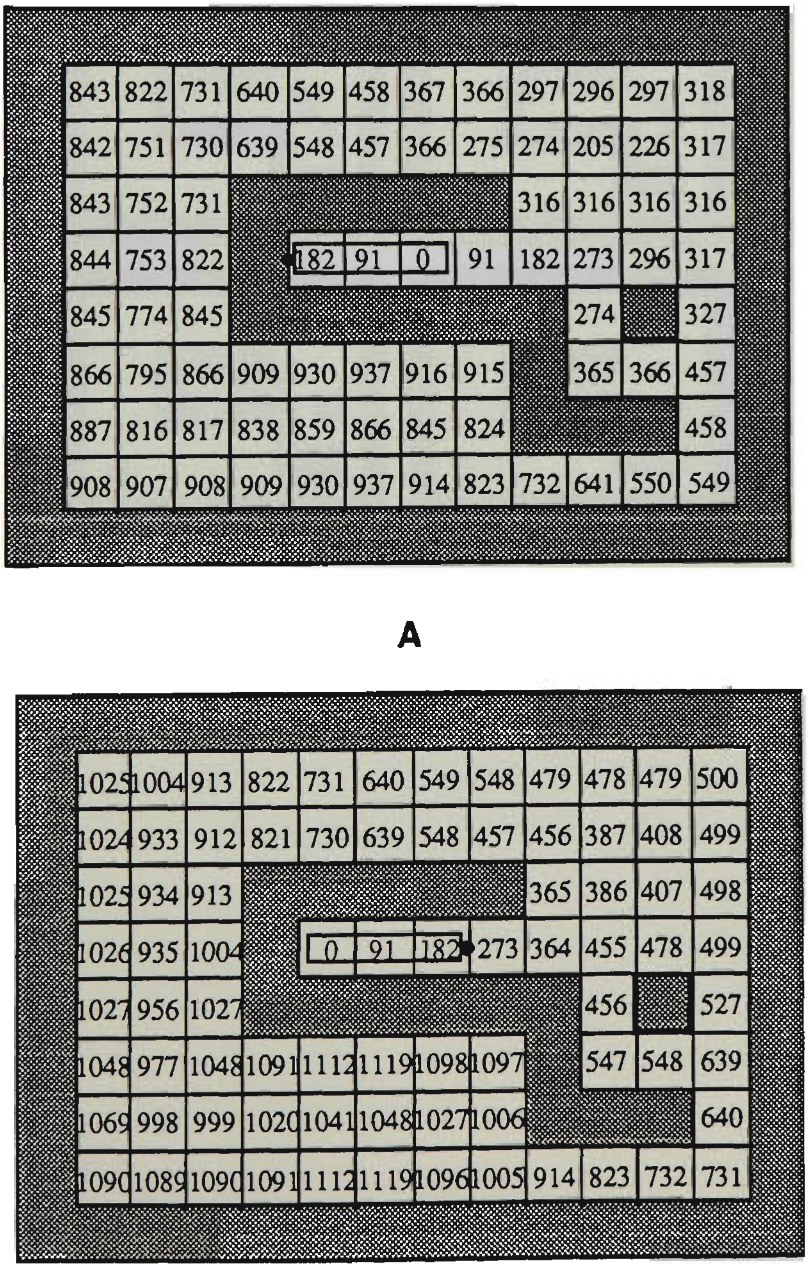
If the function VALID_MOVE determines that the proposed move is legal the PATH_PLANNING function uses the procedure RECORD to add the new move to the open list. The SHORTEN function checks if the solution path described by the open list can be shortened. This function checks if the last robot position recorded on the open list can be reached from previous moves on the open list. If the solution path can be shortened the SHORTEN function transfers the moves that are to be bypassed to the closed list. The implementation particulars of the RECORD and SHORTEN functions is straight forward and are therefore not presented.

If all the possible moves which are available to the robot at a particular location in the environment are invalid, the PATH_PLANNING function adds the current location of the robot onto the closed list using the RECORD function. This indicates that no solution path exists using this configuration of control points. The PATH_PLANNING function must now back track along the open list using the BACKTRACK function to enable the robot to escape the dead end. Once a move has been successfully performed the SET procedure marks the current location of the control points as the start locations for the next iteration of the search for the next move. The PATH_PLANNING function keeps generating moves, attempting moves, and recording valid moves until both the control points reach their goal configurations. The implementation particulars of the BACKTRACK and SET functions is straightforward and are therefore not presented.

7.3 The Results

The 3DOFA was implemented in C programming language on a SUN 4/260 computer operating under the UNIX operating system. The implementation did not provide a graphics animation of the robot moving from the start configuration to the goal configuration. The aim of this research was to prove that the new 3DOFA worked, so a user interface was not considered to be essential. The 3DOF computer program provided the list of robot moves that needed to be made to solve the path planning problem.

Results are provided for solving the path planning problem which was posed in Figure 7.4, where a rectangular robot had to be reparked in the same location but facing in the opposite direction. Two control points have been selected on the mid points of the ends of the robot. The path transforms for each control point are given in Figures 7.8 (A) and (B). The computer program provided a list of moves that were attempted to solve the problem, in addition to the final solution path. The solution path was computed in less than 4 seconds of elapsed running time. The output from the computer program was converted by hand into a graphical representation. The graphical details of the search for a solution path are shown in Figure 7.9. Figures 7.9 (A) - (P) show the stages of search for a solution path from the start to the goal configurations for the rectangular robot. The dot on the robot marks the control point which is guiding the path planner. The search can take the path planner into dead end situations as shown in Figure 7.9 (H). However when the search reaches the stage shown in Figure 7.9 (J), the planner deduces the that the current configuration can be reached from the configuration shown in Figure 7.9 (G). The solution path is amended not to include the configurations shown in Figures 7.9 (H) and (I). Once the robot reaches the goal, the complete solution path can be determined. The complete solution path is shown in Figures 7.9 (A) - (G) and (J) - (P).



843	822	731	640	549	458	367	366	297	296	297	318
842	751	730	639	548	457	366	275	274	205	226	317
843	752	731						316	316	316	316
844	753	822		182	91	0	91	182	273	296	317
845	774	845							274		327
866	795	866	909	930	937	916	915		365	366	457
887	816	817	838	859	866	845	824				458
908	907	908	909	930	937	914	823	732	641	550	549

A

1025	1004	913	822	731	640	549	548	479	478	479	500
1024	933	912	821	730	639	548	457	456	387	408	499
1025	934	913						365	386	407	498
1026	935	1004		0	91	182	273	364	455	478	499
1027	956	1027							456		527
1048	977	1048	1091	1112	1119	1098	1097		547	548	639
1069	998	999	1020	1041	1048	1027	1006				640
1090	1089	1090	1091	1112	1119	1096	1005	914	823	732	731

B

Figure 7.8

The path transforms for each control point to repark the robot in the opposite direction.

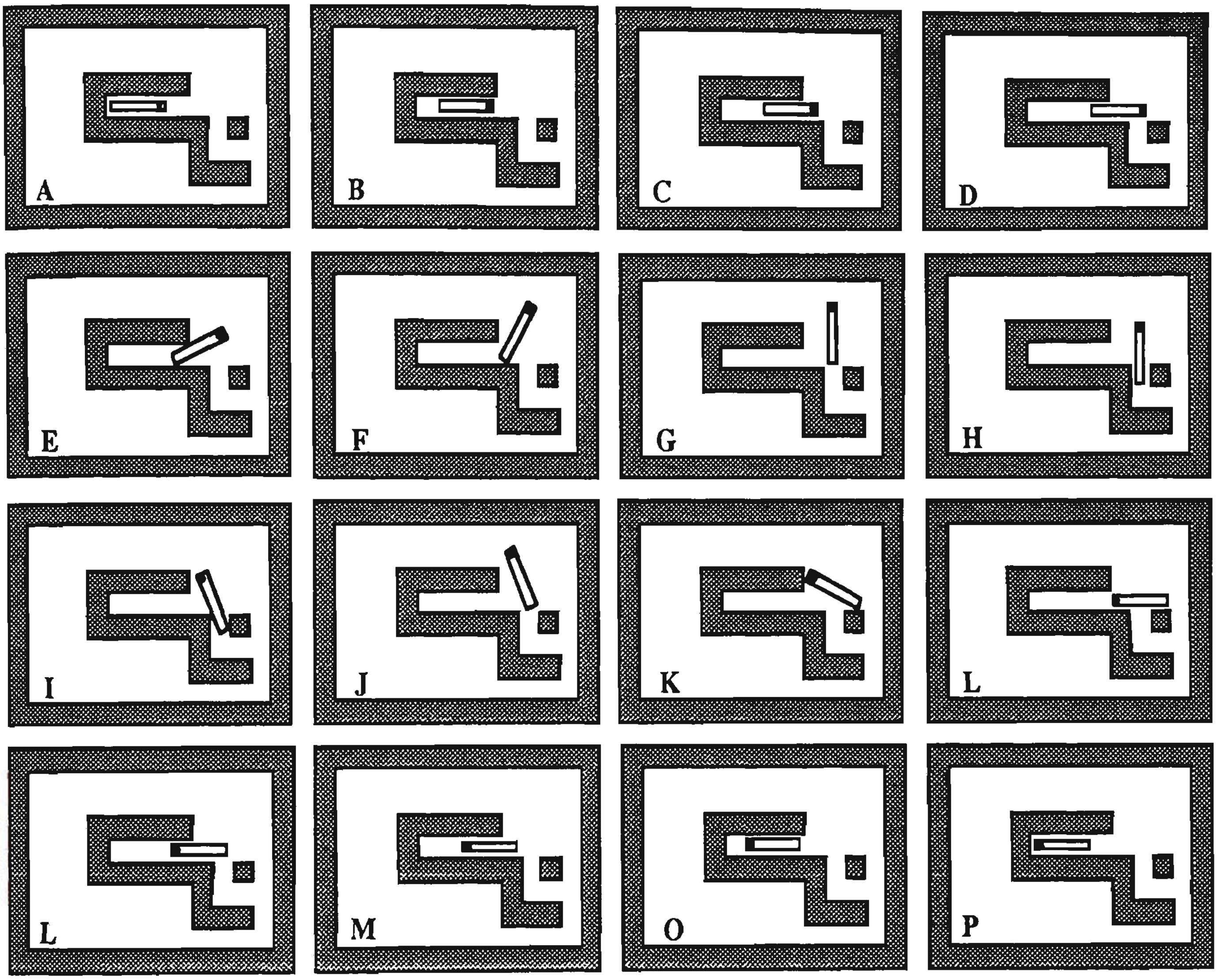


Figure 7.9

Diagrams A - P show the stages of search for a solution path from the start to the goal configurations for a rectangular robot. The complete solution path is shown in Diagrams A - G and J - P.

Another path planning experiment was performed using the same rectangular robot in the same environment as the one shown in Figure 7.8. The experiment involved moving the robot from a different start configuration to the same goal configuration shown in Figure 7.8. The path transforms computed in Figure 7.8 were applicable to this experiment. Figure 7.10 displays the final solution path between the start and goal configurations. The solution path for this experiment was computed in just over 7 seconds of elapsed running time. In this experiment the path transform values initially caused the path planner to move the robot into the narrow impassable corridor in the bottom right hand corner of the environment. When the path planner realised that it could not negotiate the bend, the path planner back tracked out of the narrow corridor. The planner then investigated an alternative solution path for the robot. The path transform values kept the solution path in the middle of the available free space areas in the environment. Using the path planning strategy that favours moving the control point which is furthest from its goal produces satisfactory solution paths. This feature is shown in this experiment by the neat move the robot performs as it swings around to the correct orientation before parking in the garage.

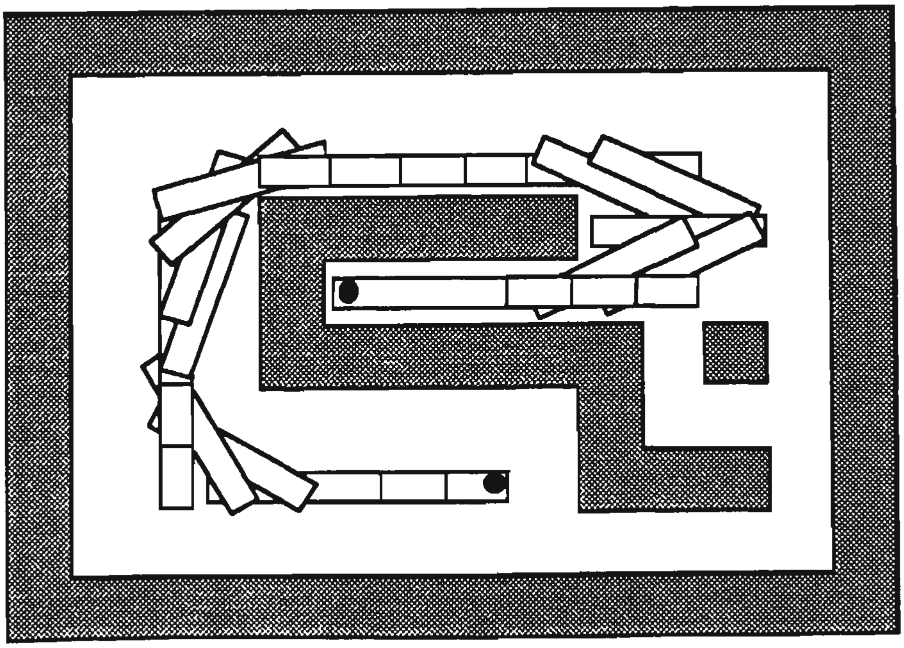


Figure 7.10

3DOF Path Planning Experiment .

7.4 The Conclusions

This chapter presented a new path planning algorithm for robots with 3 degrees of freedom. This algorithm is based upon selecting two control points on the robot and then constructing a path transform of the robot work space for each control point. The path transform functions as a heuristic to guide the search through the work place for a solution path. The path transform is an effective guide since it steers the search into areas of the work space where solutions are more likely. The path transform has the additional benefit of functioning as an orientation heuristic. The control points are naturally driven into the valleys of the path transform, thus minimising potential collisions. Collision detection is implemented using a hierarchy of space bubbles. Space bubbles allow for the quick testing of collisions between robot and obstacles in the environment. Experimentation showed that the new algorithm yields solutions to non-trivial path planning problems that can be computed quickly.

A disadvantage of this path planner is only suitable for 2-dimensional path planning problems. This is because the number of discrete configurations of the robot is exponential with the number of DOF's. A robot operating in 3-dimensional space has typically 6 DOF. Despite this disadvantage the 3DOFA algorithm is superior to the 3 DOF path planner reported by [Brooks *et. al.* 85] in terms of time complexity and quality of the solution path. The savings in time are due to the search of a reduced solution space. Quality of solution path means that the algorithm takes into account the discomfort of approaching obstacles too closely, in addition to minimising the distance to the goal.

The 3DOF path planner reported by [Noborio *et. al.* 89] was shown to be an order of magnitude faster than the the planner reported by [Brooks *et. al.* 85]. Given the different hardware platforms used by [Noborio *et. al.* 89] and the work reported in this research it is difficult to conclude which path planner is fastest. However given the increasing expense of calculating the grid distance transform as the resolution of the map increases, it unlikely that the 3DOFA would out perform the [Noborio *et. al.* 89] path planner.

It was shown by [Ilari *et. al.* 90] that their path planner is in the order of 100 to 200 times faster in search than path planners which only use distance to the goal as the search heuristic. Since the algorithm reported in this research uses similar heuristics to those used by Ilari *et. al.*, the 3DOFA algorithm should perform with a comparable efficency. However the 3DOFA has a number of advantages over the Ilari *et. al.* algorithm. The 3DOFA does not require the two stage search used by Ilari *et. al.* to find a solution path. By using a single stage search the 3DOFA is able to elegantly backtrack if the planner chooses to explore a path to the goal which has no solution. Backtracking with the Ilari *et. al.* algorithm is awkward since this planner keeps the robot on the global path, and the planner cannot explore alternatives. The Ilari *et. al.* approach to backtracking requires checking whether the robot is trapped. If the robot is trapped then the current global path is removed from the path network and a search for a new global path is performed. However the most important advantage the 3DOFA has over the Ilari *et. al.* algorithm is the quality of the solution path. The 3DOFA does not suffer from the "too far" problem and the final solution path does not require a path smoothing process. Ilari *et. al.* smooth the final solution path to remedy the deficiencies of the planning process.

Chapter 8
Conclusions and Further Work

8.1 Conclusions

In Chapter 1 it was stated that the goals of the research reported in this thesis were to further the development of data structures and algorithms in four areas of mobile robotics. Namely:

- * Environment Mapping with sonar range data.
- * Path Planning for mobile robots.
- * Path Planning behaviours for mobile robots.
- * Path Planning for mobile robots with 3 degrees of freedom.

An environment mapping method using sonar range data was presented in Chapter 2 of this thesis. This method allowed a mobile robot to map an unknown indoor environment. It was demonstrated how high resolution maps of indoor environments could be produced using the low resolution Polaroid Corp. Ultrasonic Rangefinder. The noise and uncertainty of sonar data was handled by applying the sonar mapping test. The sonar mapping test is a new approach to environment mapping, and it was shown to discriminate effectively against false reflections of sonar sound waves, thus allowing the mobile robot to produce accurate maps of the environment. The map was sufficiently rich in detail that it could be used by higher level mobile robot navigation functions such as path planning, object recognition etc. The mapping technique described in this research yields an inexpensive and reasonably fast method for mapping indoor environments.

In all the research work reported in this thesis it was assumed that an accurate measure of the robot's true location within the environment was known at all times. The reported work used dead reckoning to track the robot's position. All the reported experiments were conducted in a small area. The drift in the information about the robot was small and therefore conveniently ignored. To apply the work reported in this thesis to mobile robots operating in large indoor environments an accurate position estimation system such as external beacons is essential.

The second goal of this research was to make a contribution to path planning for mobile robots. A new Environment Exploration Algorithm (EEA) was presented in Chapter 4. The new algorithm is based upon using quadtrees and distance transforms in a novel way. It was shown that quadtrees and distance transforms provide an effective mechanism for exploring and learning the structure of an environment with a mobile robot. The environment can be efficiently modelled with quadtrees, and distance transforms can be applied to explore paths in known and unknown portions of the environment. The problem of expensive computation of the distance transform was addressed by limiting the recomputation of the distance transform to a partial update. A new mechanism was provided for the efficient extraction of "fine" solution paths from the "coarse" chain of free space quadrants between the start and goal locations. The new mechanism allows the solution path between the start and goal locations to be fully or partially optimised. This is useful because fully optimising a path in an unknown environment could be computationally wasteful, since unexpected obstacles will necessitate replanning.

The third goal of this research was to investigate how a mobile robot path planner could be endowed with the ability to exhibit different types of path planning behaviours, other than the "optimum" path planning behaviour which only finds the shortest path to a goal. In Chapter 5 of this thesis it was shown how the EEA algorithm could be extended to exhibit different path planning behaviours. This was achieved by varying the manner in which the distance transform was generated. It was shown how the "conservative", "adventurous" and "visit all" path planning behaviours which were formulated by [Jarvis *et. al.* 86, 88] for grid based distance transforms could be incorporated to operate with the EEA. In Chapter 5 four (4) new path planning behaviours were presented, namely: the "learn all", "forgetful", "safe" and "best" path planning behaviours.

Chapter 5 showed how the EEA when operating in the "learn all" path planning behaviour was an efficient and effective mechanism for systematically mapping all the unknown regions of an environment. It was demonstrated that a "forgetful" behaviour is a useful behaviour that could coexist with other path planning behaviours in a meaningful way. Chapter 5 presented a new transform called the "path transform" which is more suitable for path planning than the distance transform. The path transform has the desirable properties of potential field path planners without suffering the penalty of local minima. It was shown how the path transform can be applied to the grid and quadtree data structures to produce the "safe" and "best" path planning behaviours, and how it can be used with the grid data structure to produce a new "visit all" path. The new "visit all" path is superior to the "visit all" path generated using ordinary distance transforms.

Chapter 6 of this research reported on a study which compared the performance of grid and quadtree distance transform algorithms. This study showed that the quadtree distance transform was a suitable mechanism for mobile robot path planning. The quadtree distance transform out performed the grid distance transform in several typical path planning situations. The grid was superior in random and spiral environments and in environments of low resolution map sizes. However such environments are less likely to occur in practice. It is more likely that a robot will need to operate in an environment with a map size of at least 128x128 cells, which contains a collection of reasonably sized obstacles. The fewer obstacles there are in an environment, the greater the superiority in performance the quadtree distance transform has over the grid distance transform. In can be stated that in general the quadtree distance transform has an inferior worst case performance compared to the grid distance transform. However the quadtree distance transform has a superior average case performance compared to the grid distance transform.

The fourth and final goal of this research was to develop a path planning algorithm for mobile robots with 3 degrees of freedom. In Chapter 7 of this thesis a new algorithm to achieve this goal was presented. This algorithm was based upon selecting two control points on the mobile robot and then constructing a path transform of the robot work space for each control point. The path transform acted as a heuristic to guide the search through the work place for a solution path in addition to controlling the orientation of the robot. It was shown that this new algorithm is computationally fast and that the algorithm takes into account the discomfort of approaching obstacles too closely, while minimising the distance to the goal.

Overall, this research has shown how effective the distance transform is for path planning in mobile robotics. A mobile robot path planner must do more than just plan the shortest path to a single goal. It has been shown that distance transforms readily support not only planning the shortest path to a goal but also a variety of other path planning options, such as path planning behaviours, multiple goals, multiple robots and robots with 3 degrees of freedom.

8.2 Further Work

A drawback of the Environment Exploration Algorithm (EEA) is that the lowest size resolution of the quadtree is considered to be the same size as the robot. This can exclude possible paths if a quadrant leaf is only partially occupied. Further investigation is warranted. Possible ways forward would be to further apply the quadtree division of space, but to have exceptions to the rule that a robot can only travel within a quadtree leaf. Another possibility is to "relax" the quadtree and move the quadrant leaf a sufficient amount until it is free space, and then attempt a path through the overlapping quadrants.

Using quadtrees to represent polygonal shaped obstacles yields a quadtree with minimum sized leaf quadrants along the edges of the polygon. The quadtree is large and the number of leaves in the tree is proportional to the polygon's perimeter. Solutions put forward by [Samet *et. al.* 85] using PM quadtrees and by [Ayala *et. al.* 85] using extended quadtrees offer substantial reductions in the memory requirements to represent polygonal obstacles. However the PM variants of the quadtree are not directly suitable for use with the path planning techniques presented in this thesis, since these representations only store the vertex points of a polygonal shape, thus making it difficult for the path planner to find a path through a quadrant containing a polygonal edge. Given the substantial memory savings using PM and extended quadtrees the question of whether or not these variants of quadtrees can be used in path planning deserves further investigation.

In this thesis the EEA has been developed and presented for 2 dimensional problems. Further work can be done to extend the EEA to handle 3 dimensional path planning problems. Adding the third dimension to the EEA could be achieved by using octrees.

An issue that deserves further investigation is the speeding up of the computation of the distance transform in both the grid and quadtree data structures, since the distance transform in both data structures essentially generates all possible paths from the goal. Concurrent processing could offer a solution to speeding up the computation of the distance transform. One approach to implementing the computation of the distance transform concurrently is to allocate a processing node to each grid or quadrant cell. Each node communicates with its neighbouring cells. The distance transform radiates out from the processing node containing the goal. An alternate approach to this problem is to use a hierarchical arrangement of processors. The processing work for computation of the distance transform is farmed out to a set of slave processors. Given the considerable computational burden of the distance transform the issue of speeding up the computation using concurrent processing deserves further attention.

The choice of grid size for the 3 DOF path planning algorithm is an important consideration. A decision must be made on the magnitude of robot motion at each step of the path planning. One strategy could be to plan a path using a coarse grid. If no solution path is found then attempt the problem on a finer grid. This is repeated until the finest grid is reached or a solution is found. Such an approach could prove to be expensive, especially when no solution path exists. Currently the grid size is chosen by selecting the smallest dimension of the rectangle which bounds the robot, and adding a small safety factor. This has proven to be satisfactory. However the issue requires further investigation.

The 3 DOF path planning algorithm is based upon coordinating the motion of two control points with a number of physical constraints. This algorithm can be regarded as a special case of coordinating two robots. There is no reason why this algorithm cannot be extended to coordinate the motion of multiple robots with 3DOF. An issue of coordinating multiple robots that requires further investigation is that of allowing the robots to operate at varying velocities so that collisions can be avoided.

Bibliography

[Adams *et. al.* 90] M.D. Adams, H. Hu and P.J. Probert, "Towards a Real-Time Architecture for Obstacle Avoidance and Path Planning in Mobile Robots", Proceedings of IEEE International Conference on Robotics and Automation, pp584-589, May 1990.

[Arkin 89] R.C. Arkin, "Motor Schema - Based Mobile Robot Navigation", International Journal of Robotics Research, Vol. 8 No.4, pp92-112, 1989.

[Ayala *et. al.* 85] D. Ayala, P. Brunet, R. Juan and I. Navazo, "Object Representation by means of Non-minimal Divsion Quadtrees and Octrees", ACM Transactions on Graphics, Vol. 4 No. 1, pp41-59, January 1985.

[Bauzil *et. al.* 81] G. Bauzil, M. Birot and R. Ribes "A Navigation Sub-System using Ultrasonic Sensors for the Mobile Robot HILARE", Proceedings of 1st Conference on Robot Vision and Sensory Control, 1981.

[Borgefors 84] G. Borgefors, "Distance Transform in Arbitary Dimensions", Computer Vision, Graphics and Image Processing, 27, pp321-345, 1984.

[Brooks 83] R.A. Brooks, "Solving the Find-Path Problem by a Good Representation of Free Space", IEEE Trans. on Systems, Man and Cybernetics, SMC-13 No.3, pp190-197, March 1983.

[Brooks 84] R.A. Brooks, "Aspects of Mobile Robot Visual Map Making", Proceedings of 2nd International Symposium of Robotics Research, pp287-293, August 1984.

[Brooks *et. al.* 85] R.A. Brooks and T. Lozano-Perez, "A Subdivision Algorithm in Configuration Space for Findpath with Rotation", IEEE Trans. on Systems, Man and Cybernetics, SMC-15 No.2, pp224-233, March/April 1985.

[Brooks 86] R.A. Brooks, "A Robust Layered Control System for a Mobile Robot", IEEE Journal of Robotics and Automation, Vol. RA-2 No. 1, pp14-23, March 1986.

[Cahn *et. al.* 75] D.F. Cahn and S.R. Phillips, "ROBNAV: A Range Based Robot Navigation and Obstacle Avoidance Algorithm", IEEE Trans. on Systems, Man and Cybernetics, SMC-55, pp138-145, September 1975.

[Chatila 82] R. Chatila, "Path Planning and Environment Learning", European Conference on Artificial Intelligence, pp211-215, July 1982.

[Chatila *et. al.* 85] R. Chatila and J-P. Laumond, "Position Referencing and Consistent World Modelling for Mobile Robots", Proceedings of IEEE International Conference on Robotics and Automation, pp138-145, March 1985.

[Chattergy 85] R. Chattergy, "Some Heuristics for the Navigation of a Robot", International Journal of Robotics Research, Vol. 4 No.1, pp59-66, 1985.

[Crowley 85] J.L. Crowley, "Navigation for an Intelligent Mobile Robot", IEEE Journal of Robotics and Automation, Vol. RA-1 No. 1, pp31-41, March 1985.

[Dijkstra 56] E.W. Dijkstra, "A Note on Two Problems in Connection with Gra Numerische Mathematik,1,pp269-271, 1959.

[Drumheller 87] M. Drumheller , "Mobile Robot Localization Using Sonar", IEEE Trans. on Pattern Analysis and Machine Learning, Vol. PAMI-9 No. 2, pp325-332, March 1987.

[Duda *et. al.* 73] R.O. Duda and P.E. Hart, "Pattern Classification and Scene Analysis", New York, Wiley, 1973.

[Donald 87] B.R. Donald, "A Search Algorithm for Motion Planning with Six Degrees of Freedom", Artifical Intelligence Vol. 31 No.3, pp295-353, 1987.

[Durrant-Whyte *et. al.* 87] H.F. Durrant-Whyte and J.J. Leonard "Navigation by Correlating Geometric Sensor Data", IEEE/RSJ International Workshop on Intelligent Robots and Systems, Tsukuba Japan, pp440-447, September 1989.

[Elfes 87] A. Elfes, "Sonar-Based Real World Mapping and Navigation", IEEE Journal of Robotics and Automation, pp249-265, June 1987.

[Flynn 85] A.M. Flynn , "Redundant Sensors for Mobile Robot Navigation", MIT Artificial Intelligence Lab., AI-TR-859, September 1985.

[Hart *et. al.* 68] P.E. Hart , N.J. Nilsson and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths", IEEE Trans. of Systems Science and Cybernetics, SSC-4 No. 2, pp 100-107, July 1968.

[Ilari *et. al.* 90] J. Ilari, C. Torras,"2D Path Planning: A Configuration Space Heuristic Approach", International Journal of Robotics Research, Vol.9 No.1, pp75-91, February 1990.

[Iyengar *et. al.* 86] S.S. Iyengar, C.C. Jorgensen, S.V.N. Rao and C.R. Weisbin,"Robot Navigation Algorithms using Learned Spatial Graphs", Robotica, Vol.4, pp93-100, 1986.

[Jarvis 83] R.A. Jarvis, "Growing Polyhedral Obstacles for Collision Free Paths", Australian Computer Journal, Vol. 15 No. 3, pp103-111, August 1983.

[Jarvis *et. al.* 86] R.A. Jarvis and J.C. Byrne, "Robot Navigation: Touching, Seeing and Knowing", Proceedings of 1st Australian Conference on Artificial Intelligence, November 1986.

[Jarvis *et. al.* 88] R.A. Jarvis, J.C. Byrne and K. Ajay, "An Intelligent Autonomous Guided Vehicle: Localisation, Environment Modelling and Collision-Free Path Finding", Proceedings of 19th-ISIR The International Symposium and Exposition on Robots, pp767-792 November 1988.

[Kambhampati *et. al.* 86] S. Kambhampati and L.S. Davis, "Multiresolution Path Planning for Mobile Robot", IEEE Journal of Robotics and Automation, Vol. RA-2 No. 3, pp135-145, September 1986.

[Keirsey *et. al.* 84] D.M. Keirsey, E. Koch, J. McKisson et. al., "An Algorithm of Navigation for a Mobile Robot", Proceedings of 1984 IEEE International Conference on Robotics, pp 574-583, May 1984.

[Khatib 86] O. Khatib, "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots", International Journal of Robotics Research, Vol. 5 No.1, pp90-98, 1986.

[Khosla *et. al.* 88] P. Khosla amd R. Volpe, "Superquadric Artificial Potential for Obactacle Avoidance and Approach", Proceedings of IEEE International Conference on Robotics and Automation, pp1778-1784, May 1988.

[Krogh 84] B.H. Krogh, "A Generalised Potential Field Approach to Obstacle Avoidance Control", First World Conference on Robotics Research, August 1984.

[Krogh *et. al.* 86] B.H. Krogh and C.E. Thorpe, "Integrated Path Planning and Dynamic Steering Control for Autonomous Vehicles", Proceedings of IEEE International Conference on Robotics and Automation, pp1664-1669, April 1986.

[Kuan *et. al.* 85] D.T. Kuan, J.C. Zamiska and R.A. Brooks, "Natural Decomposition of Free Space for Path Planning", Proceedings of IEEE International Conference on Robotics and Automation, pp168-173, March 1985.

[Lozano-Perez *et. al.* 79] T. Lozano-Perez and M.A. Wesley, "An Algorithm for Planning Collision Free Paths among Polyhedral Obstacles", Communications of the ACM, Vol. 22 No. 10, pp560-570, October 1979.

[Lozano-Perez 83] T. Lozano-Perez, "Spatial Planning: A Configuration Space Approach", IEEE Trans. on Computers, C-32 No. 2, pp108-120, February 1983.

[Lumelsky 87] V.J. Lumelsky and A. A. Stepanov, "Path Planning Strategies for a Point Mobile Automaton Moving Amidst Unknown Obstacles of Arbitrary Shape", Algorithmica, Vol. 2 No. 4, pp403-430, 1987.

[Lumelsky 89] V.J. Lumelsky, S. Mukhopadhyay and K. Sun, "Sensor-Based Terrain Acquisition: a 'Seed Spreader' Strategy", IEEE/RSJ International Workshop on Intelligent Robots and Systems, Tsukuba Japan, pp62-67, September 1989.

[Miller 84] D.P. Miller, "Two Dimensional Mobile Robot Positioning using onboard Sonar", Proceedings of 9th William T. Pecora Memorial Remote Sensing Symposium, IEEE, USGS, NASA, ASP, pp362-369, October 1984.

[Miller 85] D.P. Miller, "Planning by Search Through Simulations", Phd dissertation, Yale University, Department of Computer Science, October 1985.

[Moravec 80] H.P. Moravec, "Obstacle Avoidance and Navigation in the Real World by a Seeing Rover", Phd dissertation, Stanford University, September 1980.

[Noborio *et. al.* 88] H. Noborio, T. Naniwa and S. Arimoto, "A Fast Path-Planning Algorithm by Synchronising Modification and Search of its Path Graph Representation", Proceedings of IEEE International Workshop on Artificial Intelligence for Industrial Applications, pp351-357, 1988.

[Noborio *et. al.* 89] H. Noborio, T. Naniwa and S. Arimoto, "A Feasible Motion-Planning Algorithm for a Mobile robot on a Quadtree Representation", Proceedings of IEEE International Conference on Robotics and Automation, pp327-332, May 1989.

[Polaroid 82] Polaroid Corporation, "Ultrasonic Range Finders", 1982.

[Rao *et. al.* 86] S.V.N. Rao, S.S. Iyengar, C.C. Jorgensen and C.R. Weisbin, "Robot Navigation in an Unexplored Terrain", Journal of Robotic Systems, Vol.3 No.4, pp389-407, 1986.

[Rimon *et. al.* 88] E. Rimon and D.E. Koditschek, "Exact Robot Navigation using Cost Functions", Proceedings of IEEE International Conference on Robotics and Automation, pp1791-1796, May 1988.

[Samet 81] H. Samet, "An Agorithm for Converting Rasters to Quadtrees", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-3 No. 1, pp93-95, 1981.

[Samet 82] H. Samet, "Neighbour Finding Techniques for Images Represented by Quadtrees", Computer Graphics and Image Processing, Vol. 18 No. 3, pp37-57, 1982.

[Samet 84] H. Samet, "The Quadtree and Related Heirarchical Data Structures", ACM Computing Surveys, Vol. 16 No. 2, pp187-260, June 1984.

[Samet *et. al.* 85] H. Samet and R.E. Webber, "Storing a Collection of Polygons using Quadtrees", ACM Transactions on Graphics, Vol. 4 No. 3, pp182-222, July 1985.

[Samet 88] H. Samet, "An Overview of Quadtrees, Octrees and Related Hierarchical Data Structures", NATO ASI Series, Vol. F40, Theoretical Foundations of Computer Graphics, pp51-68, Springer-Verlag Berlin Heidelberg, 1988.

[Suh *et. al.* 88] S.H. Suh and K.G. Shin, "A Variational Dynamic Programming Approach to Robot Path Planning with a Distance Safety Criterion", IEEE Journal of Robotics and Automation, Vol. RA-4 No. 3, pp334-349, September 1988.

[Schwartz *et. al.* 83] J.T. Schwartz and M. Sharir, "On the Piano Movers Problem, the Case of a Two Dimensional Rigid Polygonal Body Moving Amidst Polygonal Barriers",Communications on Pure and Applied Mathematics, Vol. 36, pp345-398, 1983.

[Tarjan 81] R. Tarjan, "Fast Algorithms for Solving Path Problems", Journal of ACM, Vol. 28 No. 3, July 1981.

[Thompson 77] A.M. Thompson, "The Navigation System of the JPL Robot", Proceedings of 5th International Joint Conference on Artificial Intelligence, pp749-757, August 1977.

[Thorpe 84] C.E. Thorpe, "FIDO:Vision and Navigation for a Robot Rover", Phd dissertation, Carnegie Mellon University, Department of Computer Science, December 1984.

[Warren 89] C.W. Warren, "Global Path Planning using Artificial Potential Fields", Proceedings of IEEE International Conference on Robotics and Automation, pp316-321, May 1989.

Appendix A
Fine Path Planning

This appendix presents the experimental results for determining the amount of computational effort that is required to plan "fine" execution paths in quadtrees and grids. Results are presented which show the proportion of "fine" path planning in the total path planning task.

Four experimental results are presented in Tables A.1 - A.4. The layouts of the environments which were used for the experiments are shown in Figure A.1. For each environment layout data was collected from seven (7) different resolution size maps ranging from 8 x 8 to 512 x 512 pixels. Each table contains the following information:

* time to compute the distance transform for grids which is shown as GRID DT.
* time to compute the distance transform for quadtrees which is shown as QUADTREE DT.
* time to compute the fine path for grids which is shown as GRID FINE PATH.
* time to compute the fine path for quadtrees which is shown as QUADTREE FINE PATH.
* time to construct the quadtree which is shown as QUADTREE BUILD.
* total time for path planning using grids which is GRID DT + GRID FINE PATH and is shown as TOTAL GRID.
* total time for path planning using quadtrees which is QUADTREE BUILD + QUADTREE DT + QUADTREE FINE PATH and is shown as TOTAL QUADTREE.
* percentage proportion of planning fine paths in grids in the total path planning task which is shown as % GRID FP.
* percentage proportion of planning fine paths in quadtrees in the total path planning task which is shown as % QUADTREE FP.

From the results shown in Tables A.1 - A.4 it can be concluded that fine path planning is an insignificant percentage of the overall path planning task. The proportion of time spent computing the fine path decreases as the map resolution size grows. The cost of computing fine paths for grids is an order of magnitude less than the cost of computing fine paths for quadtrees.

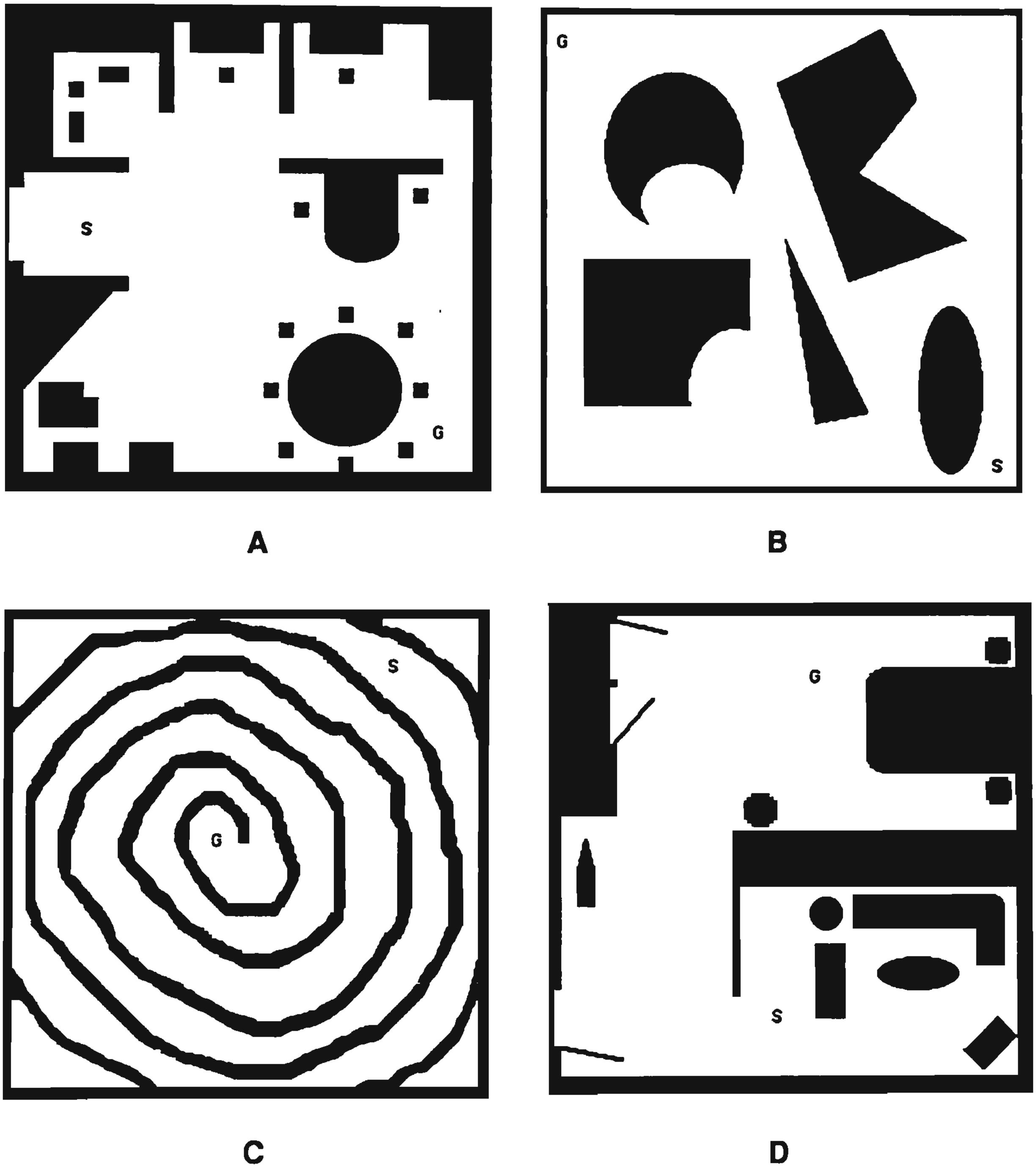


Figure A.1

The four path planning environments used for experimentation.

Map Size	Grid DT	Quadtree DT	Grid Fine Path	Quadtree Fine Path	Quadtree Build	Total Grid	Total Quadtree	% Grid FP	% Quadtree FP
8 × 8	1	2	0	0	0	1	2	–	–
16 × 16	15	70	0	1	3	15	74	–	–
32 × 32	82	460	0	7	10	82	477	–	1.467
64 × 64	436	459	1	14	29	437	502	0.229	2.788
128 × 128	1744	460	2	14	112	1746	586	0.115	2.389
256 × 256	6976	459	7	15	412	6983	886	0.100	1.693
512 × 512	27904	460	13	14	1539	27917	2013	0.046	0.695

Table A.1

The path planning statistics for the environment shown in Figure A.1 (A)

Map Size	Grid DT	Quadtree DT	Grid Fine Path	Quadtree Fine Path	Quadtree Build	Total Grid	Total Quadtree	% Grid FP	% Quadtree FP
8 × 8	0	1	0	0	1	0	2	–	–
16 × 16	10	35	0	0	3	10	38	–	–
32 × 32	64	261	0	8	10	64	279	–	2.867
64 × 64	301	1492	1	17	29	302	1538	0.331	1.105
128 × 128	1250	2832	2	38	127	1252	2997	0.160	1.268
256 × 256	5100	5456	6	86	445	5106	5988	0.118	1.436
512 × 512	20801	10356	12	141	1566	20813	12063	0.058	1.116

Table A.2

The path planning statistics for the environment shown in Figure A.1 (B)

Map Size	Grid DT	Quadtree DT	Grid Fine Path	Quadtree Fine Path	Quadtree Build	Total Grid	Total Quadtree	% Grid FP	% Quadtree FP
8 × 8	0	0	0	0	1	0	1	–	–
16 × 16	3	4	0	0	2	3	6	–	–
32 × 32	30	97	0	2	11	30	110	–	1.818
64 × 64	609	789	0	11	52	609	852	–	1.291
128 × 128	2722	24188	8	50	154	2730	24392	0.293	0.205
256 × 256	11594	59286	12	108	518	11606	59912	0.103	0.180
512 × 512	49383	145070	17	194	1732	49400	146996	0.030	0.132

Table A.3

The path planning statistics for the environment shown in Figure A.1 (C)

Map Size	Grid DT	Quadtree DT	Grid Fine Path	Quadtree Fine Path	Quadtree Build	Total Grid	Total Quadtree	% Grid FP	% Quadtree FP
8 × 8	2	8	0	0	0	2	8	–	–
16 × 16	13	65	0	0	2	13	67	–	–
32 × 32	92	295	0	7	9	92	396	–	1.768
64 × 64	399	601	2	14	31	401	646	0.499	2.167
128 × 128	1698	2695	3	47	124	1701	2866	0.176	1.640
256 × 256	6776	5390	5	91	432	6781	5913	0.074	1.539
512 × 512	27168	10880	17	167	1506	27185	12553	0.063	1.330

Table A.4

The path planning statistics for the environment shown in Figure A.1 (D)

Appendix B
Random Data

This appendix presents the experimental results for determining the amount of computational effort and computer memory that is required to produce the distance transform in grids and quadtrees in environments which have been generated randomly. The distance transform is computed for quadtrees with and without neighbour lists.

The Tables B.1 - B.7 presented in this appendix show the times that are required to compute the following tasks for the seven (7) concentrations of random data:

* distance transform for grids, which is shown as GRID DT.
* build the quadtree, which is shown as BUILD QT.
* distance transform for quadtrees, which is shown as QT DT.
* total distance transform for quadtrees , which is BUILD QT + QT DT and is shown as QT DT TOTAL.
* build the quadtree neighbour list, which is shown as BUILD QT NBR.
* distance transform for quadtrees with neighbour lists, which is shown as QT NBR DT.
* total distance transform for quadtrees with neighbour lists, which is BUILD QT + BUILD QT NBR + QT NBR DT and is shown as QT DT NBR TOTAL.

Tables B.1 - B.7 also show the memory requirements in bytes to support the computation of the distance transform for the following data structures:

* memory for grids, which is shown as GRID MEMORY.
* memory for quadtrees, which is shown as QT MEMORY.
* memory for quadtrees with neighbour lists, which is shown as QT NBR MEMORY.

This experiment was conducted with five (5) different sets of random data for each map size with a particular concentration of blocked cells. This is to ensure that map configurations which produce unreachable goals do not distort the statistics. Tables B.1 - B.7 show the maximum, minimum and average computation times and memory requirements for the five (5) random data sets. Figures B.1 - 2 show an example of a typical random data set for an 8x8 map. Figures B.1(A) - (B) show random environments with 0% and 5% concentrations. Figures B.2(A) - (F) show random environments with 10%, 20%, 30%, 40%, 50% and 60% concentrations. Figures B.3 - B.6 show in graphical form the average computation times for the experimental data presented in Tables B.1 - B.7. Figures B.7 - B.10 show in graphical form the average memory requirements for the experimental data presented in Tables B.1 - B.7.

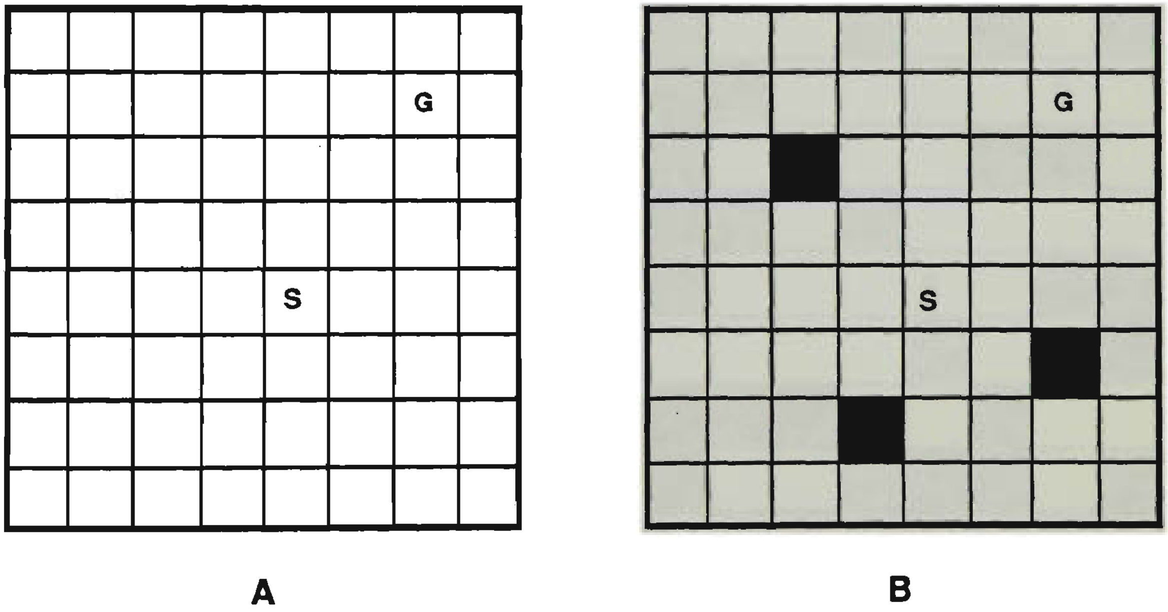


Figure B.1

Random Environments 0% and 5% .

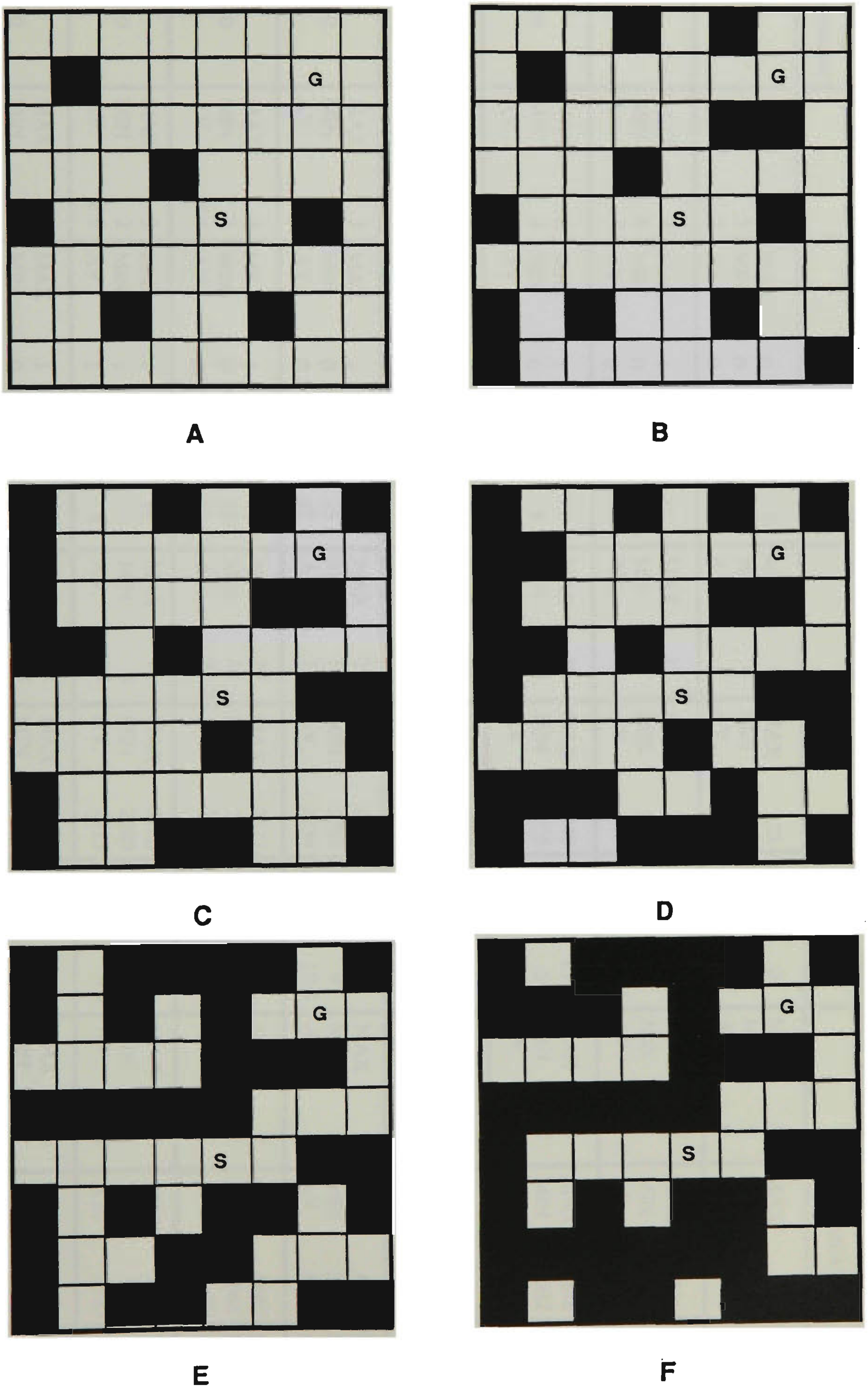


Figure B.2

Random environments 10%, 20%, 30%, 40%, 50% and 60%.

SIZE: 8 × 8
Grid Memory 256

RANDOM MAP RESULTS

Percentage Concentration	GRID DT	BUILD DT	QT DT	QT DT TOTAL	QT MEMORY	QT DT NBR	BUILD QT NBR	QT NBR MEMORY	QT DT NBR TOTAL
0	MAX 5 MIN 5 AV. 5	MAX 0 MIN 0 AV. 0	MAX 0 MIN 0 AV. 0	MAX 0 MIN 0 AV. 0	MAX 32 MIN 32 AV. 32	MAX 0 MIN 0 AV. 0	MAX 0 MIN 0 AV. 0	MAX 48 MIN 48 AV. 48	MAX 0 MIN 0 AV. 0
5	MAX 4 MIN 6 AV. 5	MAX 1 MIN 0 AV. 0.6	MAX 12 MIN 7 AV. 9.4	MAX 13 MIN 7 AV. 10	MAX 928 MIN 672 AV. 774	MAX 9 MIN 5 AV. 6.4	MAX 2 MIN 1 AV. 1.2	MAX 1886 MIN 1296 AV. 1510	MAX 11 MIN 6 AV. 8.2
10	MAX 6 MIN 4 AV. 5	MAX 1 MIN 0 AV. 0.6	MAX 19 MIN 8 AV. 13.6	MAX 19 MIN 8 AV. 14.2	MAX 1440 MIN 1056 AV. 1286	MAX 13 MIN 5 AV. 8.8	MAX 3 MIN 2 AV. 2.4	MAX 2848 MIN 2032 AV. 2502	MAX 15 MIN 7 AV. 11.8
20	MAX 5 MIN 3 AV. 4	MAX 2 MIN 0 AV. 1	MAX 20 MIN 7 AV. 15.4	MAX 23 MIN 8 AV. 16.4	MAX 1952 MIN 1568 AV. 1824	MAX 13 MIN 4 AV. 9.8	MAX 3 MIN 1 AV. 2	MAX 3680 MIN 2832 AV. 3382	MAX 17 MIN 6 AV. 12.8
30	MAX 5 MIN 3 AV. 3.8	MAX 1 MIN 0 AV. 0.8	MAX 21 MIN 15 AV. 17	MAX 22 MIN 16 AV. 17.8	MAX 2336 MIN 1952 AV. 2234	MAX 13 MIN 9 AV. 10.8	MAX 2 MIN 2 AV. 2	MAX 4224 MIN 3648 AV. 4045	MAX 16 MIN 11 AV. 13.6
40	MAX 4 MIN 3 AV. 3.4	MAX 1 MIN 0 AV. 0.8	MAX 28 MIN 13 AV. 19.4	MAX 29 MIN 14 AV. 20.2	MAX 2572 MIN 2336 AV. 2460	MAX 17 MIN 9 AV. 11.4	MAX 2 MIN 2 AV. 2	MAX 4480 MIN 4064 AV. 4243	MAX 20 MIN 10 AV. 14.4
50	MAX 3 MIN 2 AV. 2.6	MAX 1 MIN 1 AV. 1	MAX 10 MIN 7 AV. 8	MAX 11 MIN 8 AV. 9	MAX 2720 MIN 2080 AV. 2413	MAX 5 MIN 4 AV. 4.4	MAX 2 MIN 2 AV. 2	MAX 4432 MIN 3504 AV. 3990	MAX 8 MIN 7 AV. 7.6
60	MAX 4 MIN 2 AV. 2.6	MAX 1 MIN 0 AV. 0.6	MAX 14 MIN 3 AV. 8.4	MAX 14 MIN 4 AV. 9	MAX 2592 MIN 2336 AV. 2490	MAX 7 MIN 2 AV. 4.2	MAX 2 MIN 1 AV. 1.2	MAX 4128 MIN 3808 AV. 3997	MAX 8 MIN 4 AV. 6

Table B.1
Path Planning Statistics for 8 x 8 map

SIZE: 16 × 16
Grid Memory 1024

RANDOM MAP RESULTS

Percentage Concentration	GRID DT	BUILD DT	QT DT	QT DT TOTAL	QT MEMORY	QT DT NBR	BUILD QT NBR	QT NBR MEMORY	QT DT NBR TOTAL
0	MAX 14	MAX 1	MAX 0	MAX 1	MAX 32	MAX 0	MAX 0	MAX 48	MAX 1
	MIN 14	MIN 1	MIN 0	MIN 1	MIN 32	MIN 0	MIN 0	MIN 48	MIN 1
	AV. 14	AV. 1	AV. 0	AV. 1	AV. 32	AV. 0	AV. 0	AV. 48	AV. 1
5	MAX 21	MAX 4	MAX 53	MAX 55	MAX 3488	MAX 38	MAX 3	MAX 7264	MAX 44
	MIN 14	MIN 1	MIN 26	MIN 28	MIN 2592	MIN 18	MIN 6	MIN 5216	MIN 25
	AV. 19.6	AV. 2.2	AV. 40.4	AV. 42.6	AV. 3976	AV. 28.6	AV. 4.6	AV. 6579	AV. 35
10	MAX 27	MAX 5	MAX 113	MAX 115	MAX 5408	MAX 77	MAX 7	MAX 10864	MAX 89
	MIN 20	MIN 2	MIN 66	MIN 68	MIN 4896	MIN 44	MIN 7	MIN 9712	MIN 53
	AV. 21.4	AV. 2.6	AV. 94.4	AV. 97	AV. 5203	AV. 64.2	AV. 7	AV. 10419	AV. 74
20	MAX 25	MAX 3	MAX 145	MAX 147	MAX 8096	MAX 94	MAX 11	MAX 15456	MAX 107
	MIN 18	MIN 2	MIN 64	MIN 67	MIN 7072	MIN 42	MIN 8	MIN 13232	MIN 54
	AV. 22.4	AV. 2.4	AV. 107.8	AV. 110.2	AV. 7533	AV. 70	AV. 9.4	AV. 14234	AV. 81.8
30	MAX 28	MAX 5	MAX 123	MAX 128	MAX 9120	MAX 76	MAX 11	MAX 16480	MAX 90
	MIN 22	MIN 4	MIN 88	MIN 93	MIN 8608	MIN 55	MIN 9	MIN 15536	MIN 71
	AV. 23.2	AV. 4.6	AV. 108	AV. 112.6	AV. 8915	AV. 67.6	AV. 10	AV. 16163	AV. 82.2
40	MAX 31	MAX 6	MAX 143	MAX 149	MAX 10016	MAX 78	MAX 14	MAX 17376	MAX 98
	MIN 24	MIN 3	MIN 74	MIN 78	MIN 9376	MIN 41	MIN 11	MIN 16192	MIN 58
	AV. 24.4	AV. 4	AV. 103.8	AV. 107.8	AV. 9632	AV. 57.4	AV. 12.8	AV. 16663	AV. 74.2
50	MAX 27	MAX 7	MAX 82	MAX 89	MAX 10400	MAX 43	MAX 13	MAX 17344	MAX 62
	MIN 16	MIN 3	MIN 42	MIN 46	MIN 9632	MIN 22	MIN 10	MIN 16032	MIN 39
	AV. 22.4	AV. 4.4	AV. 67	AV. 71.4	AV. 9914	AV. 34.8	AV. 11.4	AV. 16763	AV. 50.6
60	MAX 18	MAX 6	MAX 57	MAX 63	MAX 9888	MAX 29	MAX 10	MAX 15936	MAX 44
	MIN 9	MIN 4	MIN 30	MIN 36	MIN 9120	MIN 15	MIN 9	MIN 14992	MIN 30
	AV. 14	AV. 5.6	AV. 44.6	AV. 50.2	AV. 9478	AV. 21.6	AV. 9.4	AV. 15478	AV. 36.6

Table B.2

Path Planning Statistics for 16 x 16 map

SIZE: 32 × 32
Grid Memory 4096

RANDOM MAP RESULTS

Percentage Concentration	GRID DT	BUILD DT	QT DT	QT DT TOTAL	QT MEMORY	QT DT NBR	BUILD QT NBR	QT NBR MEMORY	QT DT NBR TOTAL
0	MAX 75	MAX 7	MAX 0	MAX 7	MAX 32	MAX 0	MAX 0	MAX 48	MAX 7
	MIN 75	MIN 7	MIN 0	MIN 7	MIN 32	MIN 0	MIN 0	MIN 48	MIN 7
	AV. 75	AV. 7	AV. 0	AV. 7	AV. 32	AV. 0	AV. 0	AV. 48	AV. 7
5	MAX 143	MAX 13	MAX 389	MAX 398	MAX 13472	MAX 254	MAX 29	MAX 28464	MAX 291
	MIN 107	MIN 8	MIN 254	MIN 267	MIN 12704	MIN 167	MIN 27	MIN 26944	MIN 201
	AV. 128.8	AV. 10.4	AV. 320.4	AV. 330.8	AV. 13062	AV. 212.2	AV. 27.6	AV. 27730	AV. 249
10	MAX 170	MAX 14	MAX 839	MAX 854	MAX 21152	MAX 529	MAX 43	MAX 42320	MAX 587
	MIN 102	MIN 10	MIN 359	MIN 369	MIN 19872	MIN 228	MIN 40	MIN 39984	MIN 268
	AV. 123.6	AV. 12.2	AV. 533.2	AV. 545.4	AV. 20589	AV. 337.4	AV. 41.6	AV. 41076	AV. 389.2
20	MAX 185	MAX 15	MAX 902	MAX 917	MAX 30240	MAX 539	MAX 55	MAX 57744	MAX 566
	MIN 124	MIN 12	MIN 632	MIN 662	MIN 28704	MIN 380	MIN 51	MIN 54496	MIN 447
	AV. 154.4	AV. 14.4	AV. 739.2	AV. 753.6	AV. 29421	AV. 441	AV. 53	AV. 56028	AV. 489
30	MAX 167	MAX 16	MAX 980	MAX 995	MAX 36000	MAX 551	MAX 59	MAX 65520	MAX 857
	MIN 111	MIN 12	MIN 675	MIN 690	MIN 34720	MIN 382	MIN 55	MIN 62704	MIN 456
	AV. 133.2	AV. 14.8	AV. 803.6	AV. 818.4	AV. 35386	AV. 452.8	AV. 57	AV. 64150	AV. 587.4
40	MAX 172	MAX 20	MAX 1067	MAX 1083	MAX 39968	MAX 570	MAX 59	MAX 69408	MAX 642
	MIN 122	MIN 16	MIN 202	MIN 214	MIN 37792	MIN 107	MIN 55	MIN 68144	MIN 182
	AV. 146.8	AV. 17.2	AV. 621.6	AV. 639	AV. 39072	AV. 330	AV. 57.2	AV. 68606	AV. 404.6
50	MAX 128	MAX 16	MAX 756	MAX 772	MAX 40096	MAX 369	MAX 53	MAX 67152	MAX 434
	MIN 42	MIN 16	MIN 151	MIN 167	MIN 38432	MIN 73	MIN 47	MIN 63808	MIN 138
	AV. 93.2	AV. 16	AV. 345.6	AV. 361.6	AV. 39302	AV. 169.6	AV. 50	AV. 65744	AV. 235.6
60	MAX 143	MAX 17	MAX 369	MAX 385	MAX 39762	MAX 167	MAX 41	MAX 63824	MAX 222
	MIN 35	MIN 16	MIN 116	MIN 132	MIN 37562	MIN 51	MIN 38	MIN 60272	MIN 105
	AV. 74.6	AV. 16.4	AV. 185	AV. 201.4	AV. 38334	AV. 83.2	AV. 39.4	AV. 61869	AV. 139

Table B.3
Path Planning Statistics for 32 x 32 map

SIZE: 64 × 64
Grid Memory 16384

RANDOM MAP RESULTS

Percentage Concentration	GRID DT		BUILD DT		QT DT		QT DT TOTAL		QT MEMORY		QT DT NBR		BUILD QT NBR		QT NBR MEMORY		QT DT NBR TOTAL	
0	MAX	305	MAX	25	MAX	0	MAX	25	MAX	32	MAX	0	MAX	0	MAX	48	MAX	25
	MIN	305	MIN	25	MIN	0	MIN	25	MIN	32	MIN	0	MIN	0	MIN	48	MIN	25
	AV.	305	AV.	25	AV.	0	AV.	25	AV.	32	AV.	0	AV.	0	AV.	48	AV.	25
5	MAX	719	MAX	40	MAX	3625	MAX	3665	MAX	53920	MAX	2354	MAX	122	MAX	114640	MAX	2495
	MIN	576	MIN	36	MIN	1821	MIN	1857	MIN	51616	MIN	1192	MIN	115	MIN	109136	MIN	2237
	AV.	607	AV.	38	AV.	3091	AV.	3129	AV.	52947	AV.	2021	AV.	118	AV.	112355	AV.	2375
10	MAX	824	MAX	49	MAX	4532	MAX	4577	MAX	83744	MAX	2879	MAX	194	MAX	170608	MAX	3106
	MIN	548	MIN	45	MIN	2233	MIN	2278	MIN	81440	MIN	1416	MIN	182	MIN	165824	MIN	1644
	AV.	658	AV.	45	AV.	3333	AV.	3379	AV.	82464	AV.	2113	AV.	187	AV.	168032	AV.	2345
20	MAX	1119	MAX	56	MAX	8777	MAX	8832	MAX	122016	MAX	5289	MAX	234	MAX	234144	MAX	5577
	MIN	744	MIN	55	MIN	4525	MIN	4581	MIN	119072	MIN	2729	MIN	223	MIN	228304	MIN	3008
	AV.	917	AV.	55	AV.	6274	AV.	6329	AV.	120454	AV.	3783	AV.	229	AV.	231030	AV.	4067
30	MAX	1005	MAX	61	MAX	8983	MAX	9043	MAX	143392	MAX	5133	MAX	249	MAX	261632	MAX	5424
	MIN	778	MIN	60	MIN	6268	MIN	6328	MIN	141216	MIN	3582	MIN	231	MIN	257264	MIN	3880
	AV.	866	AV.	60	AV.	7081	AV.	7141	AV.	142445	AV.	4164	AV.	240	AV.	259504	AV.	4465
40	MAX	982	MAX	68	MAX	4818	MAX	4881	MAX	157856	MAX	2597	MAX	230	MAX	275296	MAX	2885
	MIN	290	MIN	63	MIN	1614	MIN	1677	MIN	153888	MIN	868	MIN	225	MIN	268192	MIN	1166
	AV.	681	AV.	64	AV.	2865	AV.	2929	AV.	155321	AV.	1433	AV.	227	AV.	270578	AV.	1724
50	MAX	1282	MAX	68	MAX	2362	MAX	2430	MAX	160032	MAX	1191	MAX	204	MAX	268416	MAX	1463
	MIN	128	MIN	64	MIN	665	MIN	743	MIN	156576	MIN	335	MIN	197	MIN	263008	MIN	599
	AV.	512	AV.	66	AV.	1346	AV.	1412	AV.	158112	AV.	680	AV.	201	AV.	265286	AV.	946
60	MAX	574	MAX	64	MAX	1053	MAX	1117	MAX	156064	MAX	495	MAX	166	MAX	252720	MAX	724
	MIN	110	MIN	63	MIN	522	MIN	585	MIN	152096	MIN	245	MIN	165	MIN	247168	MIN	473
	AV.	231	AV.	64	AV.	833	AV.	897	AV.	154195	AV.	889	AV.	165	AV.	250291	AV.	618

Table B.4
Path Planning Statistics for 64 x 64 map

SIZE: 128 × 128
Grid Memory 65536

RANDOM MAP RESULTS

Percentage Concentration	GRID	DT	BUILD	DT	QT	DT	QT DT TOTAL		QT MEMORY		QT DT NBR		BUILD QT NBR		QT NBR MEMORY		QT DT NBR TOTAL	
0	MAX	1221	MAX	106	MAX	0	MAX	106	MAX	32	MAX	0	MAX	0	MAX	48	MAX	106
	MIN	1221	MIN	106	MIN	0	MIN	106	MIN	32	MIN	0	MIN	0	MIN	48	MIN	106
	AV.	1221	AV.	106	AV.	0	AV.	106	AV.	32	AV.	0	AV.	0	AV.	48	AV.	106
5	MAX	2875	MAX	160	MAX	18404	MAX	18564	MAX	214432	MAX	11947	MAX	514	MAX	457968	MAX	12581
	MIN	2315	MIN	159	MIN	15616	MIN	15776	MIN	209696	MIN	10143	MIN	474	MIN	446624	MIN	10629
	AV.	2688	AV.	1160	AV.	15928	AV.	17088	AV.	212939	AV.	10980	AV.	491	AV.	454128	AV.	11578
10	MAX	6593	MAX	188	MAX	39092	MAX	39279	MAX	333984	MAX	24536	MAX	722	MAX	684194	MAX	25445
	MIN	3290	MIN	185	MIN	21406	MIN	21594	MIN	328608	MIN	13427	MIN	703	MIN	671888	MIN	14318
	AV.	4569	AV.	187	AV.	32103	AV.	32289	AV.	331808	AV.	20144	AV.	715	AV.	679222	AV.	21045
20	MAX	4394	MAX	175	MAX	36503	MAX	36676	MAX	483104	MAX	23078	MAX	691	MAX	929584	MAX	23941
	MIN	3902	MIN	173	MIN	15195	MIN	15380	MIN	471456	MIN	9546	MIN	688	MIN	904592	MIN	10414
	AV.	4186	AV.	174	AV.	29365	AV.	29542	AV.	478923	AV.	18539	AV.	690	AV.	920555	AV.	19404
30	MAX	7040	MAX	230	MAX	42457	MAX	42652	MAX	572960	MAX	23711	MAX	971	MAX	1046432	MAX	24912
	MIN	4881	MIN	195	MIN	33868	MIN	35283	MIN	566816	MIN	20112	MIN	706	MIN	1033792	MIN	21013
	AV.	6082	AV.	207	AV.	37126	AV.	37333	AV.	569973	AV.	21553	AV.	796	AV.	1040112	AV.	22827
40	MAX	5792	MAX	293	MAX	58426	MAX	58719	MAX	616736	MAX	30532	MAX	908	MAX	1074880	MAX	31733
	MIN	3225	MIN	215	MIN	16844	MIN	17065	MIN	613920	MIN	9406	MIN	669	MIN	1072976	MIN	10296
	AV.	4584	AV.	243	AV.	35481	AV.	35724	AV.	615371	AV.	19151	AV.	750	AV.	1072821	AV.	20144
50	MAX	1277	MAX	298	MAX	6860	MAX	7158	MAX	636320	MAX	3355	MAX	814	MAX	1069136	MAX	4467
	MIN	809	MIN	224	MIN	4323	MIN	4547	MIN	633504	MIN	2258	MIN	598	MIN	1062320	MIN	3080
	AV.	1120	AV.	273	AV.	5985	AV.	6258	AV.	634605	AV.	2973	AV.	739	AV.	1064997	AV.	3985
60	MAX	659	MAX	295	MAX	3190	MAX	3485	MAX	618784	MAX	1196	MAX	674	MAX	1003568	MAX	1197
	MIN	440	MIN	218	MIN	2103	MIN	2394	MIN	618016	MIN	228	MIN	488	MIN	953840	MIN	1902
	AV.	537	AV.	268	AV.	2585	AV.	2853	AV.	618315	AV.	790	AV.	610	AV.	986821	AV.	1668

Table B.5
Path Planning Statistics for 128 x 128 map

SIZE: 256 × 256
Grid Memory 262144

RANDOM MAP RESULTS

Percentage Concentration	GRID DT	BUILD DT	QT DT	QT DT TOTAL	QT MEMORY	QT DT NBR	BUILD QT NBR	QT NBR MEMORY	QT DT NBR TOTAL
0	MAX 4884	MAX 257	MAX 0	MAX 257	MAX 32	MAX 0	MAX 0	MAX 48	MAX 257
	MIN 4884	MIN 257	MIN 0	MIN 257	MIN 32	MIN 0	MIN 0	MIN 48	MIN 257
	AV. 4884	AV. 257	AV. 0	AV. 257	AV. 32	AV. 0	AV. 0	AV. 48	AV. 257
5	MAX 18594	MAX 432	MAX 66329	MAX 66759	MAX 661536	MAX 44667	MAX 1081	MAX 1345616	MAX 45766
	MIN 16553	MIN 429	MIN 44233	MIN 44662	MIN 659232	MIN 29525	MIN 1077	MIN 1341680	MIN 31034
	AV. 17902	AV. 430	AV. 58182	AV. 58612	AV. 660128	AV. 38816	AV. 1079	AV. 1343351	AV. 40326
10	MAX 35063	MAX 494	MAX 123025	MAX 123505	MAX 966304	MAX 79193	MAX 1444	MAX 1859552	MAX 81117
	MIN 27478	MIN 480	MIN 97370	MIN 97859	MIN 960800	MIN 62228	MIN 1436	MIN 1854896	MIN 64360
	AV. 30323	AV. 488	AV. 105949	AV. 106437	AV. 963147	AV. 67952	AV. 1439	AV. 1856992	AV. 69879
20	MAX 56421	MAX 545	MAX 394947	MAX 395490	MAX 1243168	MAX 263298	MAX 2454	MAX 2364192	MAX 265380
	MIN 38285	MIN 542	MIN 267995	MIN 268537	MIN 1229088	MIN 178633	MIN 1539	MIN 2352528	MIN 181280
	AV. 44749	AV. 543	AV. 313291	AV. 313784	AV. 1234933	AV. 208824	AV. 2033	AV. 2317317	AV. 211400
30	MAX 48050	MAX 545	MAX 405078	MAX 405623	MAX 1240096	MAX 226224	MAX 1733	MAX 261392	MAX 228542
	MIN 39869	MIN 544	MIN 336093	MIN 336638	MIN 123532	MIN 187698	MIN 1255	MIN 2116272	MIN 189517
	AV. 42794	AV. 545	AV. 360766	AV. 361310	AV. 1237664	AV. 201477	AV. 1434	AV. 2167904	AV. 203456
40	MAX 42425	MAX 634	MAX 345460	MAX 346094	MAX 1338940	MAX 186463	MAX 1855	MAX 2332507	MAX 188753
	MIN 37421	MIN 585	MIN 299368	MIN 299958	MIN 1333688	MIN 161586	MIN 1656	MIN 2182824	MIN 164031
	AV. 39641	AV. 603	AV. 319148	AV. 319751	AV. 1336314	AV. 172262	AV. 1734	AV. 2236079	AV. 174598
50	MAX 6212	MAX 621	MAX 33265	MAX 33886	MAX 1381300	MAX 16269	MAX 1715	MAX 2315496	MAX 18605
	MIN 4112	MIN 594	MIN 22019	MIN 22626	MIN 1375882	MIN 10769	MIN 1482	MIN 2166905	MIN 12852
	AV. 5412	AV. 605	AV. 28981	AV. 29588	AV. 1378591	AV. 14174	AV. 1584	AV. 2221187	AV. 16363
60	MAX 2224	MAX 630	MAX 11817	MAX 12401	MAX 1345537	MAX 5022	MAX 1488	MAX 2136172	MAX 6920
	MIN 1098	MIN 584	MIN 5860	MIN 6490	MIN 1340254	MIN 2815	MIN 1260	MIN 2007005	MIN 4380
	AV. 1521	AV. 608	AV. 8100	AV. 8704	AV. 1342893	AV. 3442	AV. 1354	AV. 2053427	AV. 5400

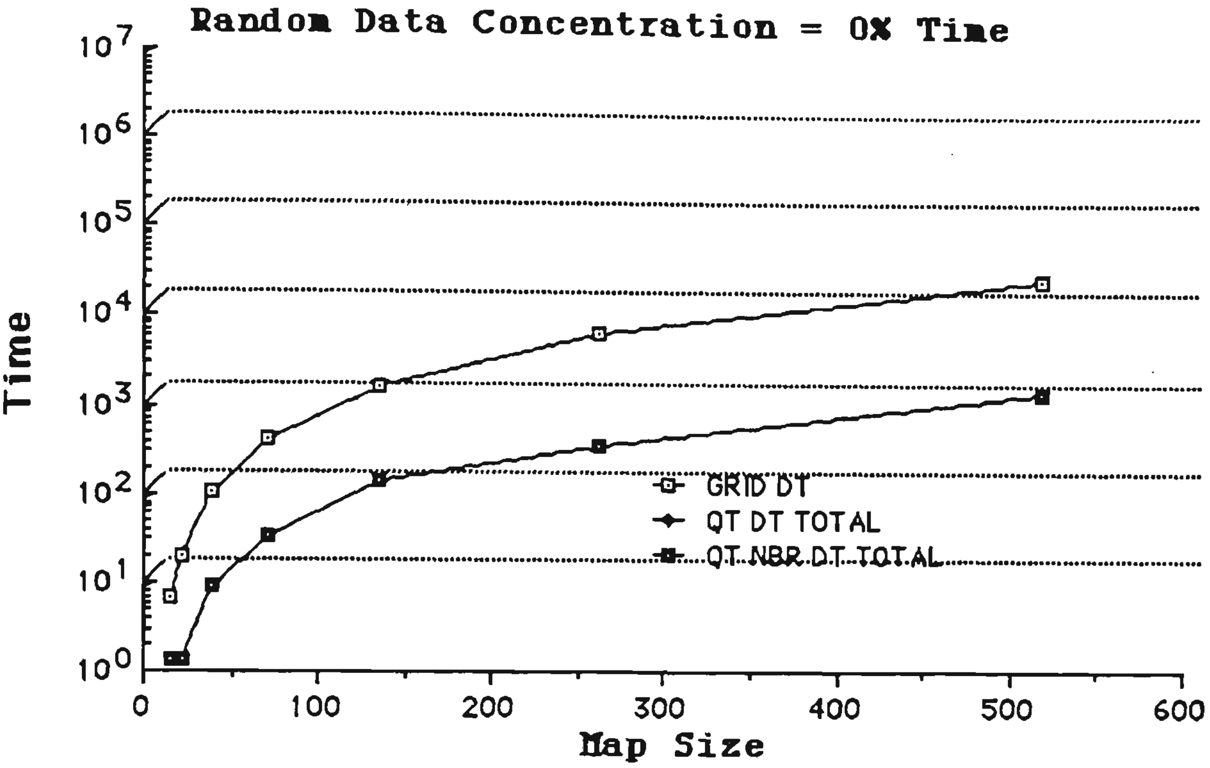
Table B.6
Path Planning Statistics for 256 x 256 map

SIZE: 512 × 512
Grid Memory 1048576

RANDOM MAP RESULTS

Percentage Concentration	GRID DT		BUILD DT		QT DT		QT DT TOTAL		QT MEMORY		QT DT NBR		BUILD QT NBR		QT NBR MEMORY		QT DT NBR TOTAL	
0	MAX	19536	MAX	1085	MAX	0	MAX	1085	MAX	32	MAX	0	MAX	0	MAX	48	MAX	1085
	MIN	19536	MIN	1085	MIN	0	MIN	1085	MIN	32	MIN	0	MIN	0	MIN	48	MIN	1085
	AV.	19536	AV.	1085	AV.	0	AV.	1085	AV.	32	AV.	0	AV.	0	AV.	48	AV.	1085
5	MAX	73640	MAX	1728	MAX	208741	MAX	210469	MAX	2381529	MAX	312886	MAX	2163	MAX	4846378	MAX	316777
	MIN	66212	MIN	1691	MIN	132699	MIN	134390	MIN	2289158	MIN	198905	MIN	2156	MIN	4578316	MIN	202752
	AV.	69926	AV.	1704	AV.	169164	AV.	170868	AV.	2341217	AV.	253563	AV.	2160	AV.	4705846	AV.	257427
10	MAX	136400	MAX	1877	MAX	476584	MAX	478461	MAX	3710970	MAX	305664	MAX	2876	MAX	7154922	MAX	310417
	MIN	105356	MIN	1852	MIN	371610	MIN	373462	MIN	3690117	MIN	238338	MIN	2869	MIN	7114717	MIN	243059
	AV.	124311	AV.	1867	AV.	434344	AV.	436211	AV.	3701109	AV.	278573	AV.	2872	AV.	7135906	AV.	283312
20	MAX	224412	MAX	2126	MAX	1570874	MAX	1573000	MAX	5001386	MAX	1047233	MAX	4008	MAX	9384960	MAX	1053367
	MIN	153136	MIN	2110	MIN	918816	MIN	920926	MIN	4824261	MIN	612534	MIN	3118	MIN	9052590	MIN	617762
	AV.	188797	AV.	2117	AV.	1132818	AV.	1134935	AV.	4912321	AV.	755200	AV.	3761	AV.	9217832	AV.	761078
30	MAX	192313	MAX	2128	MAX	1442348	MAX	1444476	MAX	4981386	MAX	805508	MAX	3666	MAX	8725443	MAX	811302
	MIN	154111	MIN	2112	MIN	1232888	MIN	1235000	MIN	4791222	MIN	688531	MIN	3441	MIN	8567510	MIN	694084
	AV.	173251	AV.	2119	AV.	1274725	AV.	1276844	AV.	4921242	AV.	785213	AV.	3512	AV.	8620094	AV.	790844
40	MAX	169616	MAX	2589	MAX	1381156	MAX	1386334	MAX	5378857	MAX	748282	MAX	3911	MAX	9000541	MAX	754782
	MIN	149696	MIN	2511	MIN	1197408	MIN	1199919	MIN	5361100	MIN	646308	MIN	3811	MIN	8970828	MIN	652630
	AV.	159691	AV.	2551	AV.	1247528	AV.	1250079	AV.	5369931	AV.	673361	AV.	3850	AV.	8985605	AV.	679762
50	MAX	24899	MAX	2496	MAX	149394	MAX	151890	MAX	5548814	MAX	73101	MAX	3751	MAX	8944279	MAX	79348
	MIN	20511	MIN	2376	MIN	123066	MIN	125442	MIN	5421179	MIN	61289	MIN	3690	MIN	8738541	MIN	67355
	AV.	22581	AV.	2471	AV.	137111	AV.	139582	AV.	5441182	AV.	67111	AV.	3712	AV.	8770784	AV.	73294
60	MAX	13344	MAX	2512	MAX	94108	MAX	96620	MAX	5121634	MAX	46026	MAX	3912	MAX	7831526	MAX	52450
	MIN	4941	MIN	2336	MIN	39848	MIN	42184	MIN	5109812	MIN	19489	MIN	3400	MIN	7813449	MIN	25225
	AV.	6498	AV.	2447	AV.	51232	AV.	53679	AV.	5118762	AV.	25056	AV.	3542	AV.	7827136	AV.	31045

Table B.7
Path Planning Statistics for 512 x 512 map



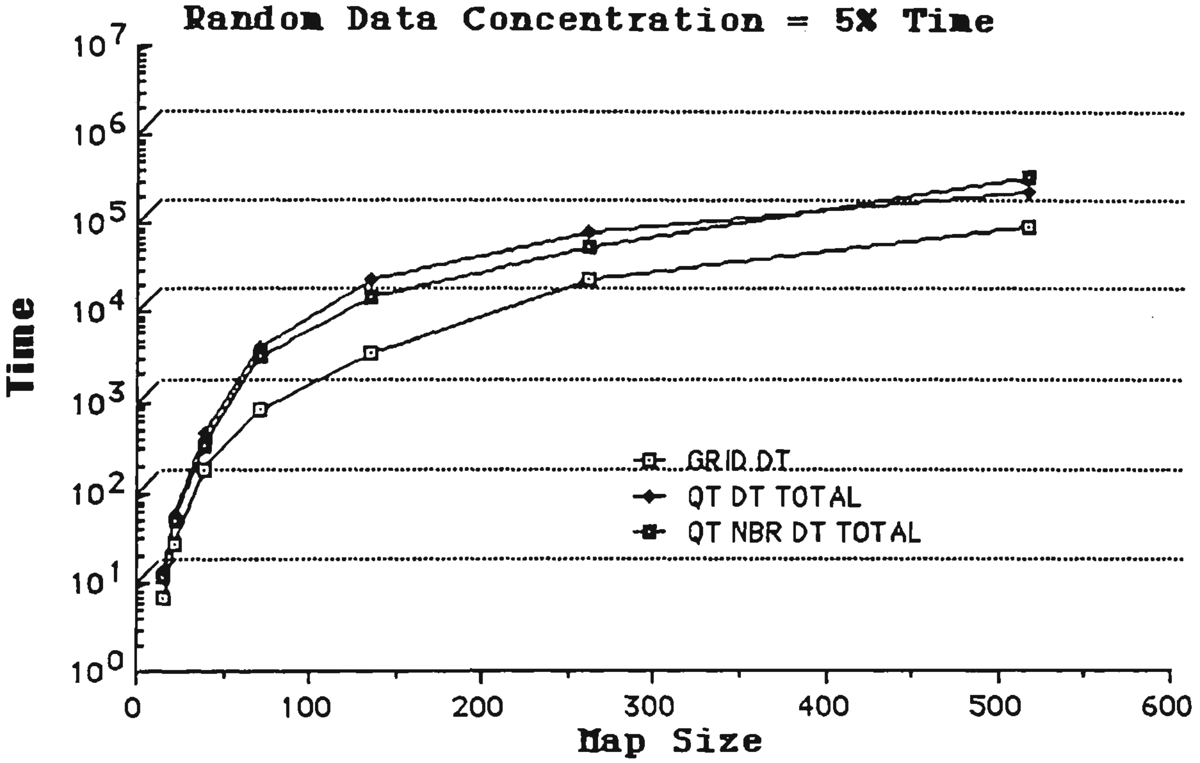
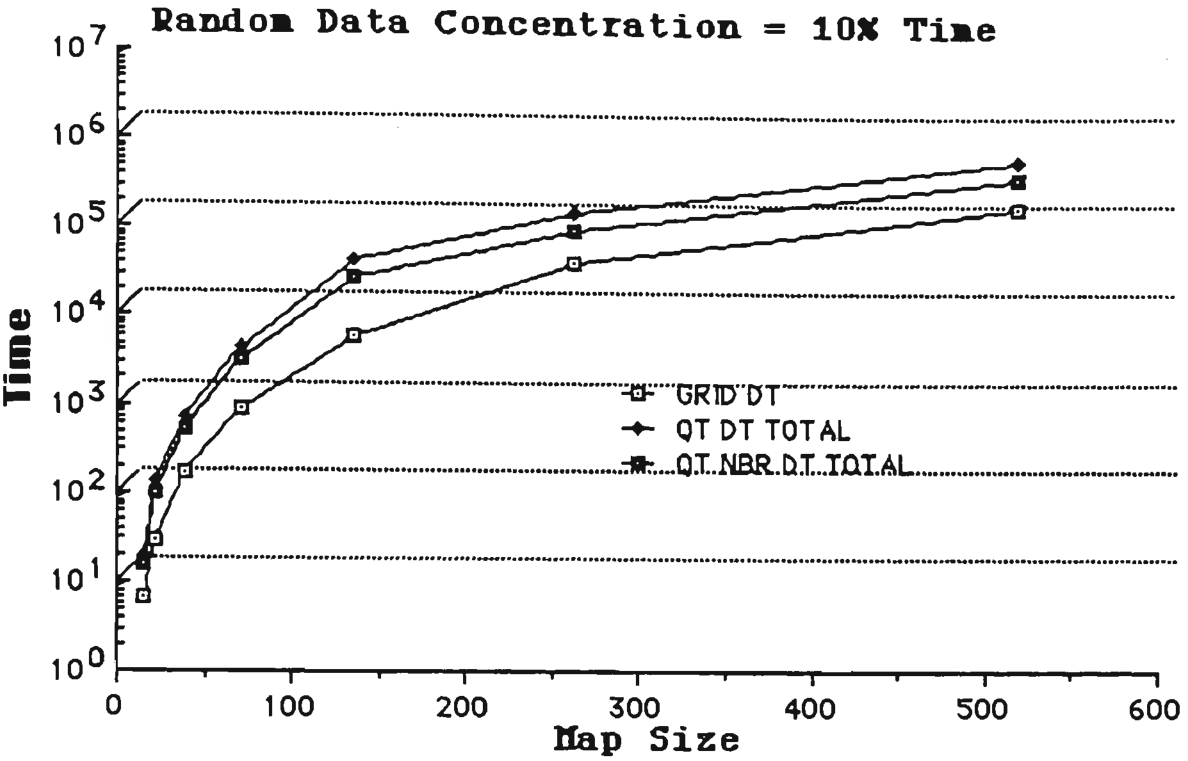


Figure B.3

Computation Times for path planning in 0% and 5% random environments.



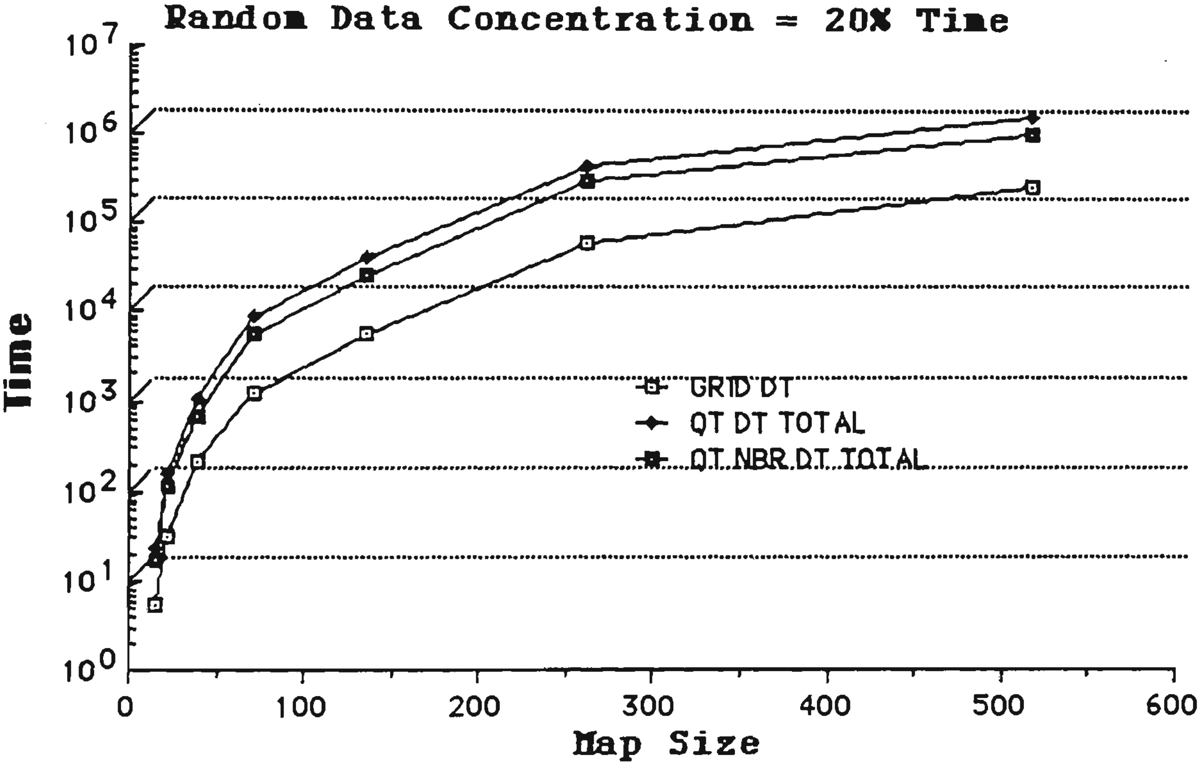
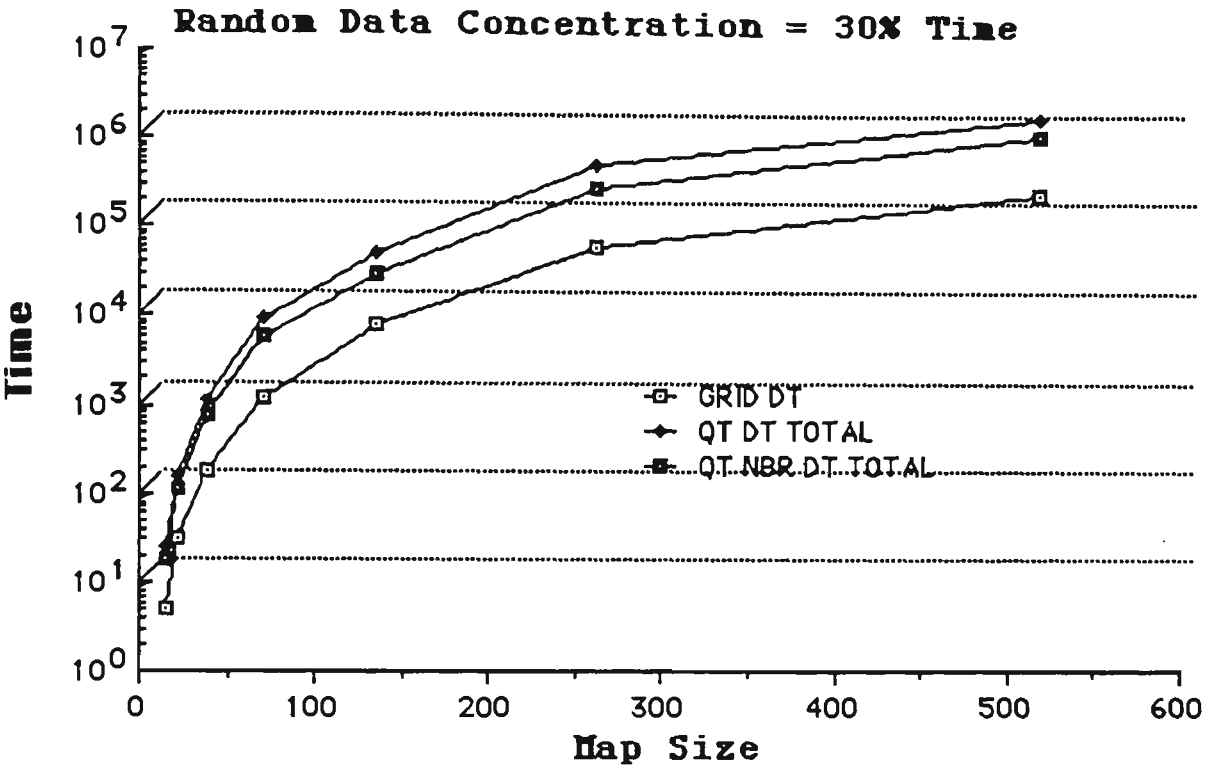


Figure B.4

Computation Times for path planning in 10% and 20% random environments.



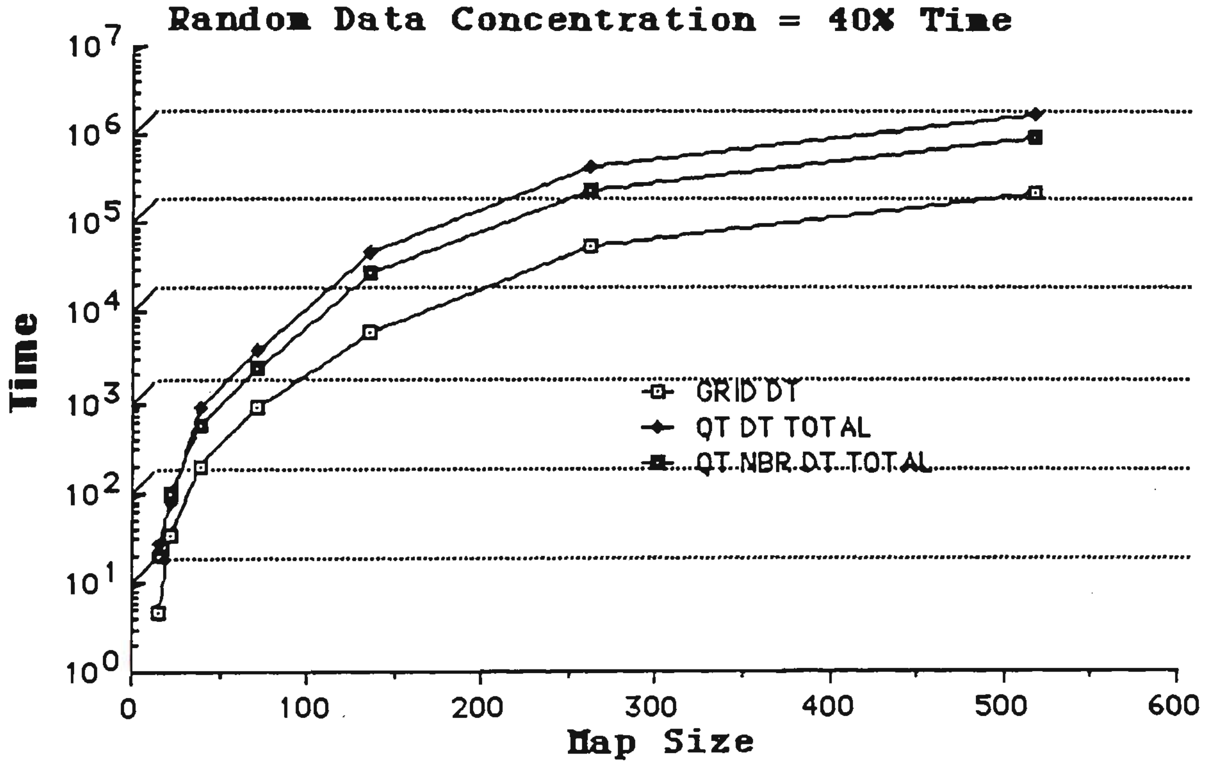


Figure B.5

Computation Times for path planning in 30% and 40% random environments.

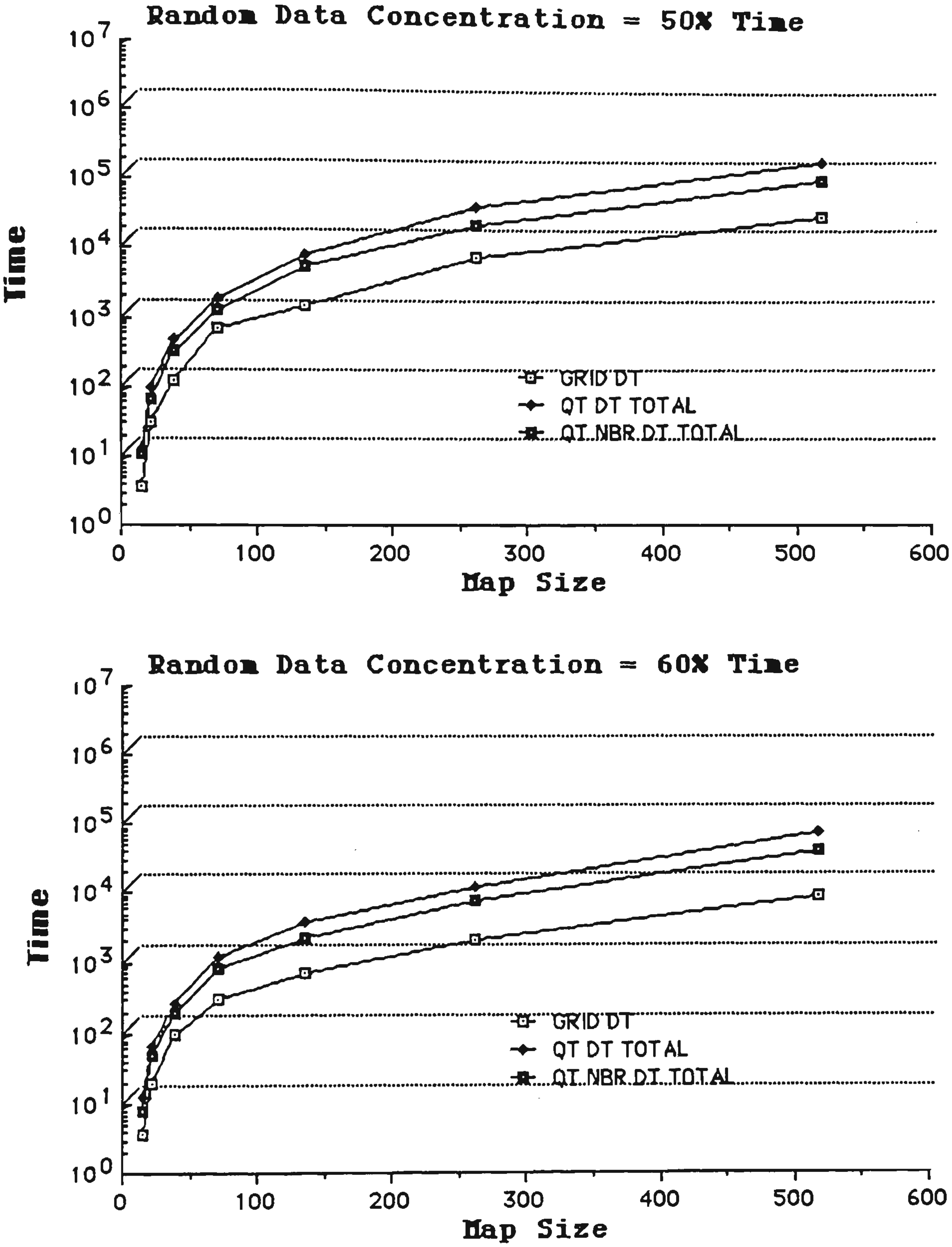
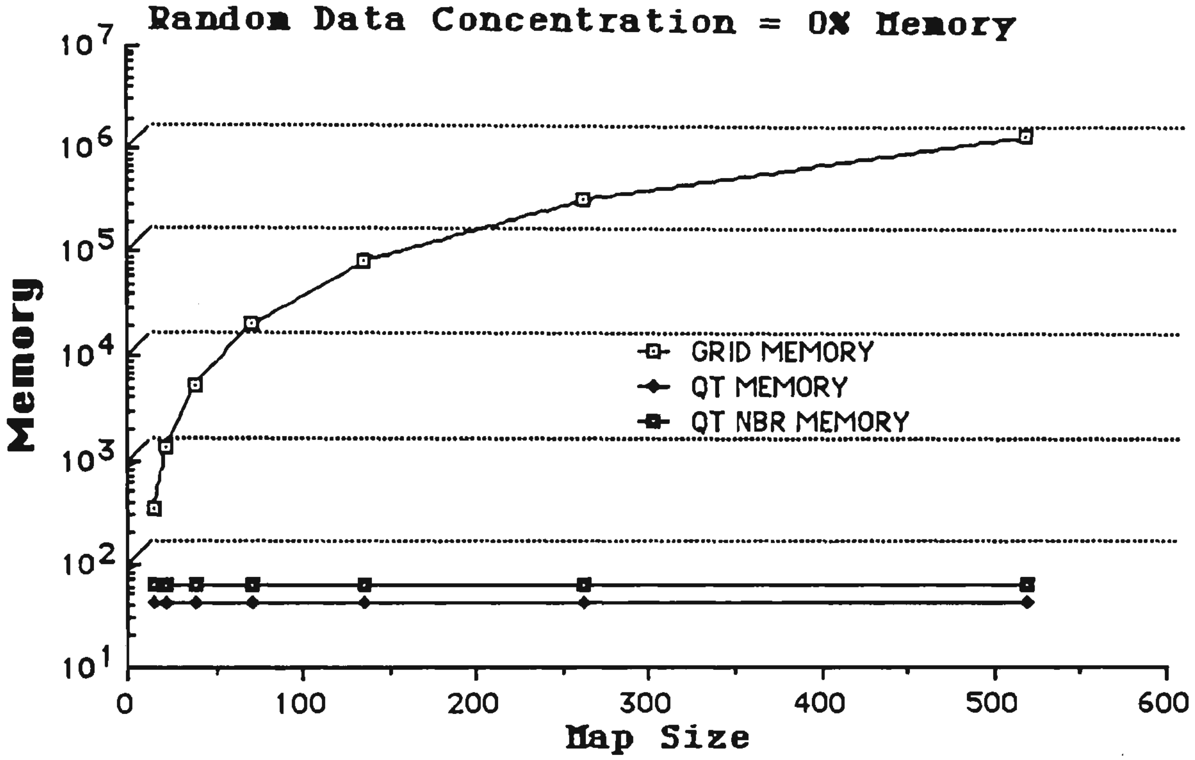


Figure B.6

Computation Times for path planning in 50% and 60% random environments.



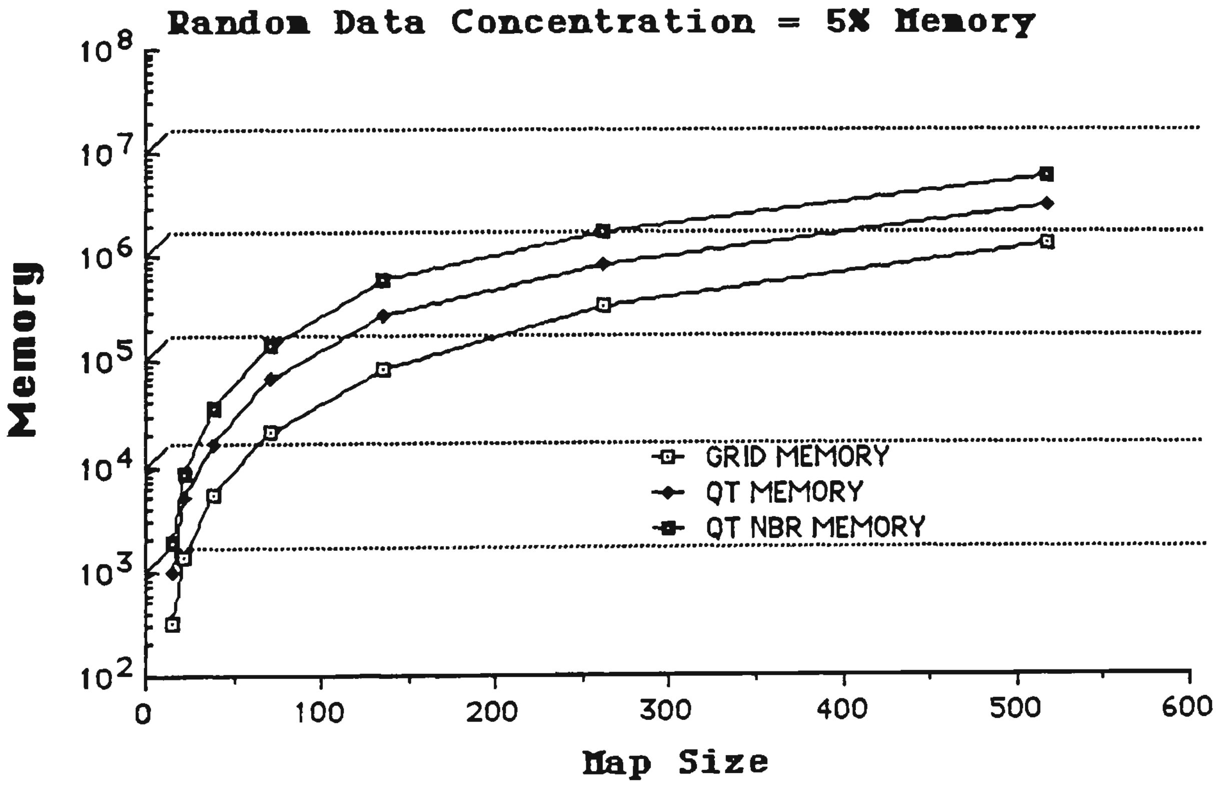
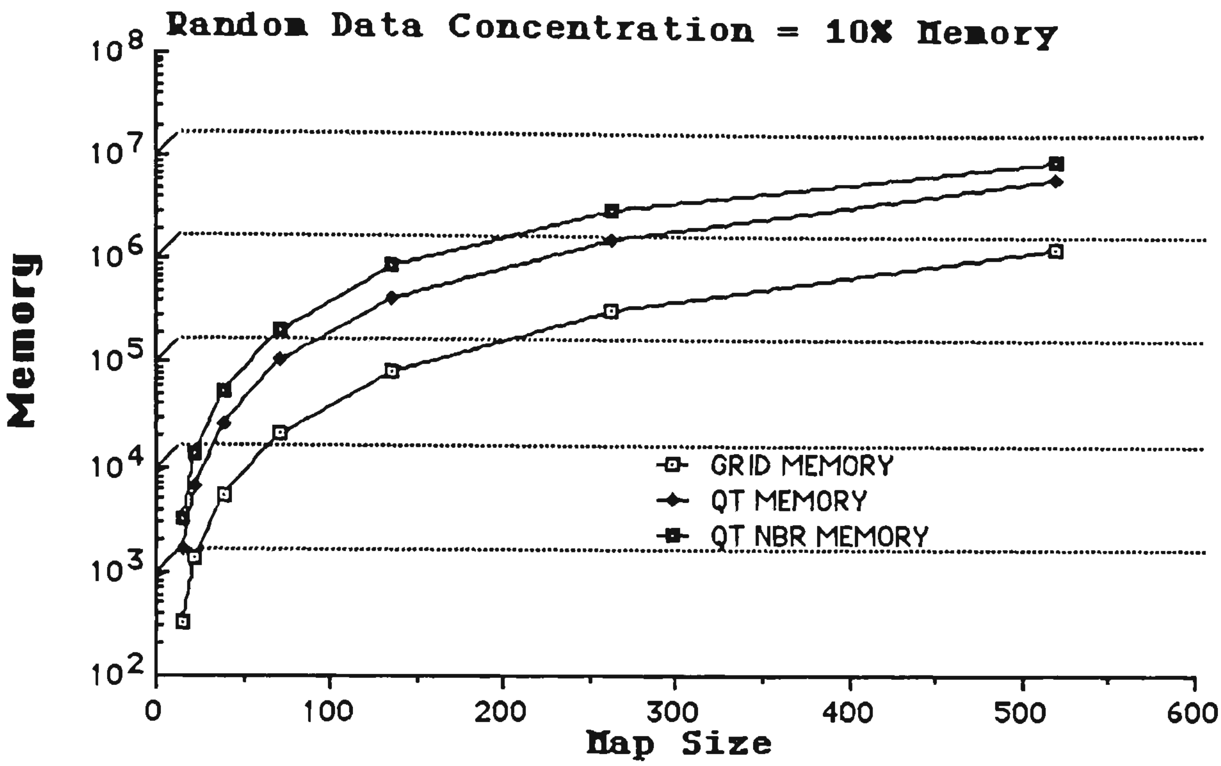


Figure B.7

Memory Requirements for path planning in 0% and 5% random environments.



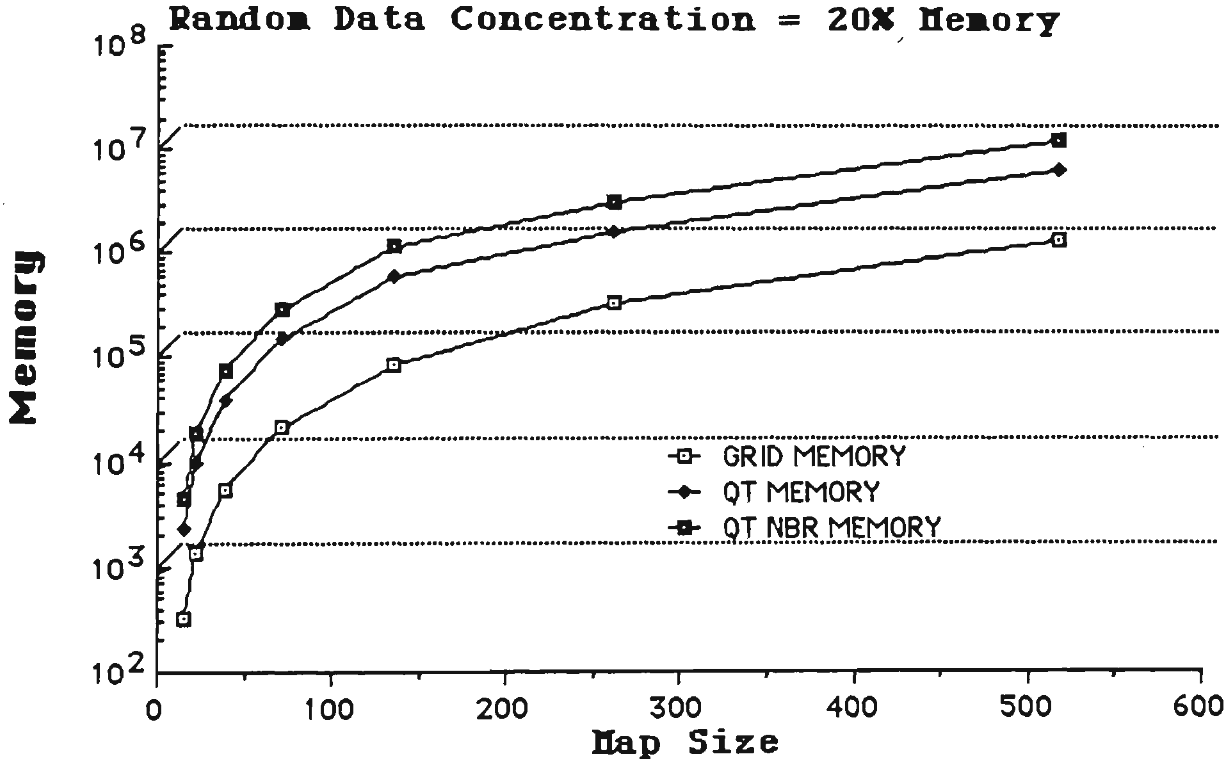


Figure B.8

Memory Requirements for path planning in 10% and 20% random environments.

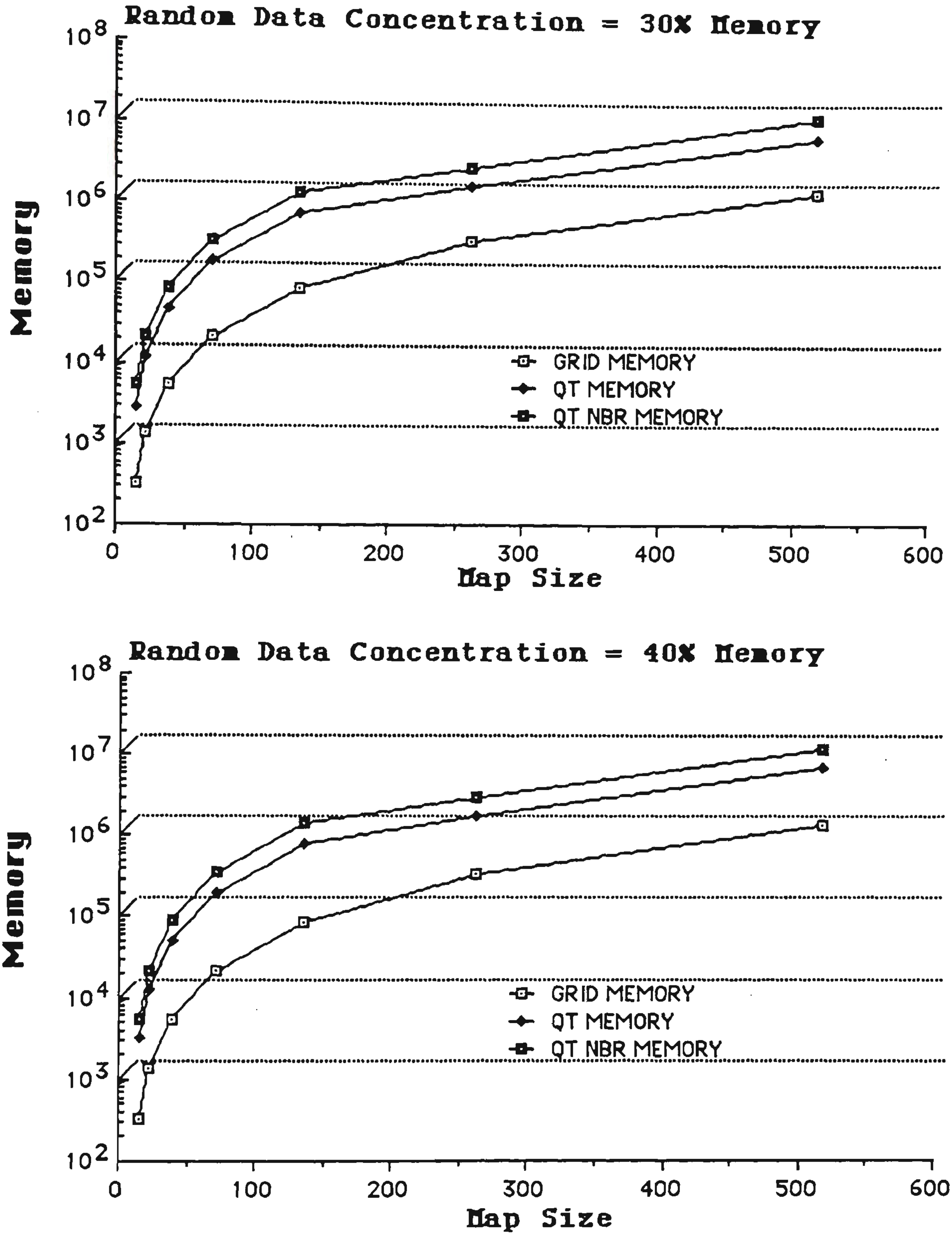
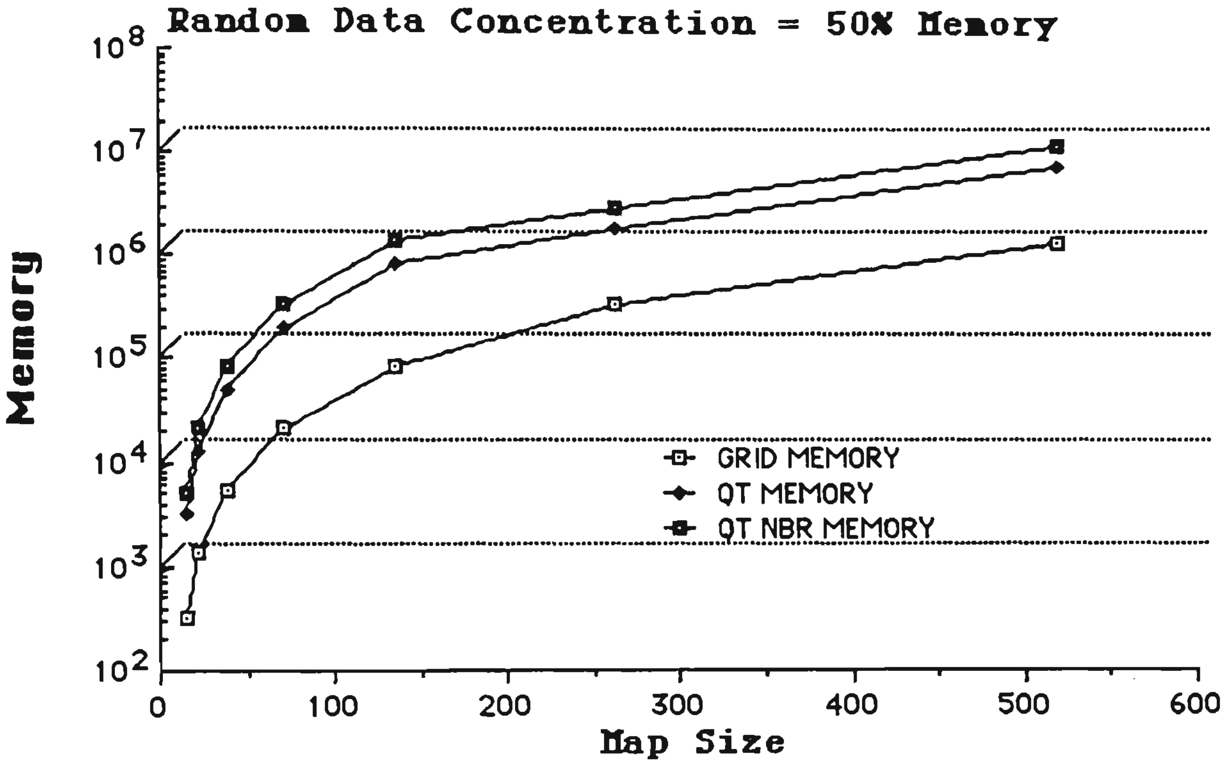


Figure B.9

Memory Requirements for path planning in 30% and 40% random environments.



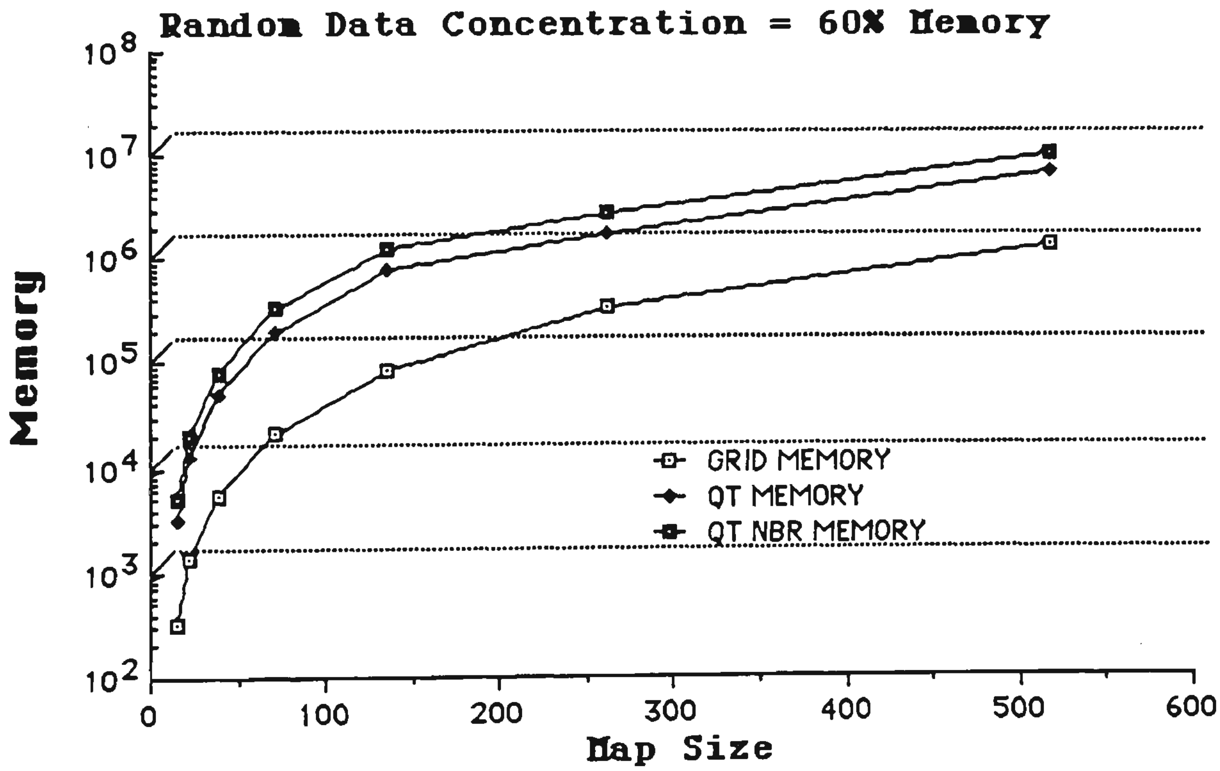


Figure B.10

Memory Requirements for path planning in 50% and 60% random environments.

Appendix C
Spiral and Maze

This appendix presents the experimental results for determining the amount of computational effort and computer memory that is required to produce the distance transform in grids and quadtrees in spiral and maze environments. The distance transform is computed for quadtrees with and without neighbour lists.

The Tables C.1 - C.2 presented in this appendix show the computation times and memory requirements that are needed to compute the distance transforms for the spiral and maze environments. The legend key to Tables C.1 - C.2 is the same as the key to tables presented in Appendix B.

Spiral Map Results

Map Size	Grid DT	Grid Memory	Build QT	QT DT	QT Total	QT Memory	QT NBR DT	QT NBR Build	QT NBR Total	QT NBR Memory
8 × 8	0	256	1	0	1	32	0	0	1	48
16 × 16	3	1024	2	3	5	2592	1	1	4	3920
32 × 32	29	4096	12	98	110	31904	50	28	90	51904
64 × 64	611	16384	52	12110	12162	99104	7860	133	8045	177648
128 × 128	2732	65536	139	23757	23896	202272	15073	294	15506	373680
256 × 256	11666	262144	523	57975	58498	466464	36942	693	38158	870224
512 × 512	49808	1048576	2082	141295	143377	1075732	90540	1621	94243	2026573

Table C.1

Path Planning Statistics for a Spiral Environment

Maze Map Results

Map Size	Grid DT	Grid Memory	Build QT	QT DT	QT Total	QT Memory	QT NBR DT	QT NBR Build	QT NBR Total	QT NBR Memory
8 × 8	1	256	0	1	1	384	1	0	1	724
16 × 16	21	1024	2	30	32	2412	17	3	22	3712
32 × 32	142	4096	9	727	736	24208	442	34	485	50214
64 × 64	585	16384	35	2182	2217	52640	1375	76	1486	95328
128 × 128	2095	65536	125	6606	6731	119328	4270	173	4568	222272
256 × 256	9110	262144	476	15761	16237	272288	10355	412	11243	519360
512 × 512	39614	1048576	1813	37604	39417	621320	25112	981	26093	1213534

Table C.2

Path Planning Statistics for a Maze Environment

Appendix D
Obstacles

This appendix presents the experimental results for determining the amount of computational effort and computer memory that is required to produce the distance transform in grids and quadtrees in environments with various configurations of obstacles. The distance transform is computed for quadtrees with and without neighbour lists.

The Tables D.1 - D.5 presented in this appendix show the computation times and memory requirements that are needed to compute the distance transforms for the five (5) configurations of obstacle data. The legend key to Tables D.1 - D.5 is the same as the key to tables presented in Appendix B, with the exception that Tables D.1 - D.5 also show the total perimeter length of all the obstacles in environment map. The perimeter is measured in quadrant cells, and is shown in the tables as PERIMETER.

Figures D.1 shows four of the five obstacle configurations that were used for this experiment. Figures D.1 (A) - (D) show the test environments with two, three, four and five obstacles. Figures D.2 - D.5 show in graphical form the computation times and the memory requirements for the experimental data presented in Tables D.1 - D.5.

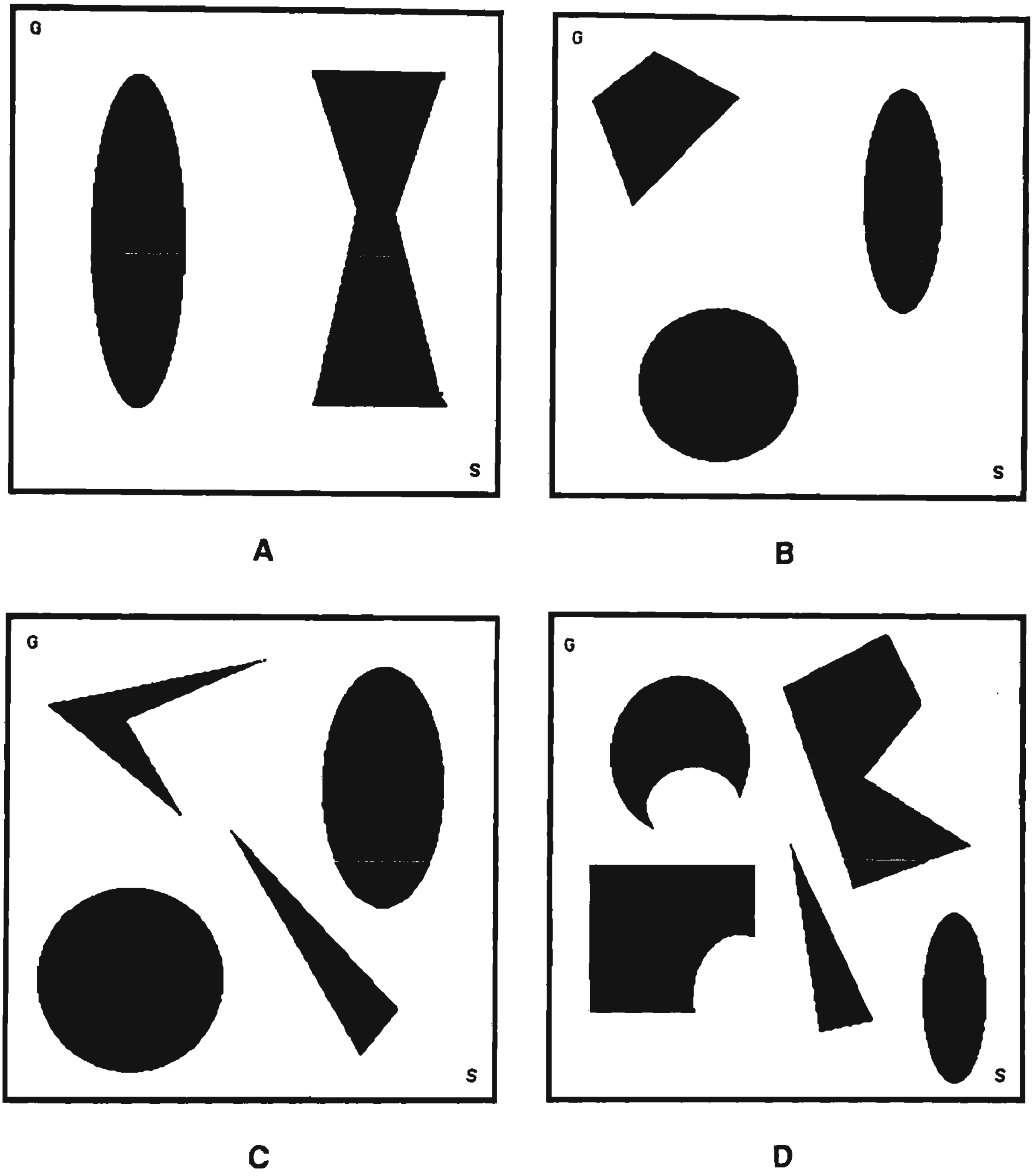


Figure D.1

Path Planning Environments of Two, Three, Four and Five Obstacles.

1 Obstacle Map Results

Map Size	Grid DT	Grid Memory	Build QT	QT DT	QT Total	QT Memory	QT NBR DT	QT NBR Build	QT NBR Total	QT NBR Memory	Perimeter
8 × 8	1	256	1	15	16	1184	10	3	14	2208	8
16 × 16	24	1024	1	41	42	2336	28	3	32	4496	12
32 × 32	102	4096	7	59	63	3872	40	6	53	7648	24
64 × 64	415	16384	27	191	218	6944	130	10	167	13360	46
128 × 128	1682	65536	112	434	547	14624	291	21	424	27872	93
256 × 256	6734	262144	416	831	1247	29088	577	48	1041	56800	190
512 × 512	26936	1048576	1545	1591	3136	57858	1144	110	2799	115752	382

Table D.1

Path Planning Statistics for a One Obstacle Environment

2 Obstacle Map Results

Map Size	Grid DT	Grid Memory	Build QT	QT DT	QT Total	QT Memory	QT NBR DT	QT NBR Build	QT NBR Total	QT NBR Memory	Perimeter
8 × 8	0	256	1	12	13	2720	8	3	12	4848	20
16 × 16	10	1024	2	15	17	3744	10	3	15	6640	34
32 × 32	43	4096	8	53	61	7712	36	7	51	14096	66
64 × 64	187	16384	23	177	200	17568	123	16	162	32864	134
128 × 128	799	65536	93	504	597	38816	352	38	483	73216	260
256 × 256	3158	262144	343	1076	1762	76064	772	84	1199	146928	522
512 × 512	12482	1048576	1272	2690	3962	149056	1530	210	3012	294852	1046

Table 6.D.2

Path Planning Statistics for a Two Obstacle Environment

3 Obstacle Map Results

Map Size	Grid DT	Grid Memory	Build QT	QT DT	QT Total	QT Memory	QT NBR DT	QT NBR Build	QT NBR Total	QT NBR Memory	Perimeter
8 × 8	1	256	0	3	3	2336	2	2	4	3904	24
16 × 16	10	1024	2	29	31	4384	18	6	26	7840	40
32 × 32	44	4096	6	93	99	9632	62	9	77	17744	68
64 × 64	187	16384	24	177	201	21664	124	24	172	41184	142
128 × 128	767	65536	95	789	884	42784	549	43	687	80784	286
256 × 256	3168	262144	351	1567	1918	86432	1114	106	1571	169088	572
512 × 512	13084	1048576	1297	3134	4431	174610	2260	260	3817	359916	1144

Table D.3

Path Planning Statistics for a Three Obstacle Environment

4 Obstacle Map Results

Map Size	Grid DT	Grid Memory	Build QT	QT DT	QT Total	QT Memory	QT NBR DT	QT NBR Build	QT NBR Total	QT NBR Memory	Perimeter
8 × 8	0	256	1	4	5	1696	2	1	4	2704	21
16 × 16	8	1024	2	39	41	6688	24	5	31	11696	43
32 × 32	41	4096	7	107	114	13600	72	17	96	25472	106
64 × 64	251	16384	26	415	441	31392	286	33	345	59168	212
128 × 128	1026	65536	101	851	952	64672	587	67	753	121856	424
256 × 256	4291	262144	362	2019	2381	128160	1439	163	1964	253056	846
512 × 512	17946	1048576	1308	4780	6088	256320	3528	408	5244	506112	1690

Table D.4

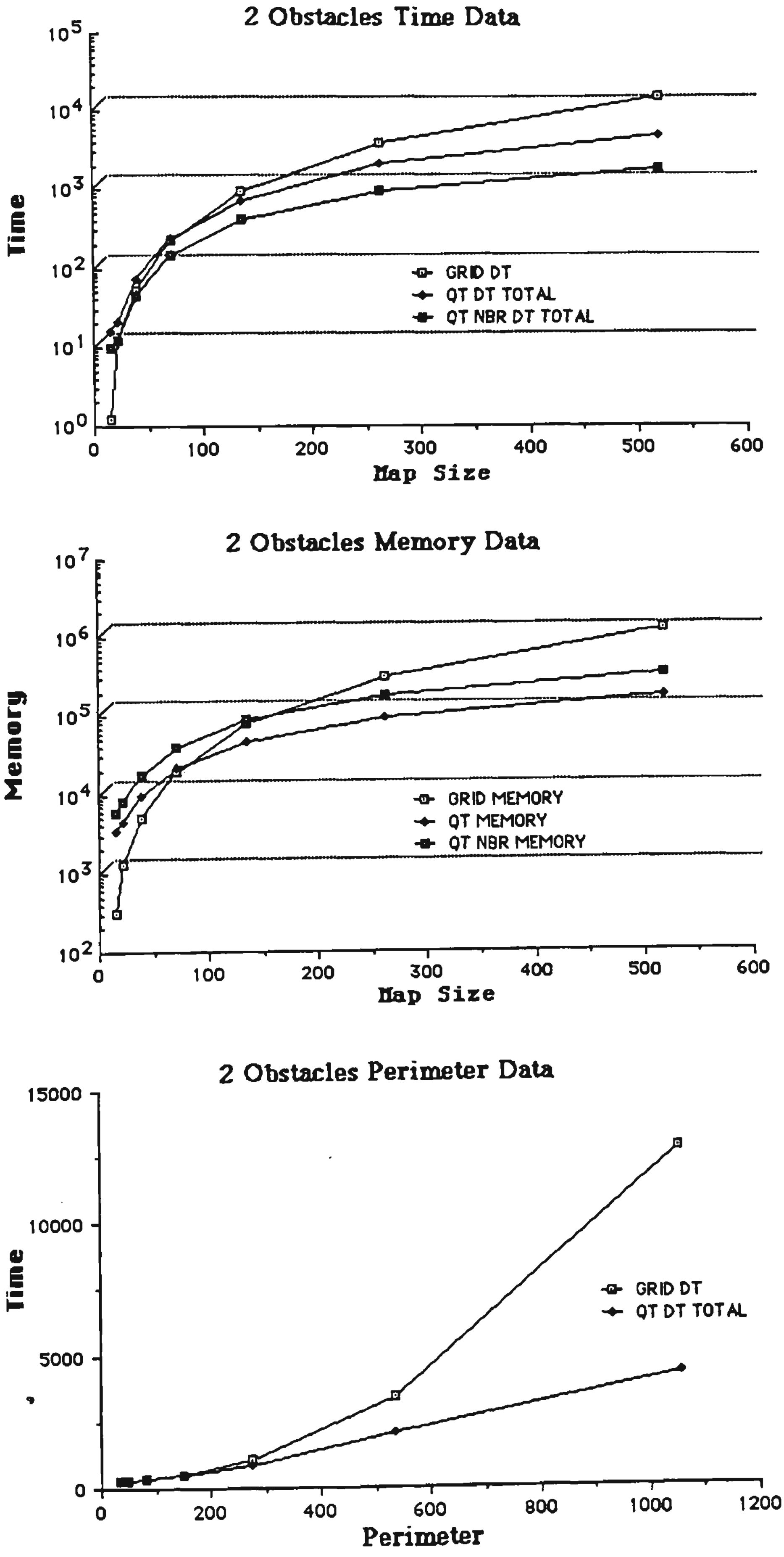
Path Planning Statistics for a Four Obstacle Environment

5 Obstacle Map Results

Map Size	Grid DT	Grid Memory	Build QT	QT DT	QT Total	QT Memory	QT NBR DT	QT NBR Build	QT NBR Total	QT NBR Memory	Perimeter
8 × 8	1	256	1	4	5	1952	3	1	5	3168	28
16 × 16	11	1024	2	32	34	6048	19	5	26	10336	64
32 × 32	58	4096	7	72	79	15648	47	15	69	28480	136
64 × 64	164	16384	28	350	378	34592	240	36	304	64480	280
128 × 128	697	65536	101	1243	1344	68256	853	71	1025	128304	562
256 × 256	2761	262144	360	3033	3393	143264	2165	185	2701	280368	1048
512 × 512	10938	1048576	1283	7400	8683	286528	5450	482	7215	560736	2094

Table D.5

Path Planning Statistics for a Five Obstacle Environment

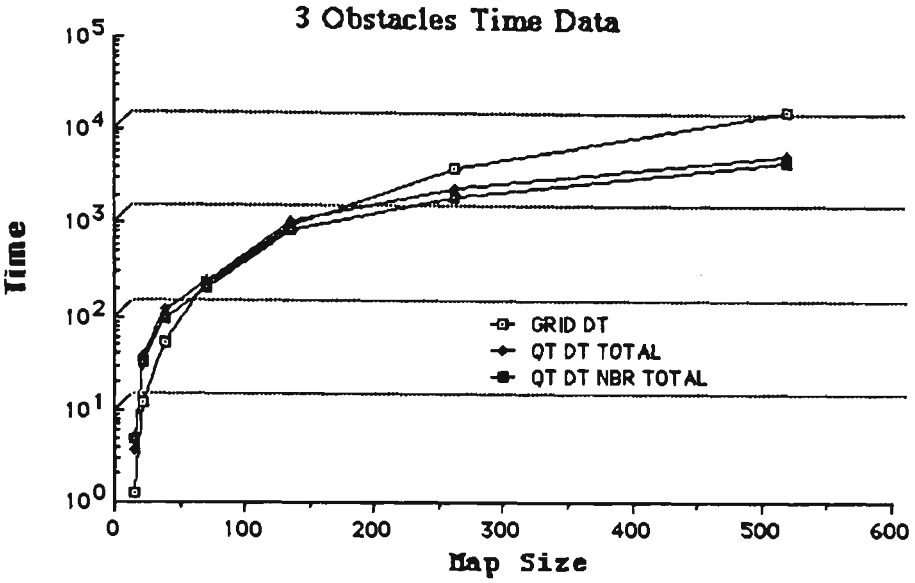

2 Obstacles Time Data

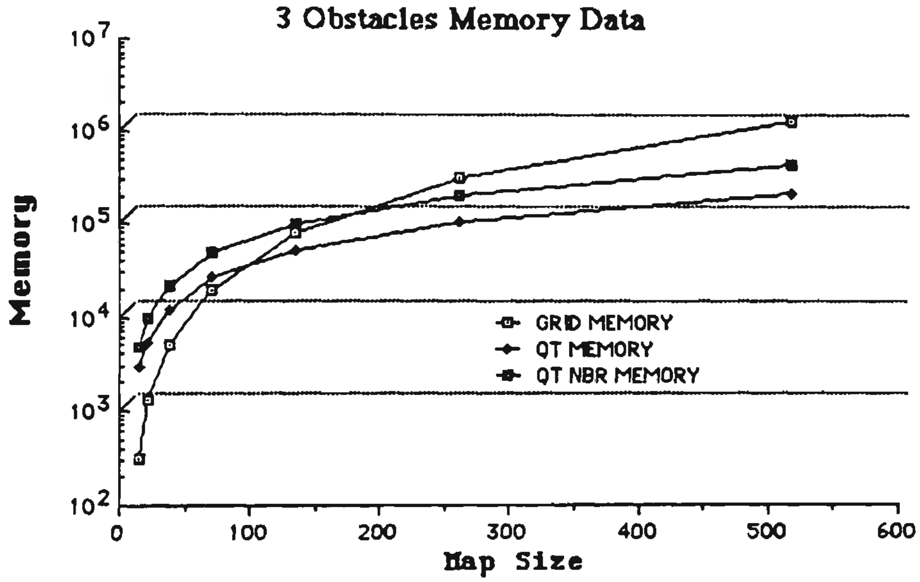
2 Obstacles Memory Data

2 Obstacles Perimeter Data

Figure D.2

Path Planning Statistics for a Two Obstacle environment.





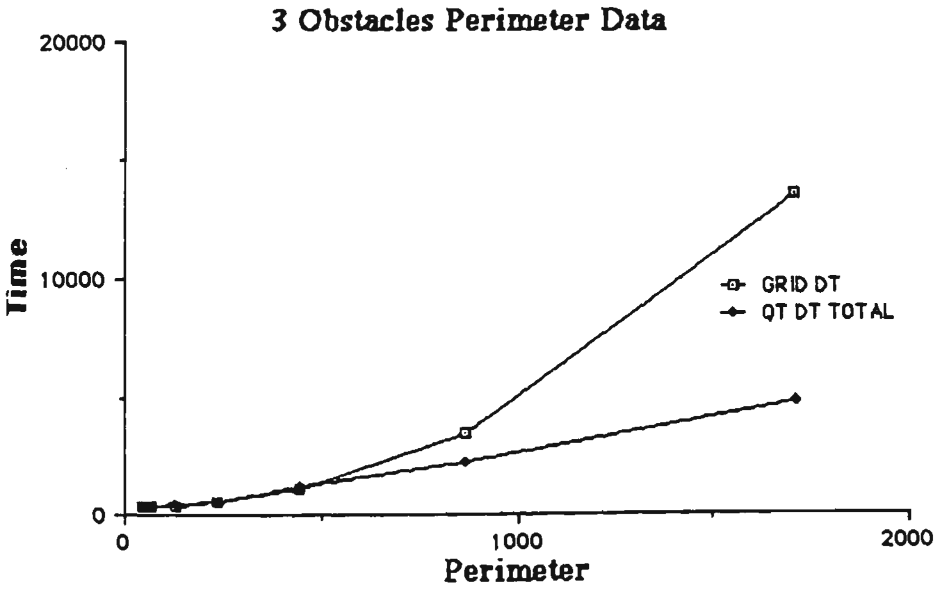


Figure D.3

Path Planning Statistics for a Three Obstacle environment.

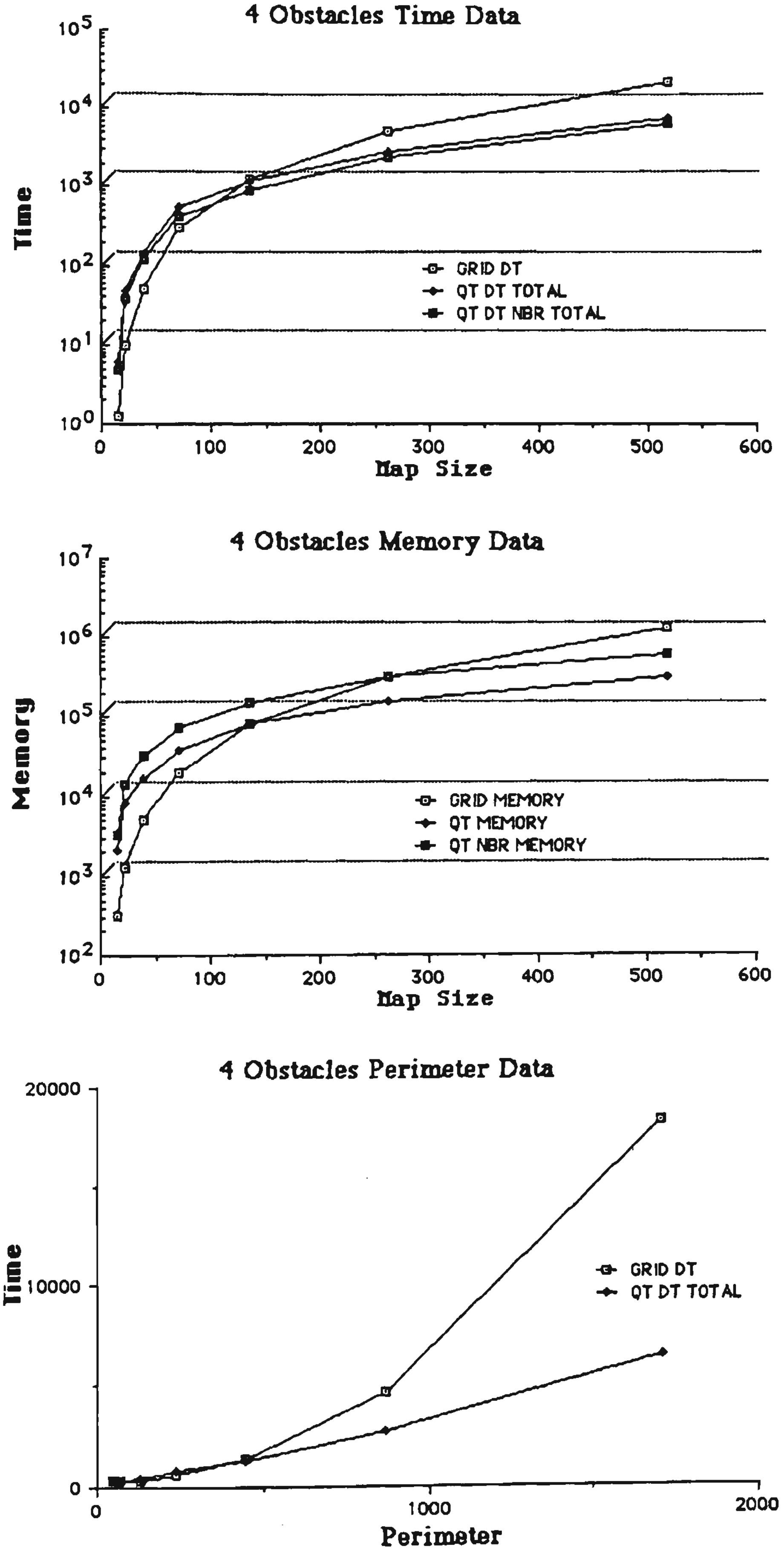


Figure D.4

Path Planning Statistics for a Four Obstacle environment.

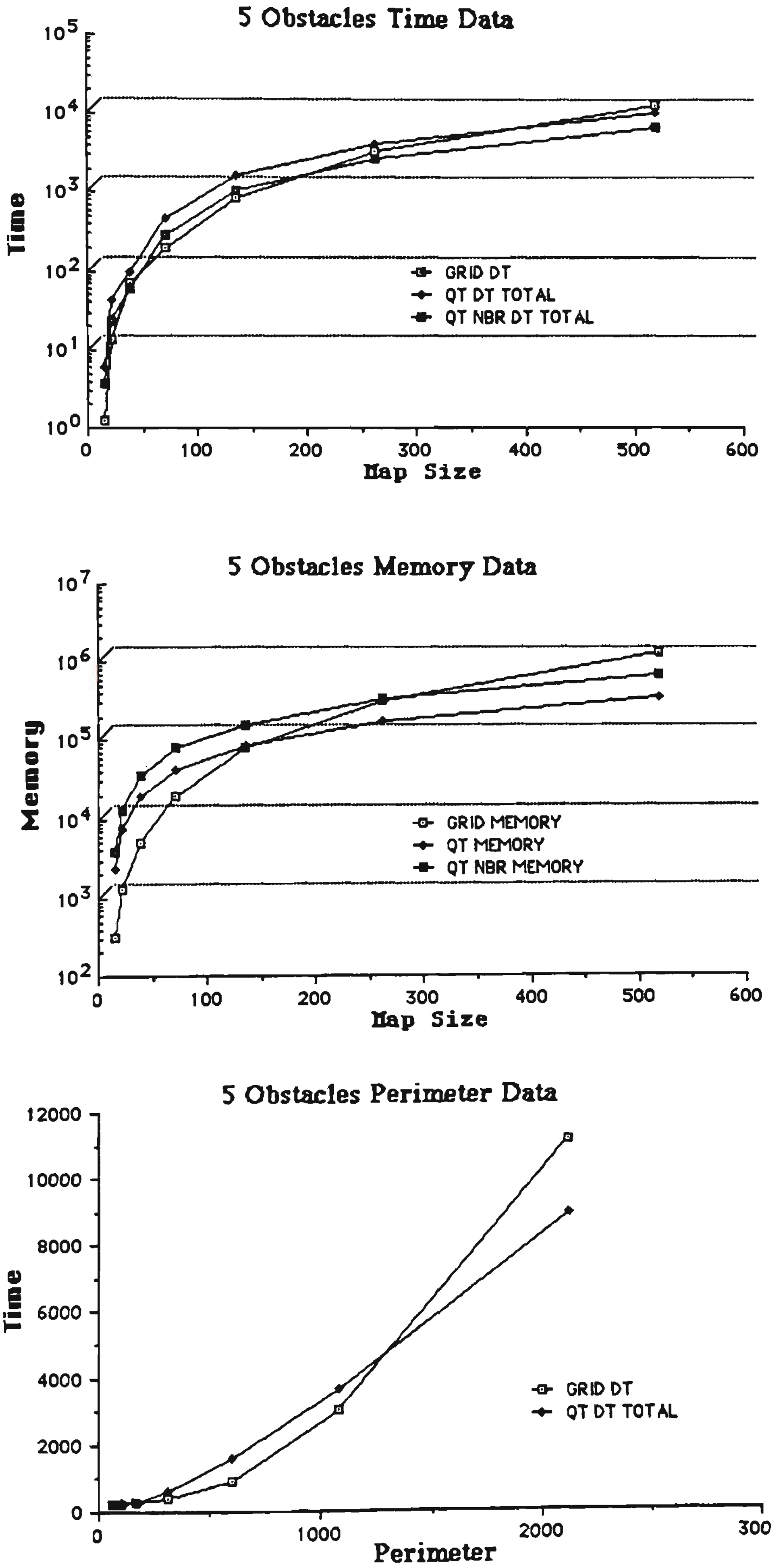


Figure D.5

Path Planning Statistics for a Five Obstacle environment.

Appendix E
Indoor Environment

This appendix presents the experimental results for determining the amount of computational effort and computer memory that is required to produce the distance transform in grids and quadtrees in three (3) indoor environments. The distance transform is computed for quadtrees with and without neighbour lists.

The Tables E.1 - E.3 presented in this appendix show the computation times and memory requirements that are needed to compute the distance transforms for the indoor environments. The legend key to Tables E.1 - E.3 is the same as the key to tables presented in Appendix B.

Computer Laboratory Map Results

Map Size	Grid DT	Grid Memory	Build QT	QT DT	QT Total	QT Memory	QT NBR DT	QT NBR Build	QT NBR Total	QT NBR Memory
8 × 8	1	256	0	1	1	1184	1	1	3	1888
16 × 16	14	1024	2	52	54	6304	32	7	41	10816
32 × 32	103	4096	10	306	316	21152	186	29	235	38832
64 × 64	318	16384	30	320	350	25760	202	37	269	47776
128 × 128	1277	65536	115	570	673	32160	361	44	508	59120
256 × 256	5103	262144	419	798	1217	50720	502	72	993	94240
512 × 512	20392	1048576	1676	1118	2794	79990	698	118	2492	150220

Table E.1

Path Planning Statistics for a Computer Laboratory Environment

Horse Stables Map Results

Map Size	Grid DT	Grid Memory	Build QT	QT DT	QT Total	QT Memory	QT NBR DT	QT NBR Build	QT NBR Total	QT NBR Memory
8 × 8	0	256	1	1	2	416	1	0	2	656
16 × 16	8	1024	3	27	30	9120	14	8	25	14784
32 × 32	77	4096	11	418	429	29088	258	42	311	52960
64 × 64	403	16384	35	644	679	36128	403	50	488	66384
128 × 128	1620	65536	117	805	922	42656	511	59	687	79520
256 × 256	6581	262144	412	1132	1544	54304	735	85	1232	103808
512 × 512	26324	1048576	1641	1415	3056	61702	952	108	2701	134124

Table E.2

Path Planning Statistics for a Horse Stable Environment

Room Map Results

Map Size	Grid DT	Grid Memory	Build QT	QT DT	QT Total	QT Memory	QT NBR DT	QT NBR Build	QT NBR Total	QT NBR Memory
8 × 8	3	256	1	8	9	1568	5	1	7	2608
16 × 16	14	1024	3	68	71	5920	40	7	50	10288
32 × 32	92	4096	14	250	264	17824	159	22	195	32432
64 × 64	398	16384	34	604	638	34464	385	46	465	63280
128 × 128	1706	65536	135	2709	2847	115360	1734	150	2022	212688
256 × 256	6818	262144	435	3412	3850	120984	2121	4178	2734	241714
512 × 512	27270	1048576	1735	4311	7786	125112	2711	201	4647	252798

Table

E.3

Path Planning Statistics for a House With 2 Rooms Environment