Analysis of Adaptive Reactive Control Systems for Autonomous Mobile Robots

by

Gary Brightwell

Submitted in fulfilment of the requirements for the degree of

Doctor of Philosophy

Department of Physics and Electronic Engineering

UNIVERSITY OF WAIKATO

APRIL 1999

Abstract

This thesis presents techniques developed for the robust reactive control of autonomous mobile robots navigating in complex environments. Traditional approaches have adopted a top-down perspective by focusing on the reasoning part of path-planning. In contrast, this thesis tackles planning from a bottom-up perspective which is characterised by a tight coupling between perception and action.

With this view in mind, control systems ranging from pure reactive controllers to controllers that can construct spatial representations of the environment are investigated. Hand-designed techniques as well as evolutionary approaches are developed and analysed. Quantitative simulation experiments measuring the performance of each control system are presented along with a new algorithm for developing topological maps of the environment from laser range-finder sensory information.

Preface

Autonomous mobile robots have the potential to be useful in a wide variety of domains, from delivering packages in office buildings to cleaning living-rooms to surveying the ocean depths. To realise these potential applications, an autonomous mobile robot needs to be able to navigate robustly within its situated real world environment. Robust navigation in a real world environment requires the robot to be able to construct accurate maps of its environment and be able to localise itself within this map. Further, robust navigation also requires the robot to have the ability to react to changes in the environment and to avoid new or moving obstacles without the need for complex path-planning.

This thesis is presented as follows:

Chapter 1 of this thesis is in the form of an introduction, providing an overview of the field of autonomous mobile robotics, from its beginnings with the micromouse contest to the current issues in this field of research.

Chapter 2 provides the background and related work already done in the field. It develops the rationale behind the need for a reactive control system when navigating in real world environments. These complex environments require maps to allow robust navigation and two different techniques of generating such spatial representations are contrasted with each other. Approaches to designing such reactive control systems are also presented.

Chapter 3 details the simulator that was developed to test the various controllers. The simulator models not only the environment but also the autonomous mobile robot itself. Simulation of the laser range-finder is detailed along with the kinematics of the autonomous mobile robot.

Chapter 4 discusses the first two developed controllers, the basic static reactive controller and the adaptive reactive controller. Results of quantitative simulation experiments are provided and the shortcomings of such systems are analysed.

Chapter 5 investigates the viability of using evolutionary techniques to design a neural network controller. Results of quantitative simulation experiments are provided and the shortcomings of such systems are analysed.

Chapter 6 details the final control system developed. This controller, based on the adaptive reactive controller, is equipped with a memory in the form of a topological map that is created 'on-the-fly' as the autonomous mobile robot explores its environment. Results of quantitative simulation experiments are provided. The implementation issues (i.e. processing time and memory requirements) on a physical robot are discussed.

Chapter 7 provides the conclusion to this thesis, listing the contributions made from this research along with its limitations and future research possibilities.

Acknowledgements

This thesis is dedicated to my parents.

I would like to thank everyone who contributed to the development of my robot. Of course, this time I was a bit wiser and never built one.

Table of Contents

List of Figures

List of Tables

List of Equations

Chapter 1

1. Introduction

1.1 Autonomous Mobile Robots

In 1979 the first international micromouse competition was held. This was one of the first attempts at building autonomous mobile robots. Every year since then competitors have been trying to design and build faster, more robust mice that can intelligently solve a maze. With the advancement made in technology over the last three decades, the micromouse competition is now somewhat trivial and to a large extent has been relegated to a useful teaching tool when once it was a 'state-of-the-art competition.' However, the world of micromice is still far removed from the world in which we live. The problem of how to design an autonomous mobile robot for our everyday environment is still an active area of research.

The micromouse competition is based around a 16 square by 16 square maze. This results in an environment that is very 'square' and hence easily mapped and stored in

memory. An array of two hundred and fifty-six bytes can easily store a map of the environment. In the real world, life is rarely 'square' and therefore it is far more difficult to efficiently store maps of the environment. For each competition, the maze is in an unknown configuration. This means the micromouse must search through the maze to find the fastest path to the middle but need never have to worry about changes taking place because it is in a static environment. So once the maze is learned it never has to alter its map of it. This is not a valid assumption in the real world. One of the major characteristics of a real world environment is that it is usually dynamic. Hence any map of such an environment will normally become obsolete so an ability to update the map is critical to the performance of the robot in a real world environment. A comparison between a real world environment and a maze is shown in Figure 1-1.

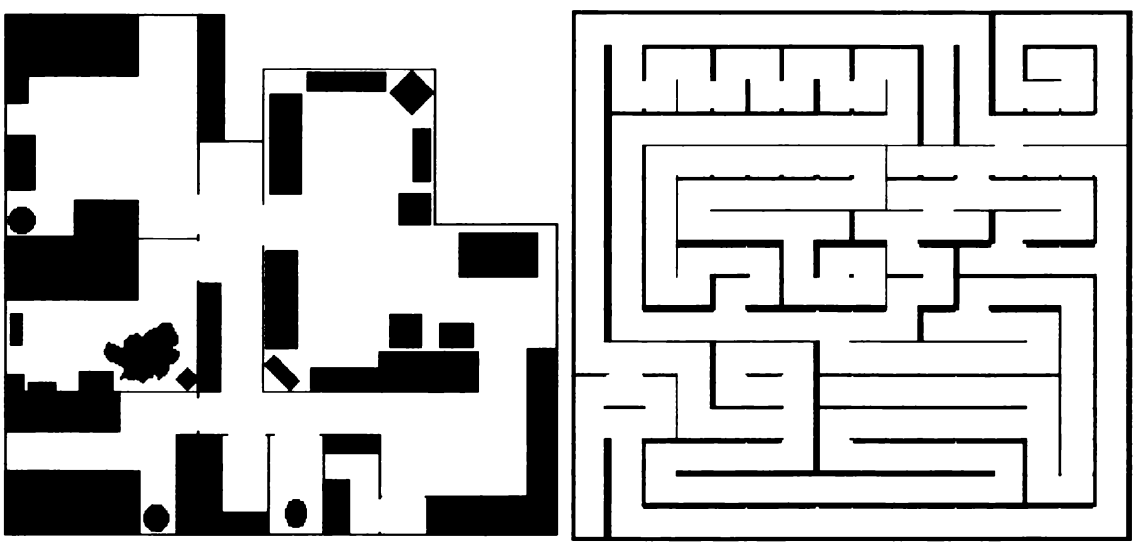

Figure 1-1 Contrasting a real world environment (layout of a house) to that of a typical environment of a micromouse.

An autonomous mobile robot (AMR) has potential applications in a wide variety of domains: from delivering packages in an office building to exploring the surface of another world, from driving a vehicle on highways to fighting fires on ships, from cleaning living-rooms to acting as an aide for the handicapped, and from performing reconnaissance on battlefields to surveying the ocean depths. All these applications are far removed from micromice running through mazes yet the fundamental requirements are still the same:

- The AMR must be able to construct accurate maps of the environment. These may be geometric or topological.

- The AMR must be able to localise itself within this map for the map to be of any great use.

- The AMR must be able to decide where to go next (even if the environment is static).

Further, these applications also require the robot to have the ability to react to changes in the environment and to avoid new or moving obstacles without the need for complex path-planning. This form of control is termed "reactive control." A reactive control system can process the robot's sensory information in 'real-time' to provide a continuous output to guide the robot.

1.2 Constraints for AMR design

When designing a micromouse there are many constraints which have to be adhered to which tend to complicate the design. A micromouse ideally needs to be less than twelve centimetres in diameter, weigh well under one kilogram and yet must carry enough batteries for fifteen minutes of operation under race conditions. These size constraints typically influence the choice of motors, gearing, batteries, sensors and processor to optimise the performance of the mouse. Of course, there are always the budget considerations that impose further limits upon the design as well.

Just as the micromouse has copious design constraints which add to the complexity of the problem, the design of an AMR for a real world environment also imposes numerous constraints which complicate the process. Size is still a problem, though not as serious as that of a micromouse which is bound by eighteen centimetre squares. A domestic robot for example, has approximately seventy centimetre doorways to negotiate. The major problem, however, with a design for an AMR in a real world

environment is the need to process the immense amount of sensory data that can be made available to the robot through the use of laser range-finders and video cameras etc. Ideally the robot needs as much processing power as possible (within budget constraints) but also power consumption must be taken into account. A DEC Alpha processor running at 500 megahertz will consume twenty-five watts of power, while lead acid batteries only have an energy density of approximately thirty watt-hours per kilogram. This limits the energy capacity of a small domestic robot to a few hundred watt-hours. Obviously processor selection is important and while a DEC Alpha or equivalent processor would be useful, they are not the most viable solution.

Owing to these size-, power- and budget-constraints, AMR designs have limited processing capabilities. So rather than "brute force" number-crunching of sensor data, new ways of extracting essential information from the AMR's sensors need to be developed. Representations of the AMR's environment must be simple enough to be manipulated with a low power processor in real-time to enable truly reactive control. These constraints are the main reasons why robust control of AMRs has yet to be achieved.

1.3 Robust Control of AMRs

Humans take for granted so much in their lives. We wander from room to room in our houses without really thinking about what we are doing. We walk into the living room and find it rearranged yet it seems so simple to make our way through this altered room to the room next door. Of course it looked different to how it was last time we were there, but we still knew that it was the same living room. We walk down the passage-way through the kitchen and back to the living room. We understand that we are not in a new room very similar to our living room, but have actually walked in a circle and are back in our living room, even though we had not noticed that we had rotated nearly three hundred and sixty degrees as we walked. Meanwhile a few million optic nerves have been transmitting conditioned sensory information from our retina to the rear of our brain. This immense amount of data is somehow processed and transformed into

high-level representations of our environment (e.g. living room, kitchen, and passage-way).

What humans find so simple are actually some of the most difficult issues in designing robust control systems for AMRs. The problem reduces down to a question of transfer functions. Somehow, our bodies manage to transform chemical reactions generated by collisions of photons into high-level abstract representations of our environment and they do this extremely well. With an AMR which is equipped with a laser range-finder, the problem equates to having a digitised array of data representing your environment and somehow performing a transform on these data to generate a high-level representation of the environment that uses very little memory and therefore can be manipulated with existing low-power microcontrollers in real-time.

Current research focuses (through a plethora of programming techniques such as case-based reasoning, neural networks, and fuzzy logic) on how to develop transforms from these sensory data to some form of output (in this case torque from motors that control the motion of the robot) to yield robust control of an AMR. Amongst these various groups there are two main philosophies which are summarised below.

1.3.1 Evolutionary Method

This technique tries to evolve a transfer function through a process of evolution as Darwin proposed for the origins of life. It is a random process and uses natural selection to evolve and adapt to environments.

1.3.2 Creationist Method

This technique tries to create a complex architecture using available knowledge of the domain that the AMR will function in to yield a transfer function that transforms sensory input to the appropriate output (action).

1.3.3 Viability of these Methods

Expressed in human terms, the above techniques could be shown as the attempt either to cause a baby to evolve from essentially nothing so that it has the capacity to learn and adapt to its environment or to create a baby that has the capacity to learn and adapt to its environment. Of course, this is an extreme example, but the origin of human life is an interesting and fundamentally important question when one is trying to design complex AMRs. The viability of these approaches shall be investigated in this thesis.

1.4 Goals of Thesis

The goal of this thesis is to develop the technology necessary for autonomous mobile robots to explore and learn the spatial structure of the environment they find themselves in and to navigate robustly within this environment. The specific aims are to produce the following contributions:

1. Quantitative results measuring the performance of various control systems.
2. Analysis of the evolutionary method for building robust reactive control systems for AMRs.
3. Techniques for constructing topological maps of the AMR's environment.
4. Techniques for self-localisation within these environments.

Included at the back of this thesis is a disk containing the source code developed for this work.

Chapter 2

2. Background and Related Work

2.1 Robotic Control Systems

2.1.1 Deliberative or Rationalist View

Traditional approaches to artificial intelligence have tended to focus on abstract tasks that more easily lend themselves to a high-level, symbolic formulation, especially problems related to what have been considered the uniquely human aspects of intelligence - consciousness, language, and planning. This deliberative or rationalist view assumes that reasoning has largely supplanted reactive behaviour commonly found in animals. Yet no explanation is offered for how this conversion from reaction to reasoning might have occurred.

With this deliberative view, planning for a task can be functionally decomposed into three main modules: perception, reasoning, and execution [Meeden 94]. First, the task

is perceived according to the initial and goal states, then a solution is fully reasoned out using a search through the appropriate problem space, and finally the computed actions are executed. The emphasis in this deliberative view is firmly on the reasoning phase of this process as shown in Figure 2-1. Perception and action are seen as auxiliary functions.

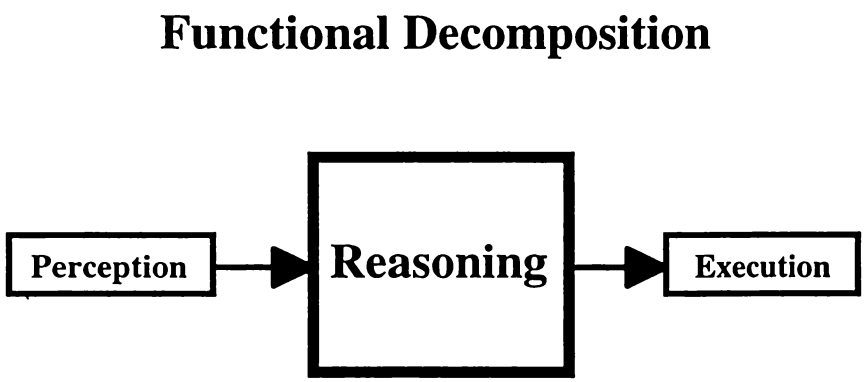

Figure 2-1 The deliberative view's decomposition of behaviour. Typically, deliberative systems are not connected to any external environment, but operate on internal models. Therefore no input from sensors into Perception or output from Execution to actuators is shown. The Reasoning module encompasses the primary computational effort.

Because deliberative processing models plan by searching through problem spaces, they have been quite successful when applied to problems with the following features:

1. The domain can be described with discrete states.
2. Tasks in the domain can be defined by an initial state and a goal state.
3. Given the current state and an operator, the next state can be uniquely determined.

However, when looking at the problem of path-planning for an AMR, this view of reasoning becomes inadequate. The deliberative view is essentially internal - both planning and execution occur completely within the scope of the system. Thus they do not involve direct execution in a real external environment and they safely ignore many of the issues inherent in such environments.

The eight puzzle is an example of a problem that can be solved by this deliberative view. Figure 2-2 depicts the pertinent problem space. There are eight numbered tiles and one blank location configured on a 3×3 board. The operators are tile movements

into the blank location (either left, right, up or down). The object of the problem is to move the tiles around until the goal state is reached. Each state is a board configuration and each tile must be at a discrete location on the board. Puzzle tasks are naturally described in terms of beginning and ending states; and eight puzzle operators produce completely predictable results.

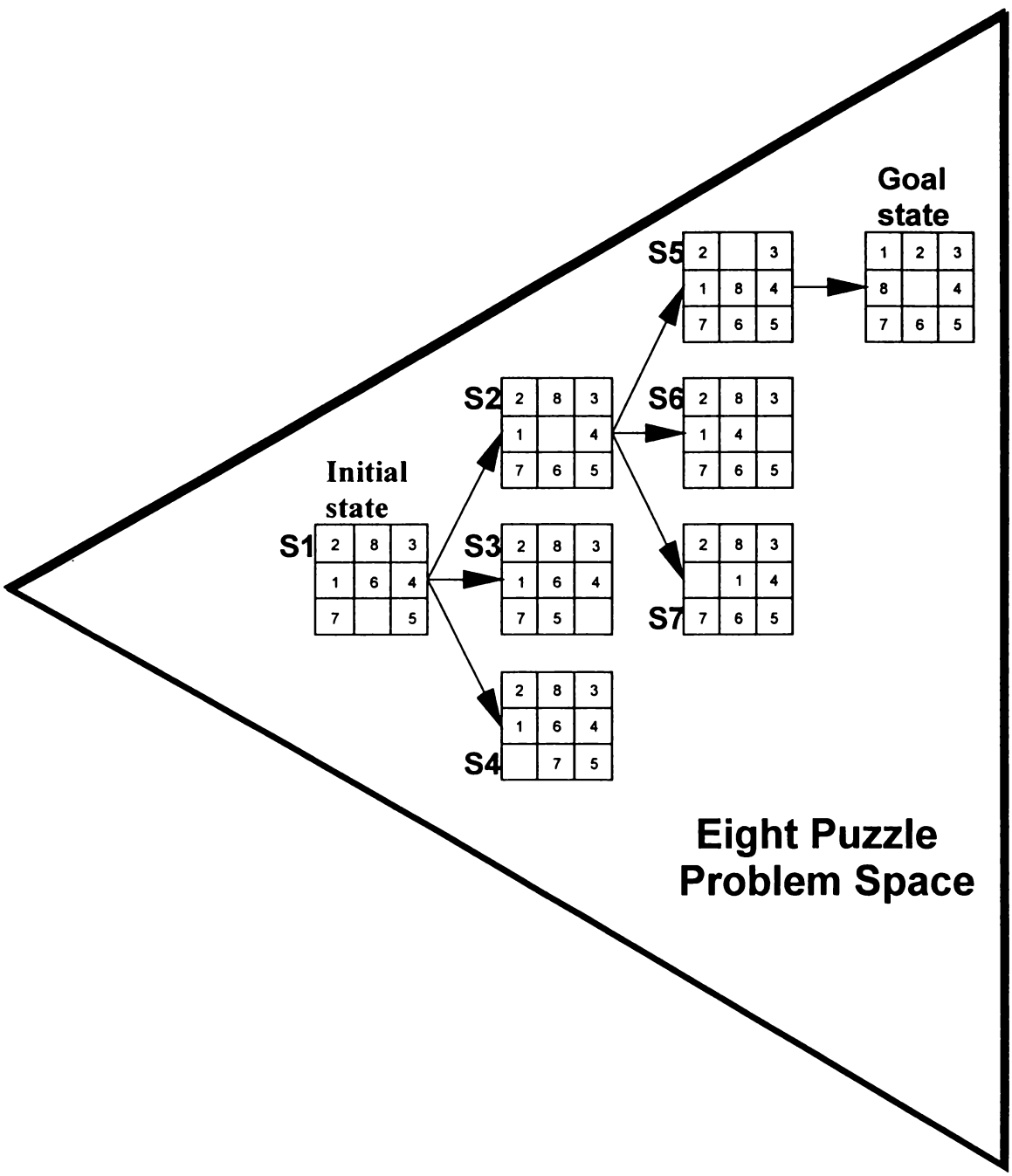

Figure 2-2 The structure of a problem space search for the eight puzzle. There are eight numbered tiles and one blank location configured on a 3×3 board. The operators are tile movements into the blank location (either left, right, up or down). State S5 will eventually lead to the goal.

Owing to these discrete states, the Eight Puzzle results in an easily searchable problem space and the distinction between planning and doing is unimportant. This is because the world of the eight-puzzle is stationary. The computer does not physically push any of the tiles around or interact with the real world in any way.

However, when path-planning for an AMR, the reasoning process cannot be treated independent of the AMR's perception and action. This is because the world in which the AMR is to function will normally be dynamic and there will be noise in both the AMR's perception of the environment and also its action within this environment. These two problems make it impossible to have an accurate knowledge of the problem space and therefore any explicit program of action produced with this deliberative view will quickly become obsolete.

2.1.2 Reactive Control Systems

A reactive control system is characterised by a tight coupling between perception and action with little or no intervening representation. This results in systems which do not perform detailed planning but are able to function in dynamic and complicated environments. 'Pure' reactive control is characterised by a stimulus-response type of relationship with the world. Mentalistic (representational) structures are denounced and the robot reacts to the immediacy of sensory information in a very low-level non-cognitive manner. Complex behaviours emerge as a combination of simple low-level responses to the rich variety of stimuli the world affords. Typically this involves decomposition of tasks into a collection of distributed parallel subtasks. Further, sensor data are normally channelled directly to the individual subtasks, reducing significantly the computational demand typically found in more deliberative navigational regimes which require world model building.

A representative example of this form of navigation is Brooks's subsumption architecture [Brooks 86] which has demonstrated robust navigation for mobile vehicles in dynamically changing domains. It is a layered architecture, well adapted for

hardware implementation [Brooks 86] [Connell 87] and has been used in a wide range of robots, including legged ones [Brooks 89].

These systems accentuate the perception and execution stages of planning while there is negligible reasoning done. The decomposition of behaviour for this reactive view is shown in Figure 2-3. This affords excellent response to dynamic environments but as the complexity of the environment increases they begin to fail for lack of deliberative reasoning.

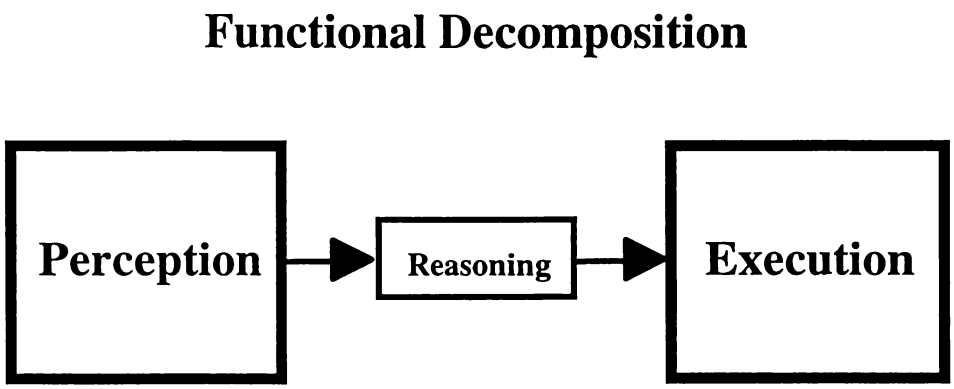

Figure 2-3 The reactive view's decomposition of behaviour. A typical reactive control system is characterised by a tight coupling between perception and execution with little or no internal representation.

As the complexity of the environment in which the AMR must function increases, there is a need for the AMR to be able to construct spatial representations of the environment. This can occur when the AMR comes to a 'dead end' when trying to reach its goal. Some reactive control systems will then resort to a form of wall following behaviour to try to escape from this 'dead end' which is actually a potential minimum in the problem space. However, in a complex environment, there may be several of these potential minima and the AMR can end up oscillating between these geographically different positions. With a pure reactive control system, the AMR will never 'realise' it is traversing between these same states resulting in a failure to reach its goal. These purely reactive control systems suffer from having no memory, and memory is essential for complex environments (refer Chapter Four).

For an AMR such as a micromouse, this is not a serious problem. A mere 256 bytes of memory is capable of adequately representing the AMR's environment. Hence it is

possible to preserve the reactivity of the system because very little time is required to plan through this environment. However, for an AMR in a real world environment, no such simple representation of the environment is possible. If a spatial representation of the environment is to be constructed in a real world environment, it must use a minimal amount of processing time to preserve the reactivity of such a system. Failing this, the resultant control system for the AMR will start to resemble that of traditional AI approaches and suffer from similar problems.

2.2 Spatial Representation of Environments

To navigate in a real world environment, an AMR needs to have a map that provides the spatial structure of the environment. These maps must be able to be created and modified by the AMR itself because most real world environments are dynamic and any map provided for the AMR will eventually become erroneous. Recent research has produced two fundamental paradigms for modelling real world environments: the grid-based (metric) paradigm and the topological paradigm.

2.2.1 Grid-based maps

One of the most widely used non-topological spatial representations is the evidence grid representation developed by Moravec and Elfes [Moravec & Elfes 85]. Evidence grids (also known as certainty grids, probability grids, and occupancy grids) represent space as a two-dimensional or three-dimensional grid of cells, each of which has an associated occupancy probability. Since this geometry corresponds directly to the geometry of the environment, the AMR's position within its model can be determined by its position and orientation in the real world. As a consequence, different positions for which sensors measure the same values (i.e. situations that look alike) are naturally rendered unambiguous in grid-based approaches. However, due to wheel-slippage and noise in sensor data, accurate absolute position information is very difficult to obtain.

To provide an adequate representation of the environment, the resolution of the grid must be fine enough to capture every important detail of the world. This results in an evidence grid which requires a large amount of memory and therefore consequent path-planning is computationally intensive. Evidence grids make poor interfaces for most symbolic problem solvers but do, however, provide the necessary metrical information to facilitate the computation of shortest paths.

2.2.2 Topological Maps

This spatial representation (such as described in Engelson & McDermott 92; Kortenkamp & Weymouth 94; Kuipers & Byun 91; Mataric 94; Pierce & Kuipers 94) represents robot environments by graphs.

Nodes in such graphs correspond to distinct situations, places, or landmarks. They are connected by edges if there exists a direct path between them. As the size of each node is determined by the complexity of the environment, the resulting map is far simpler than a grid-based approach which uses a set resolution small enough to capture every important detail. This permits efficient planning and is a convenient representation for use in symbolic planners, problem solvers and natural language interfaces. The AMR's position within the map is determined by matching distinct situations extracted from sensor data with those in the map. However, determining the position of the AMR relative to a topological map is difficult, especially in situations where two places look alike as both places return similar sensor readings. Also, since sensory input usually depends strongly on the view-point of the AMR, topological approaches may fail to recognise geometrically-nearby places.

These two approaches to robot mapping exhibit orthogonal strengths and weaknesses which are summarised in Table 2-1.

Grid-based approaches	*Topological approaches*
+ easy to build, represent, and maintain	+ permits efficient planning, low space complexity (resolution depends on the complexity of the environment)
+ recognition of places (based on geometry) is non-ambiguous and view point-independent	+ does not require accurate determination of the robot's position
+ facilitates computation of shortest paths	+ convenient representation for symbolic planners, problem solvers, natural language interfaces
- planning inefficient, space-consuming (resolution does not depend on the complexity of the environment)	- difficult to construct and maintain in larger environments
- requires accurate determination of the robot's position	- recognition of places (based on landmarks) often ambiguous, sensitive to the point of view
- poor interface for most symbolic problem solvers	- may yield suboptimal paths

Table 2-1 Comparison of grid-based and topological approaches to map building.

In this thesis, topological maps are generated instead of grid-based maps as they facilitate fast path-planning. However, the maps are grounded metrically in the environment so that similar landmarks can be rendered unambiguous. Techniques to improve the self-localisation ability of the AMR within the topological map are also investigated (refer Chapter Six).

2.3 Approaches to Designing Robotic Control Systems

When one has realised the need for a reactive control system when designing an AMR, the question arises - What form of architecture should be used to implement this reactive control system and how does one go about designing it?

2.3.1 Behaviour-Based Architectures

There are many different architectures currently in use in AMRs around the world. Brooks, a prominent advocate of a behaviour-based approach to robotics, has designed the subsumption architecture, a layered, behaviour-based architecture where the behaviours are finite-state machines (sometimes equipped with timers) which receive inputs directly from the robot's sensors and send outputs directly to the robot's effectors. These behaviours can communicate with each other through simple low-bandwidth channels. In addition, higher-level behaviours can suppress and/or inhibit the communication occurring on channels between lower-level behaviours. Robots using the subsumption architecture have been developed for tasks ranging from obstacle avoidance [Brooks 86] to legged locomotion [Brooks 89] to collecting Coke cans [Connell 87].

Arkin has designed an architecture based on the concept of motor schemas [Arkin 89]. Motor schemas are vector fields where each point in space is associated with a corresponding motor action. For example: a move-to-goal schema might include vectors pointing toward a destination from all points in the environment, while an avoid-obstacle schema might include vectors pointing in the opposite direction from a particular obstacle. These vector fields are added together, and the resulting vector field is used to generate the robot's motion.

Arkin has applied these techniques to a variety of navigation problems and has also built hybrid systems combining motor schemas with high-level symbolic knowledge. This is as a Case-Based Reasoning (CBR) system where various motor-schemas are selected depending on what type of environment the AMR currently perceives itself to be in.

Many other architectures have also been proposed for behaviour-based robotics: Rosenblatt has proposed an architecture that implements behaviours as feedforward neural networks [Rosenblatt & Payton 89]; Maes has developed an architecture

based on interconnected behaviour rules with continuously varying activation levels [Maes 91]. Saffiotti, Ruspini, and Konolige have developed behaviour-based navigation systems using fuzzy control rules [Saffiotti et al. 93]. These architectures differ in the specifics of behaviour selection and arbitration, but share the central concept of distributing the robot's control system over a set of simple reactive behaviours that operate in parallel.

2.3.2 Hybrid Spatial Learning / Reactive Architectures

To facilitate navigation in complex real world environments researchers have designed more complex architectures that integrate a spatial learning system with a reactive controller.

Kuipers and Byun developed one of the first systems for autonomously exploring and building topological maps [Kuipers & Byun 91]. Their system, NX, is based upon locally distinctive places and paths, which serve as nodes and arcs in a topological map. A distinctive place is defined as a point that maximises the number of equidistant obstacles. Once the robot recognises that it is in the neighbourhood of a distinctive place, it applies a hill-climbing algorithm to move to the point where some distinctiveness measure has its local maximum value. It then adds a node for the place to the topological map. This requires NX to navigate through its environment by moving to each distinctive place in the map (which is somewhat limiting).

Mataric has developed a spatial learning system for a mobile robot, Toto, using the subsumption architecture [Mataric 92]. Toto incorporates a behaviour-based reactive controller with behaviours for wandering, avoiding obstacles, following walls, and recognising landmarks. Toto is able to recognise a fixed set of landmarks using sonar sensors: corridors, left walls, right walls, and junk (no landmark). Each detected landmark is added as a node in a topological map with links indicating adjacency. However, the use of a limited fixed set of landmarks results in a robot only being able to function in certain environments.

Kortenkamp has developed a system, RPLAN, based on a cognitive model of spatial learning [Kortenkamp & Weymouth 94]. RPLAN uses a combination of sonar and vision sensing to learn and recognise places that correspond to gateways. These are places that mark the transition between one space in the environment and another space (e.g. doorways and hallway intersections).

RPLAN does not explore autonomously. Instead, it must be lead through the environment by a trainer who takes the robot to all of the places in the environment to which the robot may want to navigate. As the robot visits each gateway, it constructs a local representation of the gateway. The representation includes both edges that are detected using vision, and walls that are detected using sonar and matched against a predetermined set of possible gateway types. A simple Bayesian network [Pearl 88] is then used for place recognition utilising the two sets of data. This system is limited in that it does not explore autonomously and that its learning algorithm again will only function in certain environments.

Yamauchi and Beer have developed ELDEN (Exploration and Learning in Dynamic Environments) which is an integrated system that combines adaptive place networks with a reactive, behaviour-based controller for dealing with transient changes and a relocalisation system for correcting dead reckoning error [Yamauchi & Beer 96]. Their approach adds places into a topological map whenever the robot's distance to previously defined places exceeds a certain threshold. Each link between these places has a variable confidence level and whenever a link is traversed successfully the confidence of the link is increased. However, if the link was not traversed successfully within a given time, the new link confidence is decreased, and if the link confidence drops below a certain threshold, the link is removed from the map. To correct for dead reckoning error, the relocalisation system requires that the robot return to its initial starting position, which imposes limitations on the scalability of the approach.

The mobile robot RHINO, developed at the University of Bonn, uses grid-based maps to learn the spatial structure of the environment [Thrun et al. 98a][Thrun & Bucken 96]. RHINO's navigation system consists of two modules: A global planner [Thrun 93], and a reactive collision avoidance module [Fox et al. 97]. To facilitate fast path-planning,

topological maps are built on top of the grid-based maps. This is achieved by partitioning the grid-based map into a small number or regions, separated by *critical lines*. Critical lines correspond to narrow passages such as doorways. The partitioned map is then mapped into an isomorphic graph. A critical line is equivalent to the distinguishable feature used in Section 6.2.1; however, the regions between these critical lines define the nodes in the isomorphic graph whereas the distinguishable features define the nodes in the topological map in Section 6.2.

RHINO has been deployed very successfully as an interactive museum tour-guide robot [Thrun et al. 98b]. To achieve this, the robot's overall software architecture consists of approximately 25 independent modules (processes), which are executed in parallel on three on-board PCs and three off-board SUN workstations, connected via a tetherless Ethernet bridge.

The control system developed in Chapter Six is yet another form of these hybrid spatial learning / reactive architectures. It also creates topological maps of its environment, however, it does so without the need of first generating a grid-based map which is a space consuming technique. This results in a simpler representation of the AMR's environment that can be manipulated with a low-power processor allowing small low-cost AMRs to be developed.

2.3.3 Design

All these various architectures are merely different ways of encoding a transfer function that maps the sensory input to the actuators of the robot. Yet somehow they must be designed to provide robust control of the AMR. There are only two ways to solve the problem:

1. Hand-design the system and perhaps use some optimisation techniques to maximise the performance of the resultant architecture.

2. Try to evolve the architecture according to Darwinian evolution.

Advocates of evolutionary robotics expect their techniques to supersede a design by hand approach [Harvey et al. 92]. This is due to the fact that the AMR's interaction with a real world environment is very complex and hence very difficult to understand sufficiently and devise an appropriate architecture to yield robust behaviour. Natural evolution is claimed as the existence proof for the viability of this approach. However, is not natural evolution still a hypothesis? Granted there are small adaptations within species but the evolution of mankind is hardly an irrefutable fact.

A genetic algorithm is one such evolutionary technique that models Darwinian evolution. It is a stochastic search technique [Koza 92] that involves transforming a set (population) of individual mathematical objects (usually fixed length character or binary strings), each with an associated fitness value, into a new population (next generation) using operations patterned after the Darwinian principle of reproduction and survival of the fittest and after naturally occurring genetic operations (sexual recombination, mutation). Each mathematical object contains "genetic" information that describes the individual. In biology, chromosomes consist of DNA which constitute the genetic make-up of the individual. This is referred to as the genotype of the individual. The phenotype, however, is the genetic code (normally a subset of the genotype) that determines the physical expression of the individual.

The artificial evolution approach could be applied to various control architectures but is perhaps best suited to neural network architectures which are very easy to represent as a population of viable genotypes. The mathematical objects (chromosomes) provide the coding for the control system and are inter-bred and mutated according to selection pressure. This pressure is controlled by a task-oriented evaluation function: the better the robot performs its task, the more evolutionally-favoured is its control system. However, the process at the inter-breeding and mutation stage is random and when dealing with complex systems the resultant domain space is enormous. This is studied in Chapter Five which shows that for an AMR which must function in a real world environment, the domain space is so great that a random search of it is not a viable process. So far, evolutionary techniques have only been applied to small problems which have yielded encouraging results, nevertheless this technique does not scale well and this author believes you need more faith to believe in evolution than in a Creator.

If Darwinian evolution is incorrect and the process is not viable for an AMR functioning in a real world environment, then the only other way to design a control system is by hand. This is a creationist approach and while techniques can be used to optimise the resultant architecture, a very deep understanding of the overall problem of robust control within complex environments is required. To an extent this can be aided by studying human/animal behaviour and anatomy. Apart from this, all that is left is a creative act on the part of the designer - which is to be greatly admired, though impossible to formalise.

Chapter 3

3. Simulator

3.1 Overview

To design and analyse robotic control systems two essential elements are required: an AMR in which the control system is embedded and an environment in which the AMR is to function. For the purpose of this research, a simulator was designed to provide both of these requirements. The benefits of using a simulator to perform these experiments are:

- the time domain can be accelerated and therefore permit many simulations with many different 'worlds.'
- practical problems such as flat batteries and faulty circuit boards can be avoided.

However, for these simulations to be of any great use they must adequately model the AMR and its environment. A simulator that is not realistic is not adequate for research in this domain, because the AMR's perception and action within its environment are

fundamentally important to the problem. If the simulator does not adequately model this, then the transfer function that is being developed for robust behaviour will bear little resemblance to the required transfer function for the physical AMR.

3.2 Environment

The environment of the AMR for these simulations was chosen to be a room of approximately six metres square. Within this room, the AMR has to travel from one point to another, usually to opposite sides of the room. The obstacles in the room were provided by generating circular objects of various diameters that could overlap each other. Previous work done in this field [Ram et al. 92] uses environments that are grouped by their obstacle density. A similar approach was adopted and provided a variety of complex worlds through which the AMR could navigate.

Figure 3-1 shows two typical 'worlds' at twenty-five percent obstacle density. The best paths (shown in magenta) are only approximate and were determined using an algorithm variously known as Bellman's algorithm or the flooding algorithm [Brightwell 93]. The size of the environments was scaleable, however, this feature was not needed for the experiments that were performed as the shown resolution provided adequately complex worlds (i.e. worlds with 'bottlenecks' and 'dead-ends') for the AMR to navigate through.

Each world was generated randomly with controls on the following factors:

1. The obstacle density could be defined.
2. A small hemisphere around the start and goal positions was kept free of obstacles.
3. A navigable path from start to goal had to exist in the generated world.

With these controls, one thousand worlds which were used in later simulations were generated at each of 15%, 20%, 25%, 30% and 35% obstacle densities.

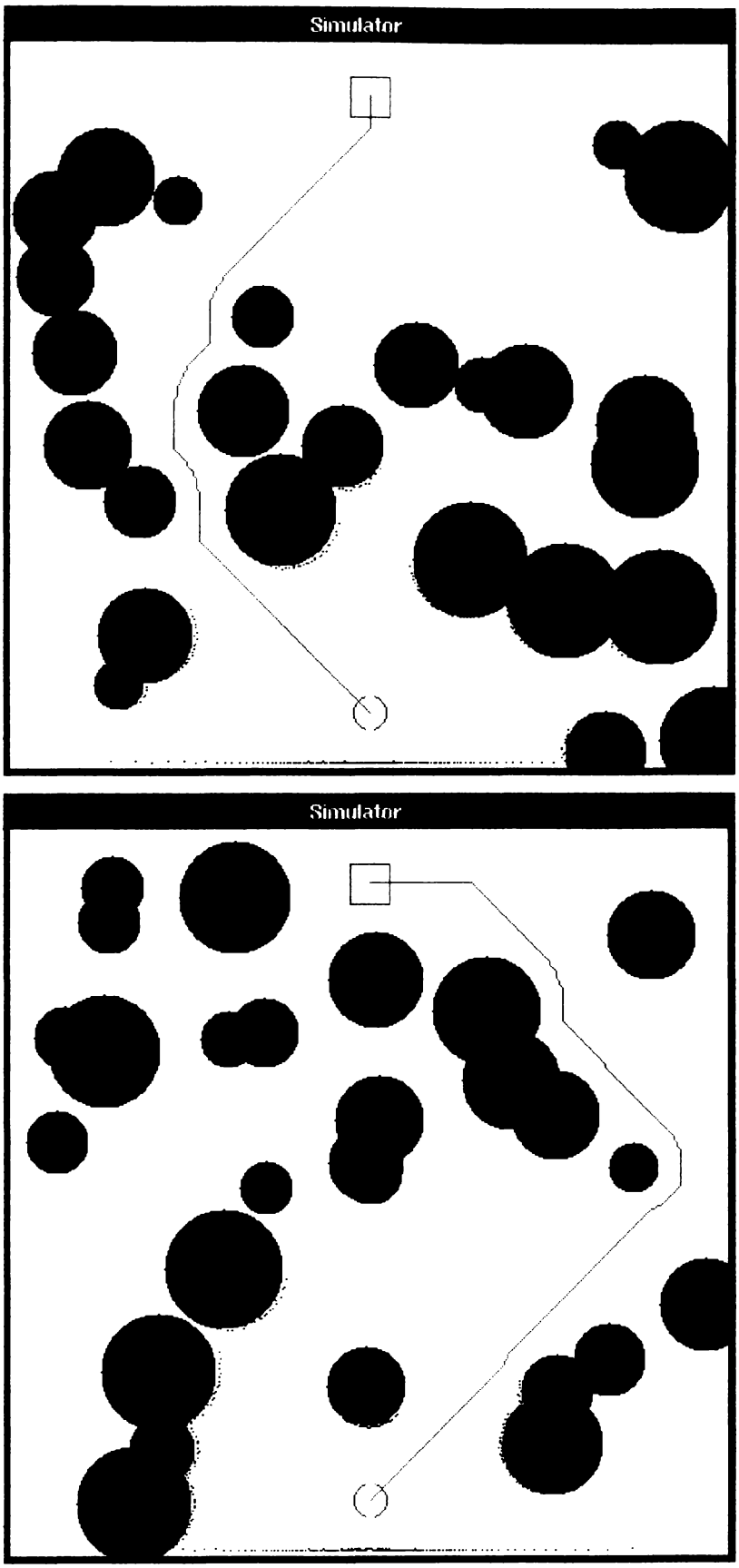

Figure 3-1 Two different simulated 'worlds' at 25% obstacle density. The robot can be seen at the bottom of each world. The square box is the goal and the line connecting them shows approximately the best path.

3.3 Autonomous Mobile Robot

To adequately model the AMR for the simulator, the drive train and sensors used must be specified. A commonly used micromouse chassis was used in a scaled up form. This type of chassis, termed a 'wheel chair configuration,' allows the AMR to turn on the 'spot' and is shown in Figure 3-2. For the experiments performed, the size of the AMR used was thirty centimetres in diameter, a small practical size for an indoor environment.

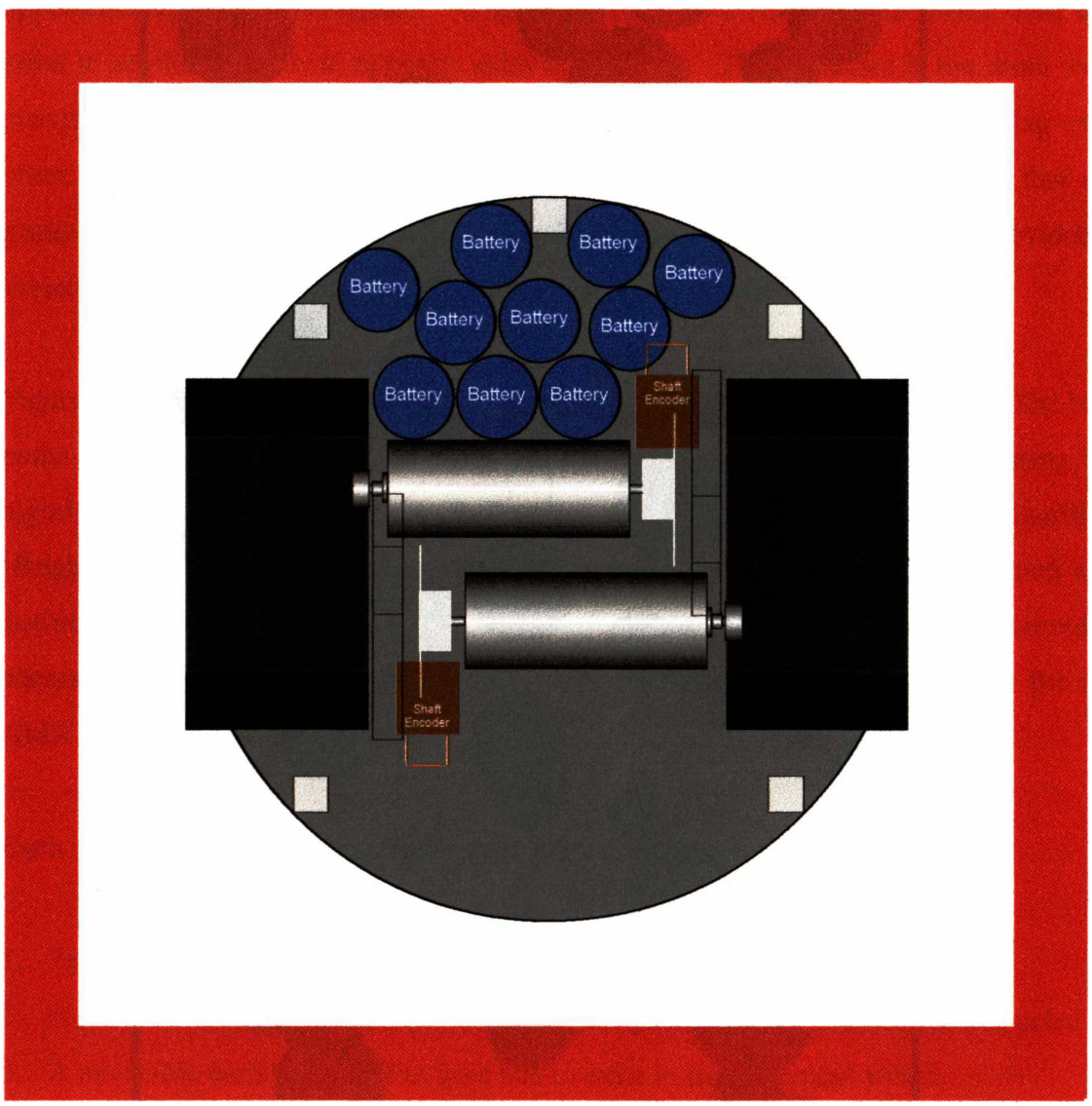

Figure 3-2 Diagram of a micromouse showing 'wheel chair chassis.' The two drive wheels are placed such that the centre of rotation is in the middle of the chassis allowing the mouse to turn on the 'spot.'

3.3.1 Obstacle Sensors

Instead of five infra red sensors used in a micromouse to detect walls in the maze, a 360 degree laser range-finder was simulated. This form of sensing, which is feasible for current AMR designs, provides far more information about the spatial structure of the AMR's environment. To keep the simulations as realistic as possible, the data provided from this simulated sensor was an array of 360 bytes of range data. The range of the laser range-finder was limited to 250 centimetres and the sample rate variable. To acquire the array of range data for whatever position the AMR was currently in, an algorithm was developed to extract the visible curvature of each of the obstacles within range as well as detecting the walls of the room. Figure 3-3 shows how the perceived width of the obstacle varies as a function of the distance from it.

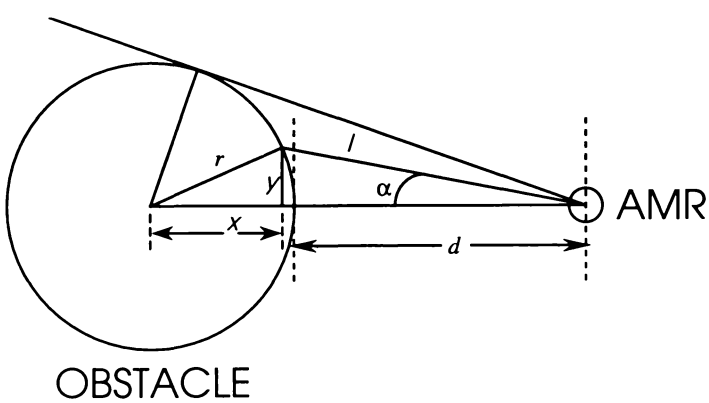

Figure 3-3 Diagram showing the visible width of an obstacle from the AMR's viewpoint.

As the laser range-finder sweeps over the obstacle, the range profile of the surface relative to the AMR is given by the following equations:

Equation 3-1

$$x = d + r - \cos^2[\alpha]\left(d + r - \sqrt{r^2 - \tan^2[\alpha](d^2 + 2dr)}\right)$$

Equation 3-2

$$y = \sqrt{-x^2 + r^2}$$

Equation 3-3

$$l = \sqrt{(d + r - x)^2 + y^2}$$

The resulting sensory input for the AMR is shown graphically in Figure 3-4. The resolution of the data stored in the array is eight bits, however, with errors generated from approximating Equation 3-1, the error free resolution is only 7 bits. This provides a fairly coarse spatial representation of the environment but is well within the capabilities of current laser range-finders.

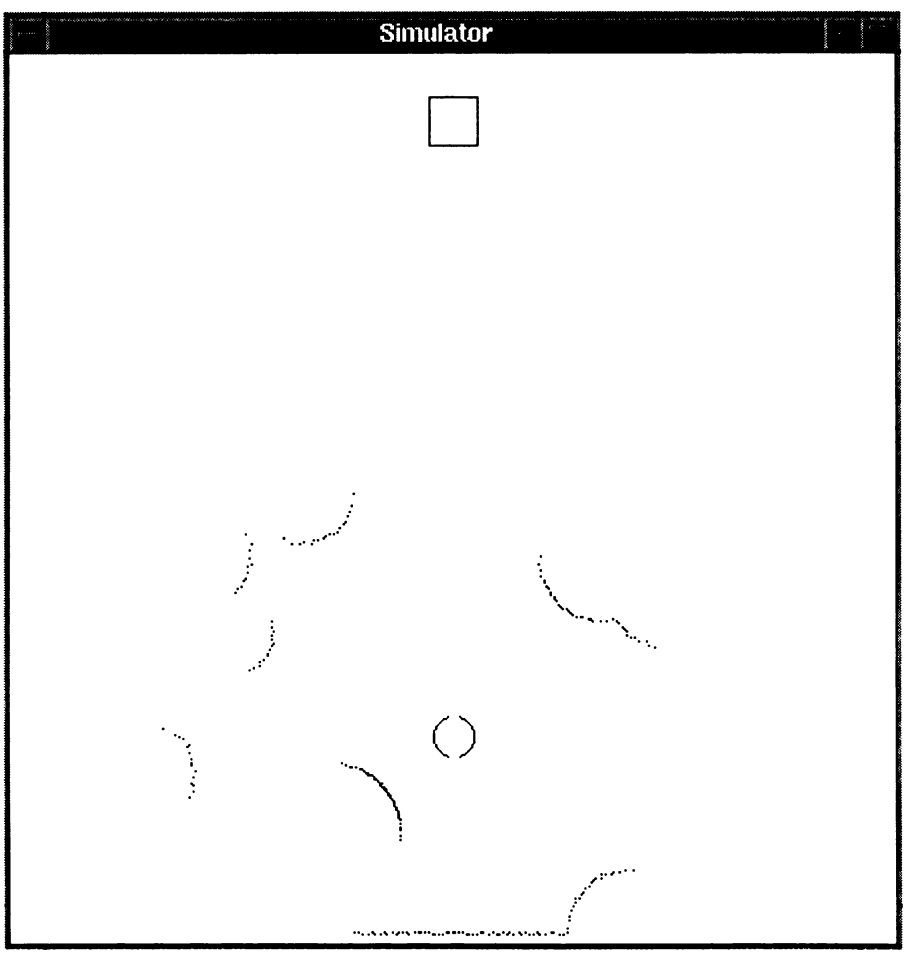

Figure 3-4 360 degree laser range-finder scan graphically showing the realistic data used in simulations.

3.3.2 Motion Sensors

Shaft encoders are commonly used on AMRs to provide positional information. A 'wheel chair' chassis configuration typically has two shaft encoders, one for each wheel. On a micromouse, these shaft encoders typically provide a resolution of twenty counts per millimetre. This may seem a fine resolution, however, due to wheel-slippage, the absolute positional accuracy over large distances is poor. They do, however, provide good velocity information. For simulation purposes, position (accurate over a short period), velocity and acceleration data are available to the control system of the AMR as this is certainly realistic in a physical robot.

Absolute bearing and rate of turn information are the other sensory inputs used in the simulator. This information could be provided from a digital compass and a rate gyro which together could supply accurate bearing and rate of turn data with no absolute drift. Figure 3-5 shows the input/output characteristics of the control system of the simulated AMR.

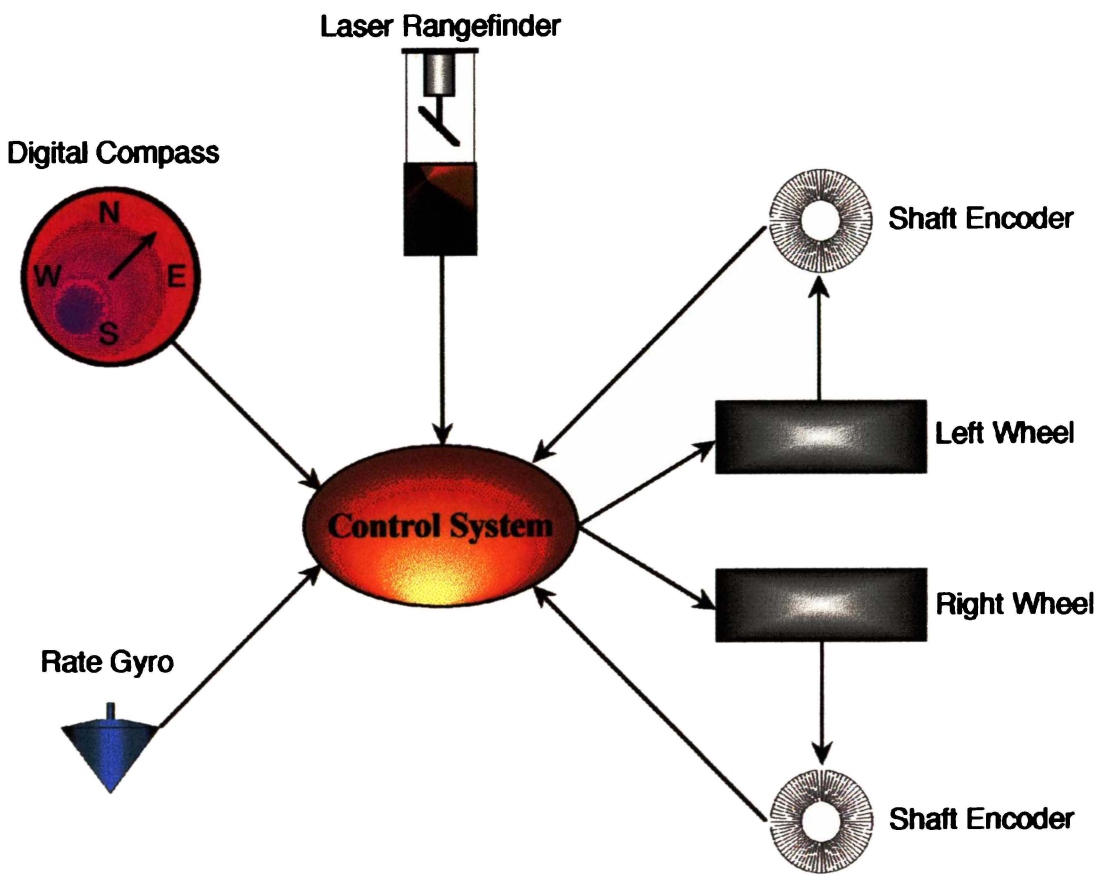

Figure 3-5 displays the sensory inputs and actuator outputs of the AMR's control system.

3.3.3 Kinematics

In Figure 3-5, the outputs of the control system are two forces applied to the wheels. With these outputs, the control system can accelerate, decelerate and rotate the AMR. This is however, the lowest-level output possible and therefore to perform even the simplest of motions, such as to rotate ninety degrees, would require several actions.

To simplify the transfer function of the control system, the kinematics of the AMR were resolved and encoded such that the output required from the control system was a bearing that represented the desired heading of the robot. This would result in the AMR altering its heading and travelling at the maximum velocity possible given the surrounding obstacles. The controller is realistic in that it limits the maximum acceleration possible for the forward and rotational motion of the AMR given its 'wheel chair' chassis design. Limiting the maximum acceleration that the AMR's tyres can withstand imposes a limit on the rate of turn. This can be shown by

Equation 3-4

$$r_{arc} = \frac{r_{robot} \cdot v_{robot}}{dwv}$$

r_{arc} = radius of arc generated due to dwv

r_{robot} = radius of robot

v_{robot} = robot velocity : average of both wheels

dwv = differential wheel velocity

Equation 3-5

$$\Delta h = \frac{v_{robot}}{r_{arc}} = \frac{dwv}{r_{robot}}$$

Δh = rate of turn

Equation 3-6

$$a_c = \frac{dwv.v_{robot}}{r_{robot}}$$

a_c = centripetal acceleration

Equation 3-7

$$\Delta h < \frac{a_c \max}{v_{robot}}$$

The maximum centripetal acceleration permitted in these experiments was 2.5m/s^2. The maximum linear acceleration permitted was 1.7m/s^2. These values were selected to avoid 'wheel spin' and are well within the performance of such a chassis design. 'Junior,' the fastest micromouse at the University of Waikato, can withstand a centripetal acceleration of 15m/s^2! Figure 3-6 shows the input/output characteristics of the modified control system of the simulated AMR.

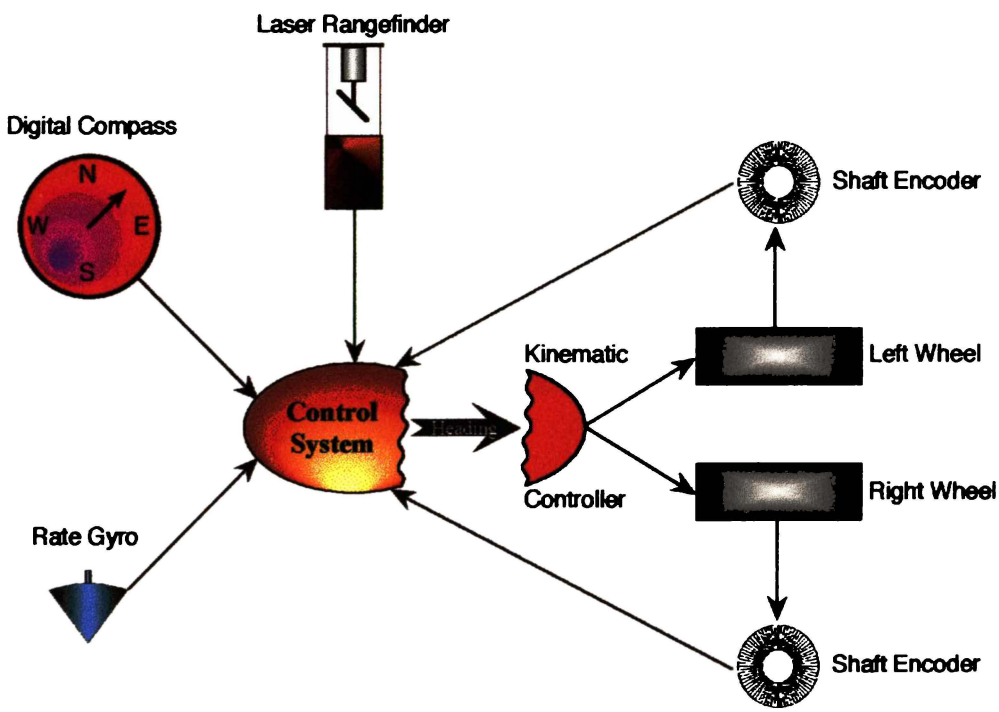

Figure 3-6 displays the sensory inputs and actuator outputs of the AMR's modified control system.

Chapter 4

4. Experiments

4.1 Pure Reactive Behaviour

The first control system evaluated with the simulator is a purely reactive control system. 'Pure' reactive control is characterised by a stimulus-response type of relationship with the world. Hence this form of controller is characterised by a tight coupling between perception and action with no intervening representation. The algorithms used in this controller require no 'memory' and can be implemented with minimal processing overheads.

Following a similar scheme to that used in earlier work [Ram et al. 92], the inputs to this system are a vector to the AMR's goal, a vector pointing in the opposite direction of any obstacles, and a random noise vector. However, the obstacle vector is determined by summing each range vector in a 360 degree scan of the laser range-finder with a simple algorithm. This controller is determined in Equation 4-1 below.

Equation 4-1

$$Heading = GG \times G_v + NG \times N_v + OG \times \sum_{\alpha=1}^{360} \begin{cases} 0 & \text{for } r_\alpha \geq S \\ \dfrac{r_\alpha}{r_\alpha - r_{Robot}} & \text{for } S > r_\alpha > r_{Robot} \\ r_\alpha & \text{for } r_\alpha \leq r_{Robot} \end{cases}$$

GG = Goal gain
G_v = Goal vector
NG = Noise gain
N_v = Noise vector
OG = Obstacle gain
α = bearing (degrees)
r_α = range to obstacle at bearing α
S = maximum sensing distance
r_{Robot} = radius of robot

4.1.1 Control Parameters

This control system can be characterised by four variables. They are described as follows:

- Goal Gain (GG) The magnitude of the attraction vector between the AMR and its final objective, the goal.

- Obstacle Gain (OG) The magnitude of the repulsion vector between the AMR and the surrounding obstacles it can perceive.

- Noise Gain (NG) The magnitude of a randomly generated vector that is used to introduce noise into the control system.

- Noise Persistence (NP) The length of time that the randomly generated noise vector is used before a new vector is generated.

4.1.2 Performance

The control system was evaluated systematically over a range of varying obstacle gains and noise gains. These experiments were performed using a library of 500 generated worlds. They were evaluated using 500 worlds from each class of obstacle densities (15%, 20%, 25%, 30% and 35%). A goal gain of six and a noise persistence of 2.5 seconds were used for all the experiments. A maximum time of 200 seconds (simulated time i.e. 20,000 control cycles) was allowed for the AMR to solve a world. If, after 200 seconds the AMR had not reached the goal, the world was recorded as being unsolvable with this particular controller. The efficiency of the solved worlds was determined by the following equation.

Equation 4-2

$$\text{Efficiency} = \frac{\text{optimal path length}}{\text{actual path length}}$$

The results are shown in Tables 4-1 through 4-5.

	15% Obstacle Density											
	Percentage of Worlds Solved				Percentage Efficiency of Solved Worlds				Average Time (sec)			
NG	OG=3	OG=5	OG=7	OG=9	OG=3	OG=5	OG=7	OG=9	OG=3	OG=5	OG=7	OG=9
1	77	79	79	76	97	95	94	92	14.5	13.1	13.3	13.2
2	80	84	84	84	94	91	88	87	15.7	16.3	17.9	17.6
3	86	87	88	86	89	86	83	81	19.8	17.7	19.2	20.1
4	91	91	93	92	80	78	76	73	23.9	21.9	23.4	23.4
5	93	94	94	94	70	68	65	63	28.6	27.3	28.3	31.6

Table 4-1 Performance of pure reactive control system in 15% obstacle density environment.

	20% Obstacle Density											
	Percentage of Worlds Solved				Percentage Efficiency of Solved Worlds				Average Time (sec)			
NG	OG=3	OG=5	OG=7	OG=9	OG=3	OG=5	OG=7	OG=9	OG=3	OG=5	OG=7	OG=9
1	61	62	65	63	96	94	91	90	17.7	16.3	17.9	17.1
2	64	68	71	70	93	89	87	84	20.7	19.3	18.6	21.9
3	71	73	74	77	86	84	80	77	27.8	22.3	24.4	25.9
4	80	80	81	80	79	75	73	69	31.0	28.8	29.3	31.4
5	85	83	85	85	67	64	64	60	37.4	37.1	34.5	36.8

Table 4-2 Performance of pure reactive control system in 20% obstacle density environment.

	25% Obstacle Density											
	Percentage of Worlds Solved				Percentage Efficiency of Solved Worlds				Average Time (sec)			
NG	OG=3	OG=5	OG=7	OG=9	OG=3	OG=5	OG=7	OG=9	OG=3	OG=5	OG=7	OG=9
1	45	49	46	46	96	93	91	89	21.9	20.5	18.3	20.0
2	50	52	54	52	91	89	86	83	27.6	22.6	24.9	24.7
3	57	58	59	57	87	82	79	76	29.8	28.2	29.8	31.0
4	63	66	63	62	79	73	71	68	34.7	34.3	35.3	35.6
5	73	72	71	70	68	61	60	58	44.2	48.7	44.4	44.8

Table 4-3 Performance of pure reactive control system in 25% obstacle density environment.

	30% Obstacle Density											
	Percentage of Worlds Solved				Percentage Efficiency of Solved Worlds				Average Time (sec)			
NG	OG=3	OG=5	OG=7	OG=9	OG=3	OG=5	OG=7	OG=9	OG=3	OG=5	OG=7	OG=9
1	40	42	41	37	98	93	92	90	21.5	21.2	20.2	21.3
2	46	47	47	43	93	88	85	83	27.9	25.5	27.4	30.1
3	51	51	52	50	86	85	79	75	31.3	27.2	31.2	35.5
4	54	56	57	58	79	73	72	67	37.3	37.9	37.0	42.5
5	64	65	65	60	66	64	59	58	50.1	46.8	49.6	48.5

Table 4-4 Performance of pure reactive control system in 30% obstacle density environment.

	35% Obstacle Density											
	Percentage of Worlds Solved				Percentage Efficiency of Solved Worlds				Average Time (sec)			
NG	OG=3	OG=5	OG=7	OG=9	OG=3	OG=5	OG=7	OG=9	OG=3	OG=5	OG=7	OG=9
1	40	40	40	36	96	94	92	90	24.6	20.2	22.3	25.3
2	45	44	43	40	92	89	87	84	27.4	24.7	26.3	31.0
3	49	51	52	47	86	81	76	76	33.1	35.0	40.1	39.2
4	55	59	58	53	77	71	70	67	42.1	43.7	44.0	46.6
5	61	64	64	58	66	63	60	57	51.7	49.6	53.2	55.6

Table 4-5 Performance of pure reactive control system in 35% obstacle density environment.

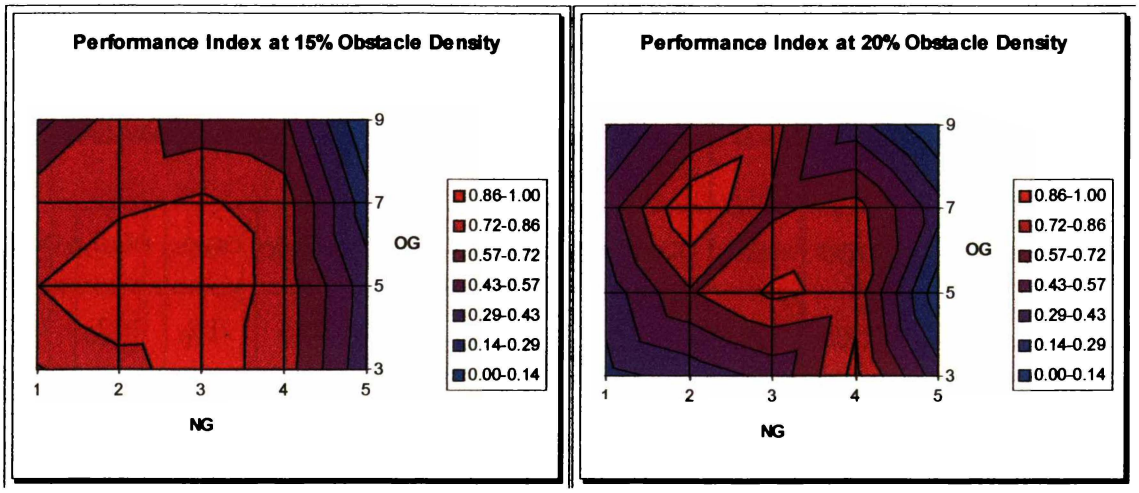

Figure 4-1 Performance characterisation at 15% and 20% obstacle densities.

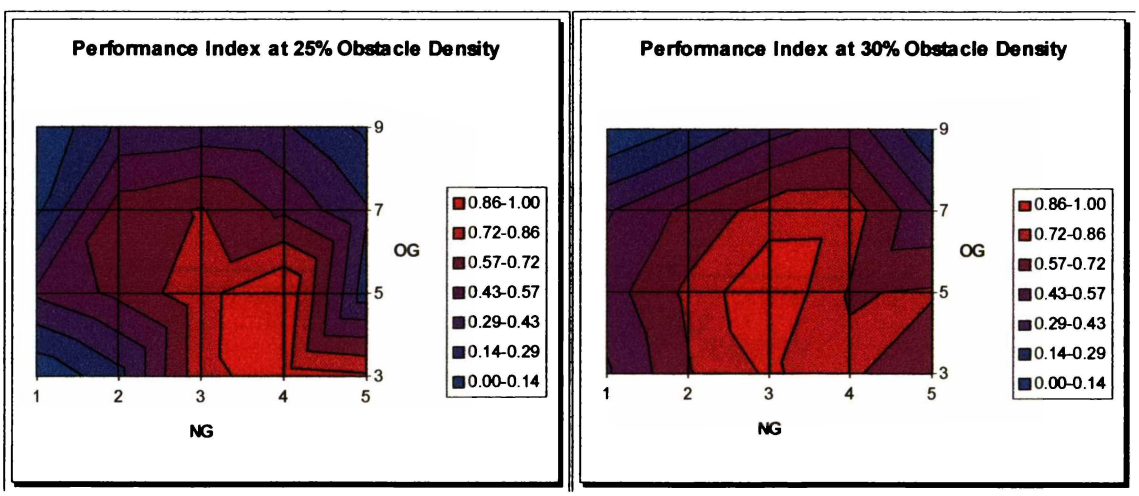

Figure 4-2 Performance characterisation at 25% and 30% obstacle densities.

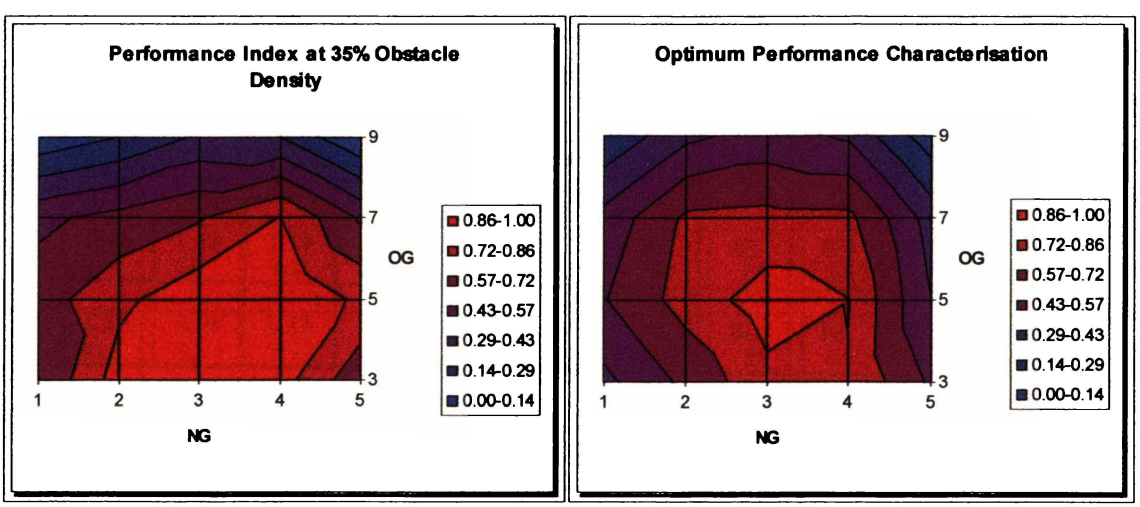

Figure 4-3 Performance characterisation at 35% obstacle density and resulting optimum performance characterisation.

To try to graphically display these data so that an optimum configuration could be found, the function in Equation 4-3 was applied to the data and plotted in Figures 4-1 through 4-3.

Equation 4-3

$$\text{performance index} = \%\,\text{solved\_worlds}^2 \times \%\,\text{efficiency\_solved} \times (100 - \text{average\_time})$$

For the controller to be robust, the AMR must be able to solve every world if there exists a navigable path to the goal. Because this is the most important factor in defining the 'robustness' of a controller, Equation 4-3 defines the performance index used with an emphasis on the ability of the AMR to solve worlds. The resulting optimal performance characterisation is determined by the summation of the normalised performance index for each obstacle density. From this the optimal configuration for the control system was determined and is shown below.

OG = 5 NG = 3 GG = 6 NP = 2.5 (sec)

The overall performance for this configuration is shown in Figure 4-4.

The performance of this controller is not satisfactory and does not produce robust behaviour. In Figure 4-5 the actual trajectories taken by the AMR are shown. The sixth world shows a typical problem that occurs when using such a controller. The AMR finds itself trapped in a position where the attractive force towards the goal is cancelled by the repulsive force of the surrounding obstacles. This situation can be described as a potential minimum in which the AMR travels into and cannot escape. That is primarily the reason for adding noise into the system. By adding a noise vector into the controller, the AMR can often escape from such situations and therefore solve more worlds. However, the 'deeper' the potential minimum, the greater the amount of noise that needs to added and this reduces the efficiency of such a controller.

Using the optimal configuration, Figure 4-5 shows the trajectories of the AMR through the first six worlds. In the sixth world, the AMR fails to reach the goal because it becomes trapped in a potential minimum and there was not sufficient noise to escape.

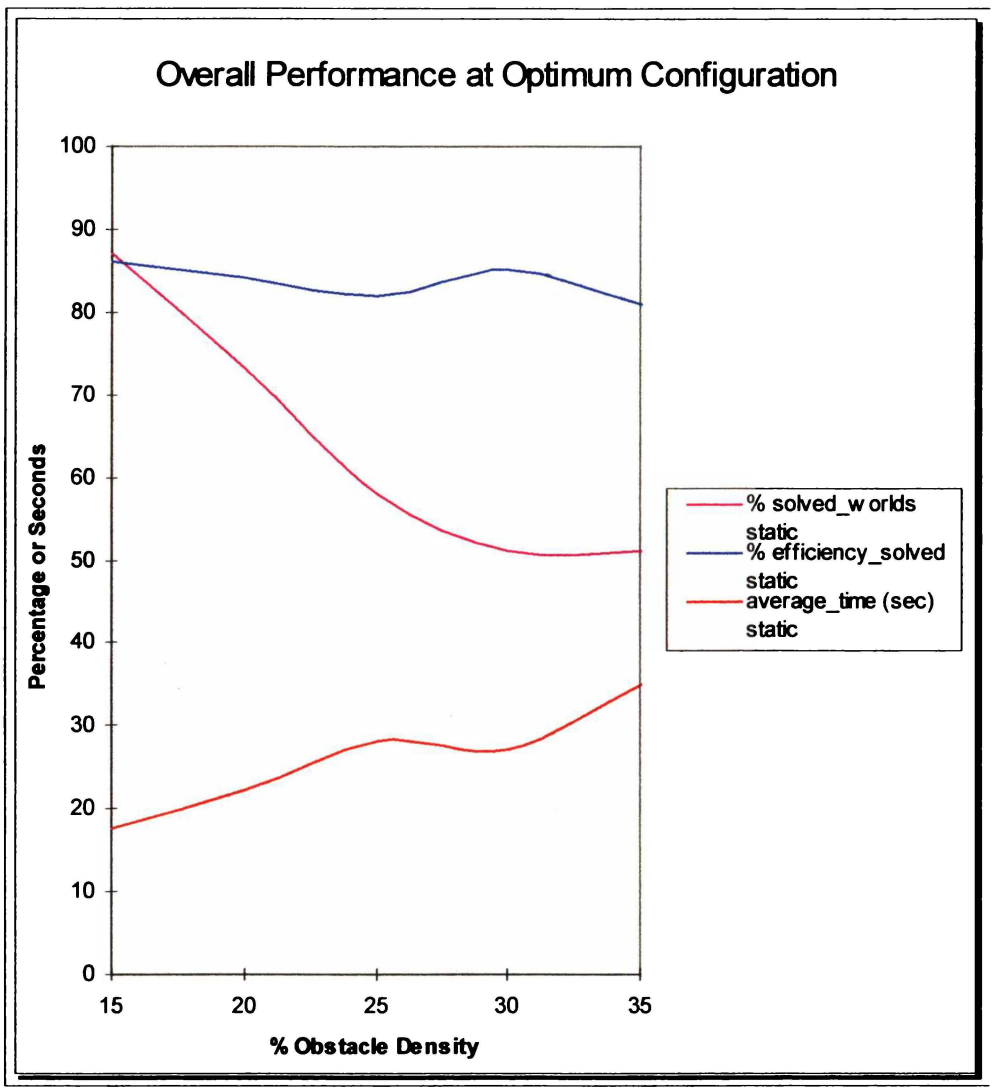

Figure 4-4 Overall performance at optimum configuration.

These potential minima can be shown graphically by determining the resultant output vectors for positions throughout the entire world. A potential minimum is an area where adjacent output vectors are differing by ~180 degrees. Figure 4-6 shows such a

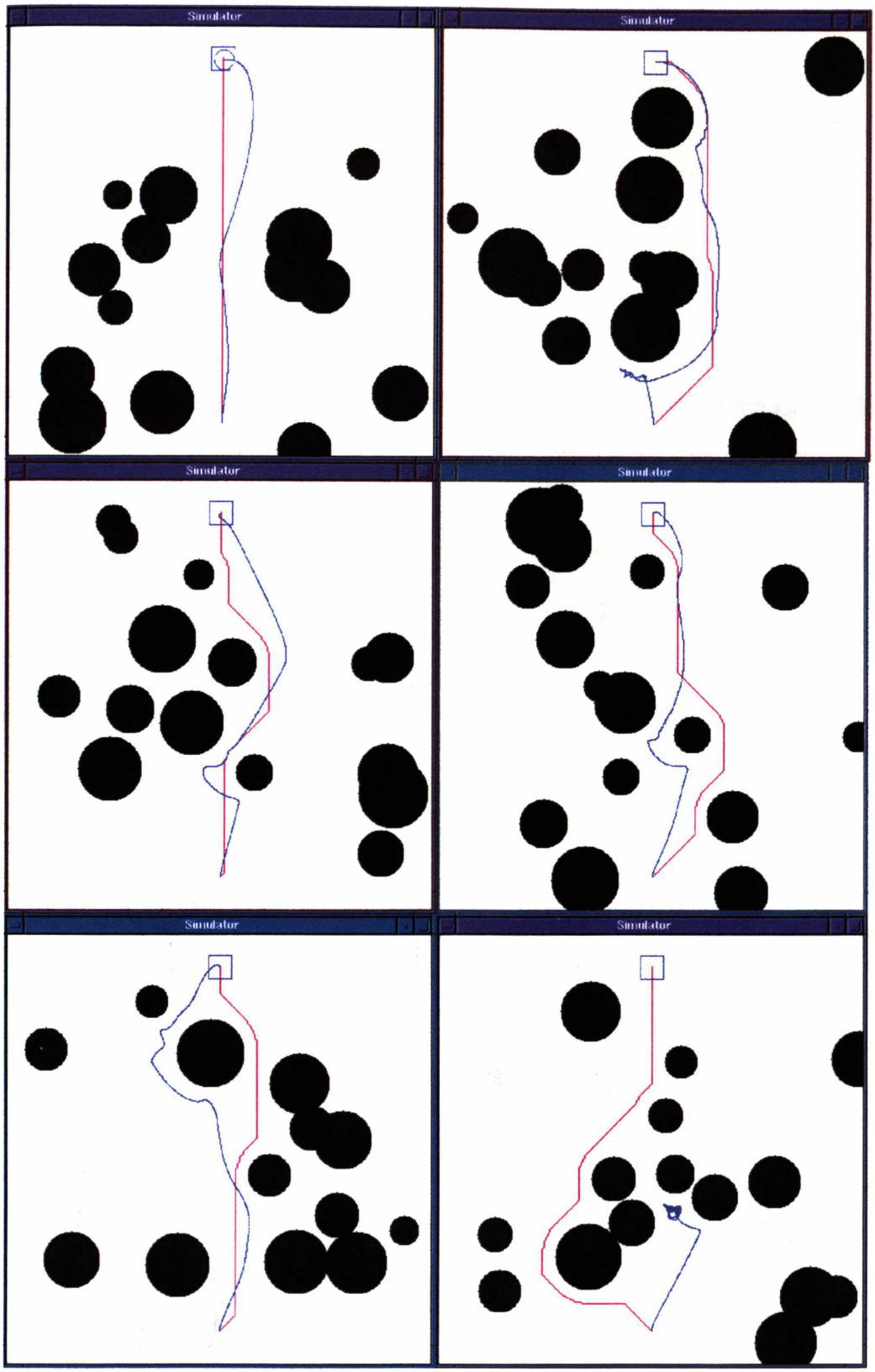

Figure 4-5 Plots of the optimal (magenta) and actual trajectories (blue) to reach the goal for the first six worlds at 15% obstacle density using the optimal configuration for the control system.

calculation for the sixth world in Figure 4-5. The noise gain was reduced to zero to highlight the potential minima. However, as can be seen from Figure 4-6 there is a large area surrounding and pointing towards the potential minimum in which the AMR was trapped. Consequently, even with the added noise vector the AMR could still not escape.

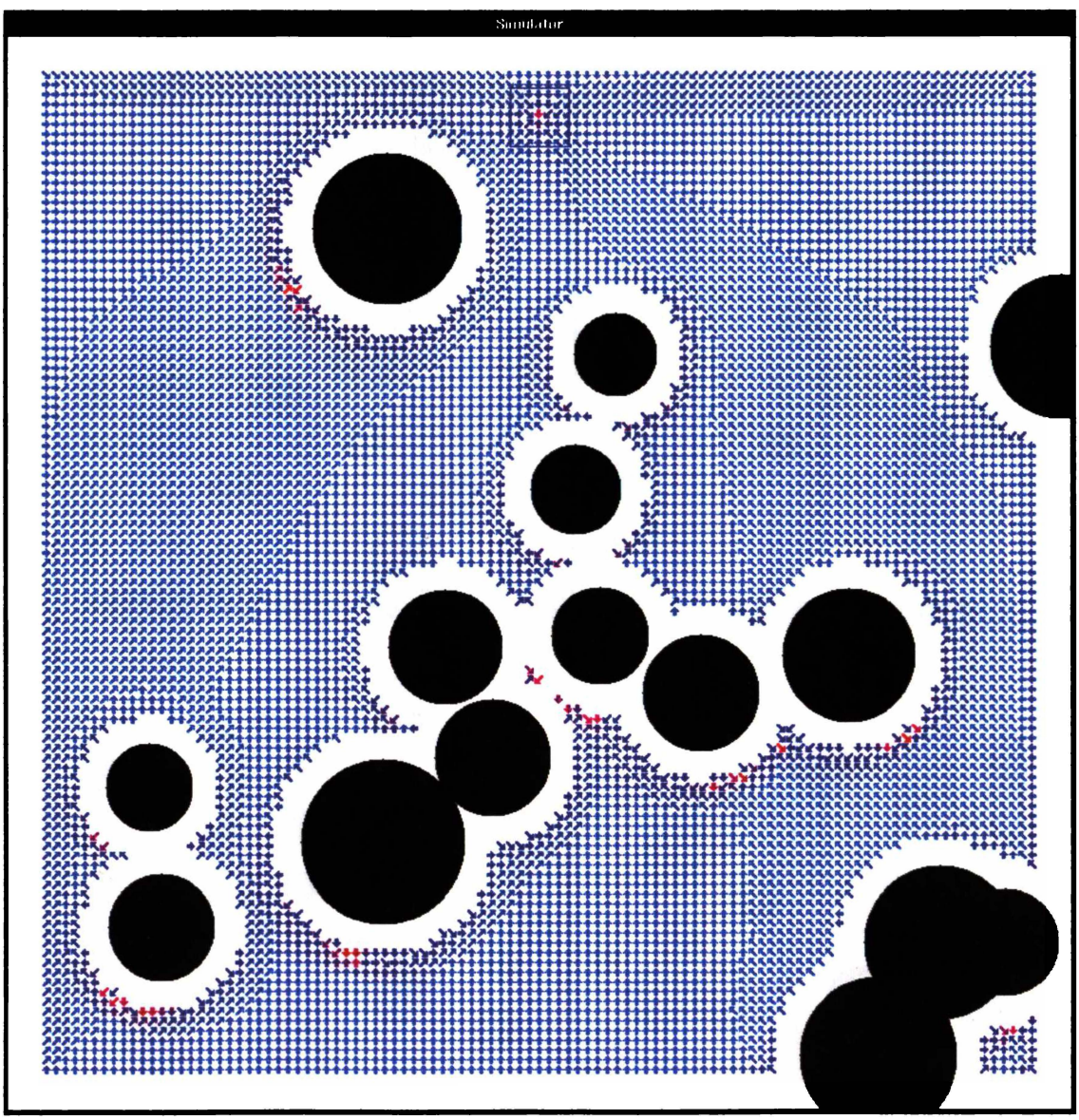

Figure 4-6 A graphical construction for the sixth world of the resultant output vector for each position using the purely reactive controller. Potential minima can be observed from the direction of the vectors and are highlighted by the pinkish arrows.

4.2 Adaptive Reactive Behaviour

The pure reactive controller that was characterised in Section 4.1 has a performance that is far from being deemed robust. While it performs very well in the simplest of worlds, its performance degrades rapidly as anything approaching a 'real world' environment is attempted. There are two obvious reasons for the performance of the pure reactive controller of Section 4.1. One is that the controller is very simple with only three main vectors to determine the behaviour of the AMR, and the other is that the gains of these vectors are static.

To try to improve the performance of the AMR, an *adaptive* reactive controller was designed and characterised, using concepts from earlier work [Ram et al. 92]. This controller has the ability to modify the gains of the various vectors that determine the AMR's resultant behaviour. They are modified depending on the AMR's perception of the external world and its own internal state. This is implemented as a form of case-based reasoning (CBR) where different cases are selected according to the AMR's state and the current environment in which the AMR finds itself situated. Each case has its own set of gains for the controller which yield differing forms of behaviour.

4.2.1 Control Parameters

This new control system is characterised by seven variables, four of which are identical to the controller developed in Section 4.1. They are described as follows:

- Goal Gain (GG) The magnitude of the attraction vector between the AMR and its final objective, the goal.

- Obstacle Gain (OG) The magnitude of the repulsion vector between the AMR and the surrounding obstacles it can perceive.

- Noise Gain (NG) The magnitude of a randomly generated vector that is used to introduce noise into the control system.

- Noise Persistence (NP) The length of time that the randomly generated noise vector is used before a new vector is generated.

- Sensing Distance (S) The maximum sensing range of the AMR. The upper limit is set at 250 (2.5m) but can be reduced to artificially limit the sensing range of the AMR.

- Temporary Goal Gain (TGG) The magnitude of a temporary vector that is generated to provide a wall following type of behaviour.

- Exit Goal Gain (EGG) The magnitude of a vector that is pointing towards a navigable space between two obstacles.

Every behaviour generated with this control system can only use these seven variables. All low level motor control has effectively been abstracted away, permitting simple high-level representations of various types of behaviour which are stored as cases.

4.2.2 Environment Information

To determine which type of behaviour the AMR should adopt at a particular point in its environment there must be some correlation between the local environment and the AMR's case library. However, to preserve the 'reactiveness' of the control system, this information about the environment must be obtainable through perceptual input during the normal reactive control process. The correlation between this information and the AMR's case library must also be able to be computed during the normal reactive control process. With these bounds in place, the following environmental information about the world is maintained:

- Clutter This metric, defined in Equation 4-4, represents how many obstacles (as perceived by the AMR) there are within the sensing range of the AMR.

Equation 4-4

$$\text{Clutter} = \sum_{\alpha=1}^{360} \begin{cases} 1 & \text{for } r_\alpha - r_{\alpha-1} > d \\ 0 & \text{for } r_\alpha - r_{\alpha-1} \le d \end{cases}$$

α = bearing (degrees)
d = diameter of AMR
r_α = range to obstacle at bearing α

- Density This metric, defined in Equation 4-5, represents the amount of occupied space that the AMR can detect within its sensing range.

Equation 4-5

$$\text{Density} = \sum_{\alpha=1}^{360} \begin{cases} 1 \text{ for } r_\alpha < S \\ 0 \text{ for } r_\alpha \ge S \end{cases}$$

α = bearing (degrees)
r_α = range to obstacle at bearing α
S = maximum sensing distance

- Proximity This is a tristate value, defined in Equation 4-6, which represents the proximity of the AMR to obstacles within a field of view either side of its current heading. The field of view is defined as 35 degrees either side of the current heading of the AMR.

Equation 4-6

$$\text{Proximity} = \begin{cases} -1 \text{ for } f \ge \frac{1}{2}S \\ 0 \text{ for } \frac{1}{2}S > f \ge d \\ 1 \text{ for } f < d \end{cases}$$

d = diameter of AMR
S = maximum sensing distance
f = closest range of obstacles within field of view

- Wander This is metric, defined in Equation 4-7, defines how efficient the AMR's current path is. This is determined by examining the ratio of the AMR's current path length to a straight path from the AMR's current position to the start position for the particular case currently in use. Because of the short time between selected cases (see Section 4.2.4) even if the optimal path is around an obstacle, the path will still be 'relatively' straight.

Equation 4-7

$$\text{Wander} = 1 - \left(\frac{l}{p}\right)$$

l = straight path length
p = AMR's current path length

- Clear-to-goal This is a binary value which, if true, indicates that the AMR is facing the goal and there are no obstacles between it and the goal along a straight line path.

- Senses-goal This a binary value which represents whether the goal is within the AMR's sensing range with nothing between the goal and the AMR.

- Goal-nearby This is a binary value which indicates that the AMR senses the goal but there are obstacles between the two.

- Nomovement This is a binary value which indicates that the AMR's velocity has dropped below a certain threshold for a period of time.

Each of the items on the above list is information that is readily available, either directly from the AMR's sensors or from a straightforward mathematical calculation of sensor data. No higher-level processing is required to produce this information.

4.2.3 The Case Library

Work already done in this field by [Ram et al. 92] provided some useful insight into the types of behaviour different environments would call for. From this work and some

experimental analysis, eight different types of behaviour were created for the case library from which the AMR could select and implement. Each case has its own set of seven gains that characterise its behaviour and can be found in Appendix A. In addition to this, each case also stores values for each of the eight environmental parameters. These are used to match the case against the prevailing conditions that the AMR senses itself to be in. The cases are listed below:

1. Clearfield In an open environment, the AMR should pay no attention to obstacles (since there will not be any), increase the goal gain, lower the noise gain and noise persistence.

2. Ballooning When there are relatively few obstacles, the AMR attempts to swing around them in a wide way (increase obstacle gain).

3. Squeezing When there are many obstacles, the AMR attempts to find a path by squeezing between obstacles (lower obstacle gain, increase goal gain and exit goal gain).

4. Hugging When there are many obstacles and the AMR is currently faced with an obstacle directly in its path, it attempts to hug the side of the obstacle as it makes its way around it.

5. Shooting Regardless of the number and size of the obstacles surrounding the AMR, if the system sees its goal and there are no obstacles in the way, it adopts an extreme version of the Clearfield strategy and goes directly to it.

6. Random The AMR raises the noise gain, obstacle gain and exit goal gain, leaves the goal gain at a medium level, and wanders for a period of time.

7. Repulsion In certain situations, the system considers moving away from the goal for a period of time. If, for example, the AMR senses the goal and there are obstacles surrounding it, it may decide to 'back away' for a distance before

attempting to get to the goal. This is accomplished by setting the goal gain to a negative amount and raising the exit goal gain.

8. Wallcrawling If there is an obstacle the system cannot seem to get around by Hugging and no progress is being made, the AMR will try to follow along the wall of the obstacle to try to get around it. This is accomplished by raising the temporary goal gain and lowering the goal gain, and allows the AMR to travel away from the goal while following the wall of an obstacle.

4.2.4 Case Selection and Matching

Since there are a limited number of cases in the library, there is little need for a sophisticated indexing scheme. As a result, a flat memory model of the case library is employed [Kolodner 92]. Cases are located by a sequential search of the memory for index matches. Although generally an inefficient method, this suffices for the AMR without impairing the 'reactiveness' of the system.

An important issue in case-based reactive control is determining when to switch cases. The simplest method is to look for a new case at regular time intervals and to switch cases if the current case does not match the environmental conditions as well as one of the other cases in the case library. The system used for this controller looks for a new case every second unless one of the last three cases in the case library has been selected (Random, Repulsion, and Wallcrawling). When this occurs it will wait three seconds before looking for a new case. These cases are normally only selected when the AMR has not been able to make any progress towards the goal and are therefore allowed to run for a longer period to hopefully move the AMR into a more favourable position in its environment.

Another important issue in case-based reactive control is that of selecting the best case from the case library. This controller uses a Goodness-Of-Match metric to find the case that best matches the current environment. In the flat indexing scheme used, this is accomplished using a matching algorithm that compares the current environment, as

perceived by the AMR, to each known case. The matching algorithm has different weightings for each of the environmental parameters measured and handles near matches through partial credit. An example of this, matching the *Clutter* environment parameter, is shown in Equation 4-8. After all the cases have been examined, the one with the best match is chosen.

Equation 4-8

$$\text{Goodness-Of-Match} = \begin{cases} 0 & \text{for } m \geq 3 \\ 1 - \dfrac{m}{3} & \text{for } m < 3 \end{cases}$$

$$m = |\text{Clutter} - \text{Case\_clutter}|$$

4.2.5 Performance

The controller was evaluated using the same 500 worlds from each class of obstacle densities. The results are shown in Table 4-6.

Performance of Adaptive Reactive Controller			
Obstacle Density	Percentage of Worlds Solved	Percentage Efficiency of Solved Worlds	Average Time (sec)
15%	98.6	86	16.9
20%	94.6	77	27.0
25%	85.2	73	34.8
30%	81.4	69	40.9
35%	79.8	64	49.8

Table 4-6 Performance of the adaptive reactive controller for each of the different obstacle density environments.

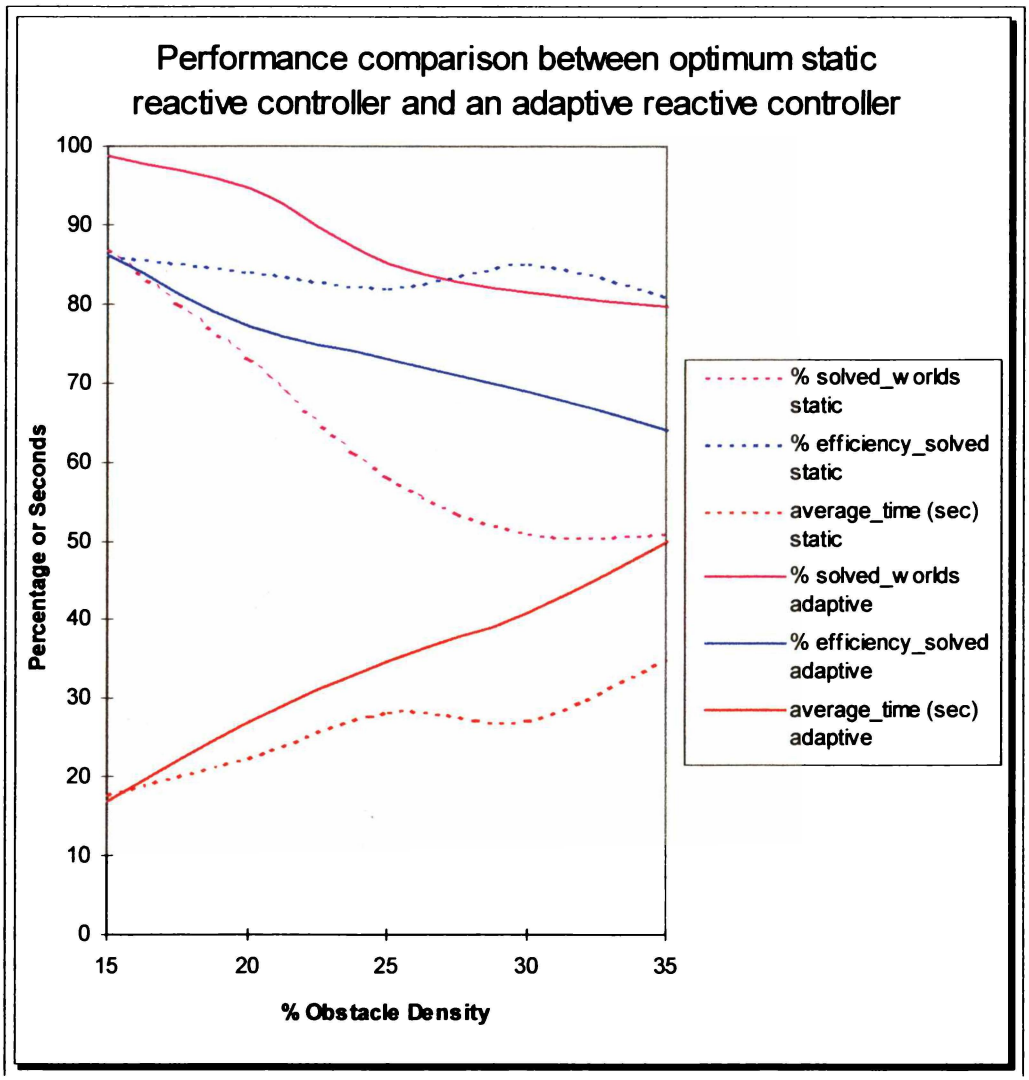

Figure 4-7 Performance comparison between the optimum static reactive controller and the adaptive reactive controller.

As can be seen from Figure 4-7, the percentage of solvable worlds is far higher using the new control system. The average time and efficiency of the solved worlds gets worse as the obstacle density increases, but this is due to the fact that there are many more difficult worlds getting solved than with the original control system. The performance of this controller is, however, still not 'robust.' While all but a handful of worlds are solvable at a 15% obstacle density environment, two out of every ten worlds are unsolvable in a 35% obstacle density environment. The cause of the problem is still the potential minima created by the environment and the type of controller used. The new controller can escape one area only to be trapped in another and because there is no effective memory the AMR can end up oscillating between these 'dead ends.'

As a comparison to Figure 4-5, Figure 4-8 shows the paths the AMR took for the first six worlds again. This time the sixth world is solvable because of the different behaviours being able to be selected from the case library.

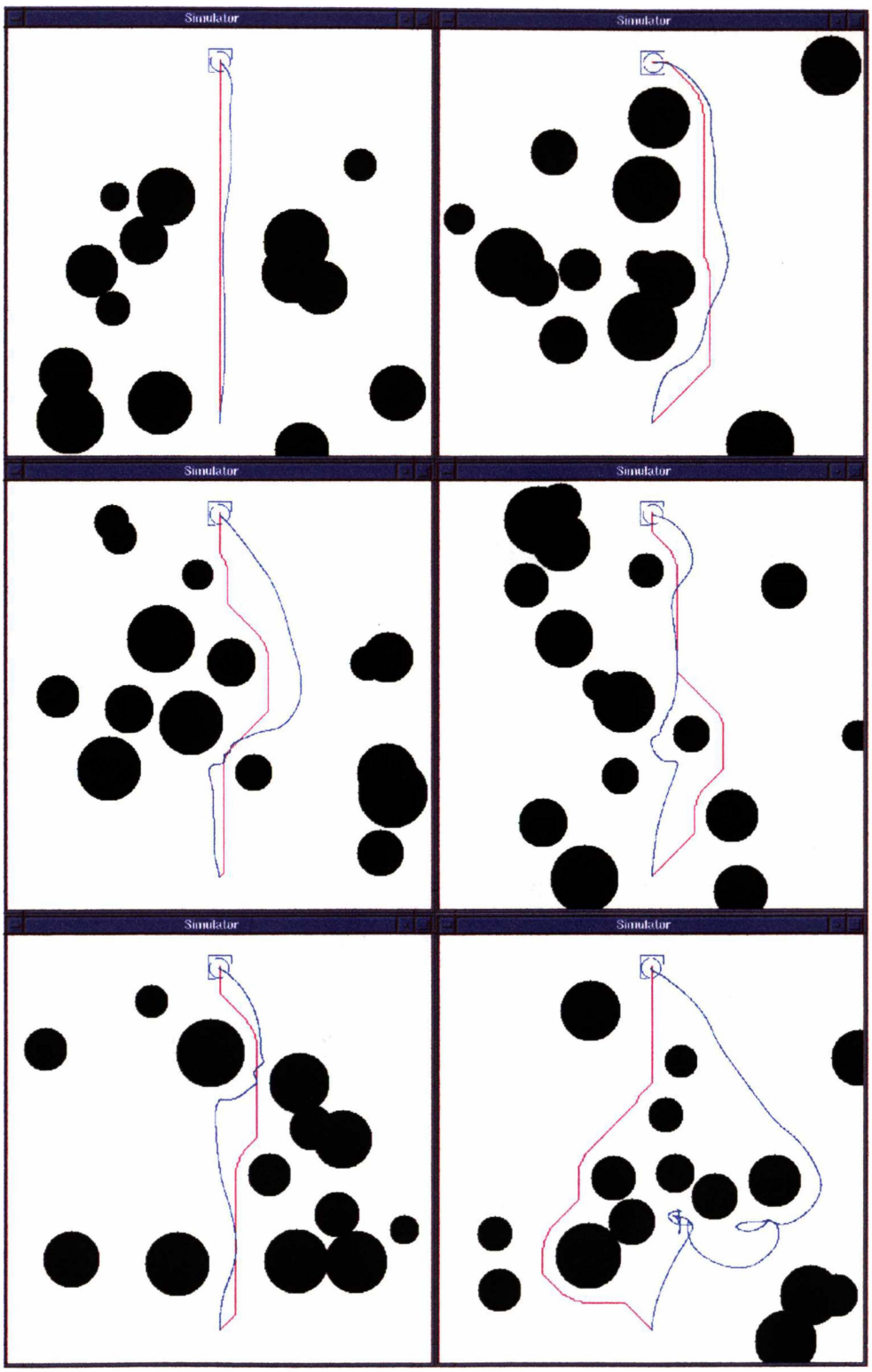

Figure 4-8 Plots of the optimal (magenta) and actual trajectories (blue) to reach the goal for the first six worlds at 15% obstacle density using the adaptive reactive control system. Note that the sixth world can now be solved.

4.2.6 The Necessity of a Memory

Reiterating, even though the performance of the AMR is greatly improved with the use of an adaptive reactive controller its performance is still not 'robust.' Figure 4-9 illustrates one of the few failures that occur in the 15% obstacle density library of worlds. This world presents quite a problem to the AMR as it contains a large 'bottleneck' type of area that is a potential minimum directly between the AMR's start position and the goal. Once the AMR travels into this area, it is trapped as the area is large enough for the AMR to circulate around inside it even when such behaviours as Repulsion or Wallcrawling are employed. In this run, the AMR does eventually manage to escape from this 'bottleneck,' yet only to re-enter it again a short time later. This problem would be avoidable if the AMR had a memory that contained some form of a spatial representation of the environment.

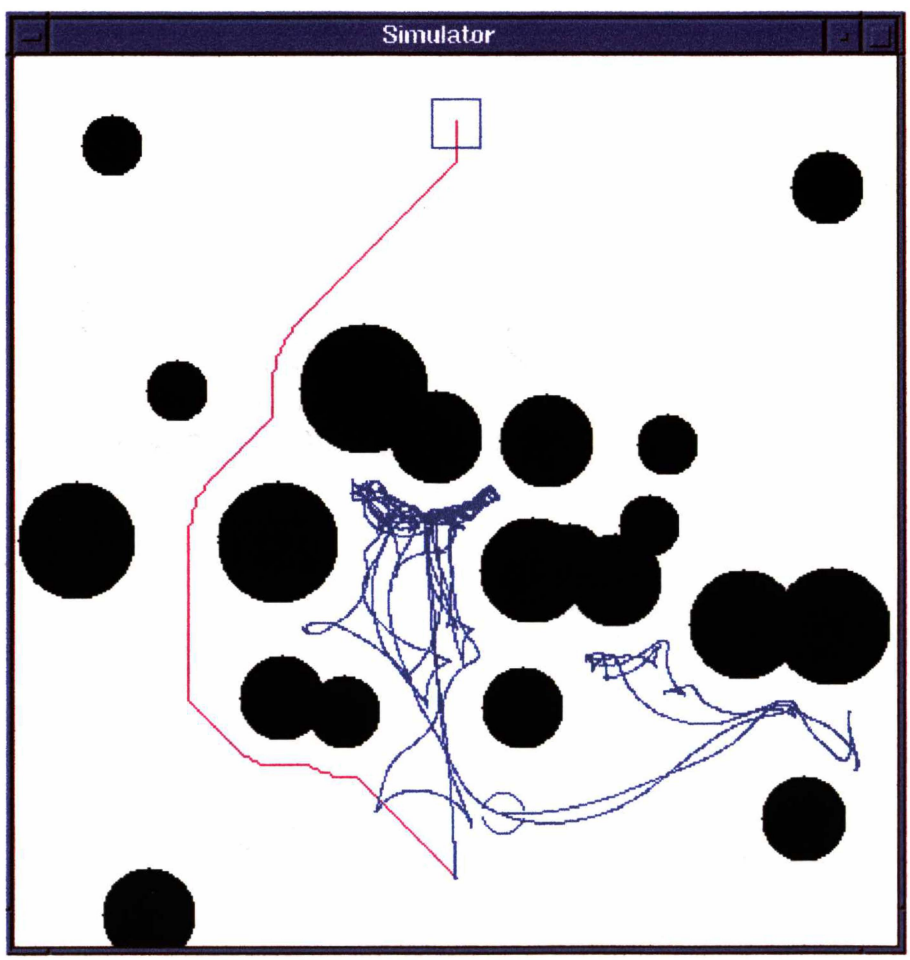

Figure 4-9 Trajectory of AMR in World No. 135, Obstacle density 15%. This world presents quite a problem to the AMR as it contains a large 'bottleneck' type of area that is a potential minimum directly between the AMR's start position and the goal.

Another example of a failure with this form of control system is shown in Figure 4-10. The structure of this world is such that the AMR ends up continually oscillating between two geographically different positions. These two positions are local potential minima and using the adaptive reactive controller the AMR is capable of escaping each of them. However, without some form of a spatial representation of the environment, the AMR has no means of stopping itself from oscillating backwards and forwards between these two areas.

It is possible to improve the performance of the AMR by introducing more cases and trying to optimise the case selection algorithm. However, without some form of a spatial representation of the environment there will still remain a sizeable number of unsolvable worlds and the efficiency of many of those that are solvable will be unacceptably poor.

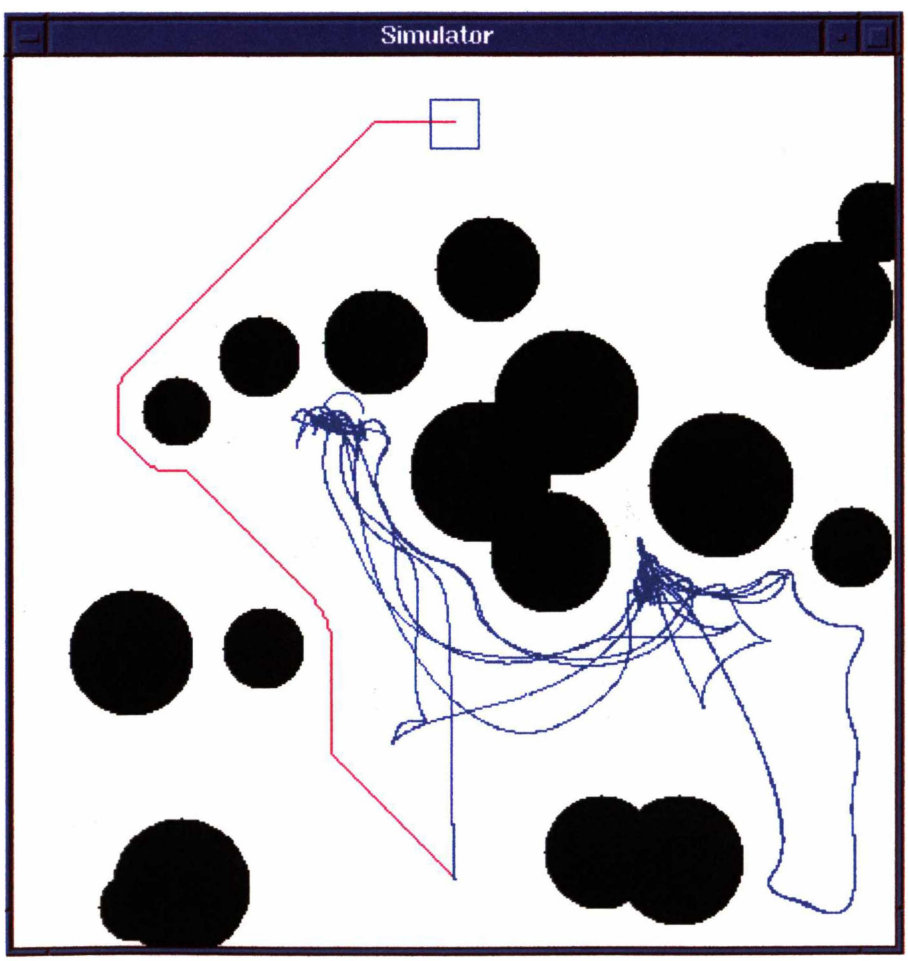

Figure 4-10 Trajectory of AMR in World No. 69, Obstacle density 20%. The AMR fails in this world again due to a lack of being able to recognise previous localities as being 'dead ends.'

Chapter 5

5. Evolving Reactive Controllers

5.1 Overview

The controllers developed in Chapter Four were designed by hand. The question arises, was the performance of the controller limited by the composition of the cases that were used? If so, how is the performance to be improved upon? The only option available, apart from a hand-design technique, is to try to evolve a control system that will yield improved performance.

A neural network provides a representation of a control system that is very easy to evolve. Techniques such as genetic algorithms can be applied to a neural network to drive the evolution of the network towards its optimal configuration in a given domain. This form of controller was developed and evaluated to see how feasible it is to evolve reactive controllers for an AMR functioning in the same complex spatial domains as those used in the previous experiments.

5.2 Neural Network Structure

To compare the performance of the neural network with that of the CBR control system the inputs and outputs used in Section 4.2 are preserved in the neural network as the input layer and output layer respectively. The neural network structure used for these simulations is a simple five neuron feedforward network. A feedforward network is characterised by having no feedback in the system. The overall structure of the network is shown in Figure 5-1.

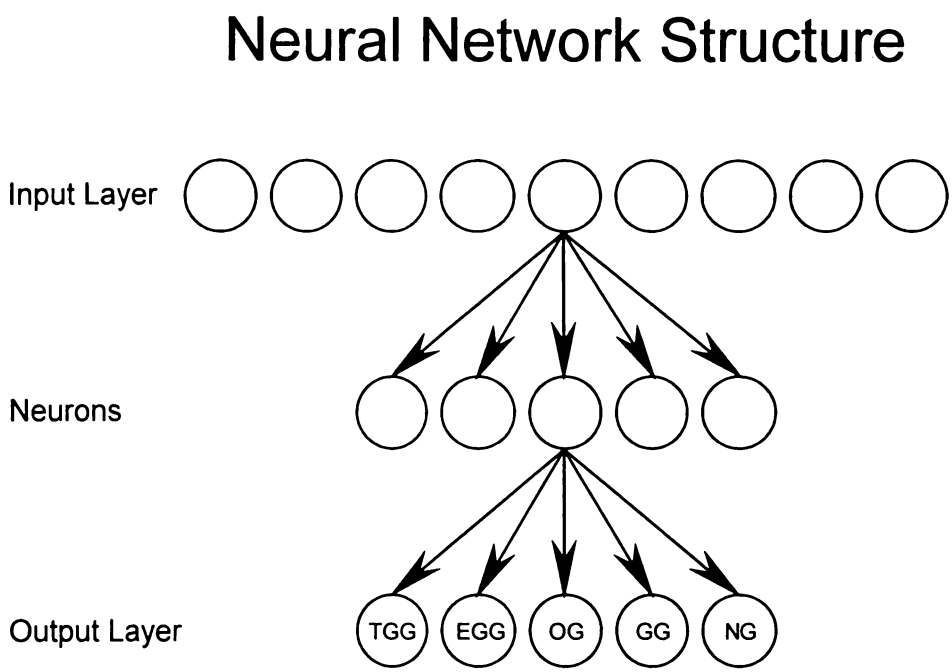

Figure 5-1 The structure of the neural network used. Input 5 and Neuron 3 are examples showing how each layer is connected. Every input is connected to every neuron and every neuron is connected to every output

The input layer is equivalent to the environment information used for the CBR system. These parameters are - clutter, density, proximity, wander, clear-to-goal, senses-goal, goal-nearby, and nomovement. Descriptions of each environment parameter can be found in Section 4.2.2. The only other input used in this network is a reference input that supplies a constant input of 0.5 to the network.

The hidden layer consists of five neurons. Each neuron has nine inputs and five outputs with associated weights. Each of these connections also has a minimum and a maximum activation level that determines whether the signal is transmitted or not.

The output layer is equivalent to the control parameters used in the CBR system. These parameters are - temporary goal gain (TGG), exit goal gain (EGG), obstacle gain (OG), goal gain (GG), noise gain (NG). The noise persistence and sensing distance were kept constant and did not form part of the neural network. Descriptions of each control parameter can be found in Section 4.2.1.

5.3 Biological Description of the Neural Network

The history of neural networks has its roots in the study of the human brain. Therefore, the following sections describe the neural network from a biological point of view. This provides a measure of the complexity of the AMR's controller when comparing to biological entities (see Section 5.8).

5.3.1 Genotype

The genotype of the neural network consists of a binary bit string of 3640 bits, which is composed of five chromosomes. This is all the potential genetic information that is contained describing the network. The resultant phenotypical network will have a bit-string length less than this if parts of the genotype are not expressed. For instance, if some connections in the network have a weighting of zero, then there is effectively no connection.

5.3.2 Chromosomes and Genes

A chromosome contains all the genetic information for a single neuron. The chromosome consists of a binary bit string of 728 bits, which is composed of fourteen genes. Each gene defines a particular connection to or from the neuron. The connection has a weighting and a minimum and maximum activation value as defined by the gene. The gene's bit string is comprised of fifty-two bits. Four bits are used to define the type

of connection, sixteen bits are used to store the weight of the connection and a further thirty-two bits are used to store the two activation values associated with the connection. In generating the initial population, the weights and activation values are determined randomly. However, each chromosome is provided with the fourteen possible different types of connection so that every input and every output is connected to the neuron. In future generations it is only possible for a 'gene type' to change through a mutation. The structure of the neural network from a biological perspective is shown in Figure 5-2.

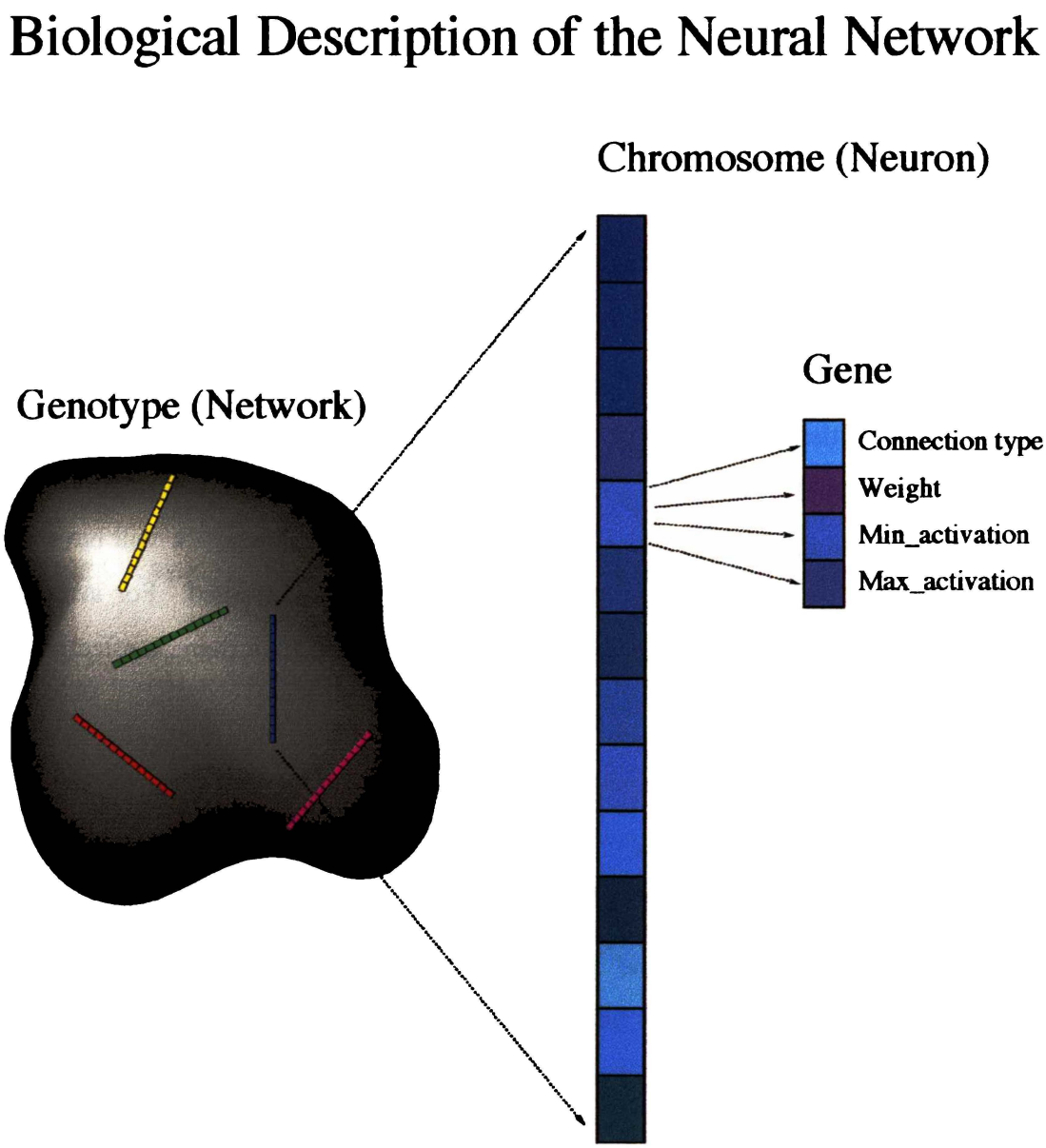

Figure 5-2 The structure of the neural network from a biological perspective.

5.4 Evolution of the Neural Network

5.4.1 Genetic Algorithms

Genetic algorithms [Holland 75] [Goldberg 89] are global search techniques patterned after Darwin's theory of natural evolution. A population of potential solutions (i.e. networks) is generated and evaluated in a specific task. The best solutions are then combined to form new solutions, which are inserted into the population. This process - evaluation and recombination - is termed a generation. Structures that led to good solutions in previous generations can be combined to form even better solutions in subsequent generations.

By describing the network as a binary bit string, as in Section 5.3.1, the maximum possible number of different solutions can be determined; in this case the 3640 bit genotype corresponds to 2^{3640} or $\sim 10^{1095}$ different solutions. Genetic algorithms are very useful for such large solution spaces as they sample many different areas of the solution space concurrently. Such a parallel search can also be very advantageous in multimodal (multi-peaked) search spaces that contain several good but suboptimal solutions.

5.4.2 Fitness

To find the best solutions in the population, the solutions must be evaluated in a specific task. This evaluation results in a "fitness" being ascribed to the solution which enables it to be ranked with the rest of the population. This forms the basis of the selection process such that 'fit' solutions are allowed to combine (or 'reproduce') and form new solutions. However, genetic algorithms do not require explicit credit assignment to individual actions. The only feedback that is required is a general measure of proficiency for each potential solution at the given task. Credit assignment for each action is made implicitly, since poor solutions generally select poor individual

actions. For this reason genetic algorithms belong to the general class of reinforcement learning algorithms.

In these simulations, the specific task is to navigate successfully from the starting position to the goal position within the simulated world. The 'measure-of-proficiency' used to determine the fitness of the evaluated solution is shown in Figure 5-3.

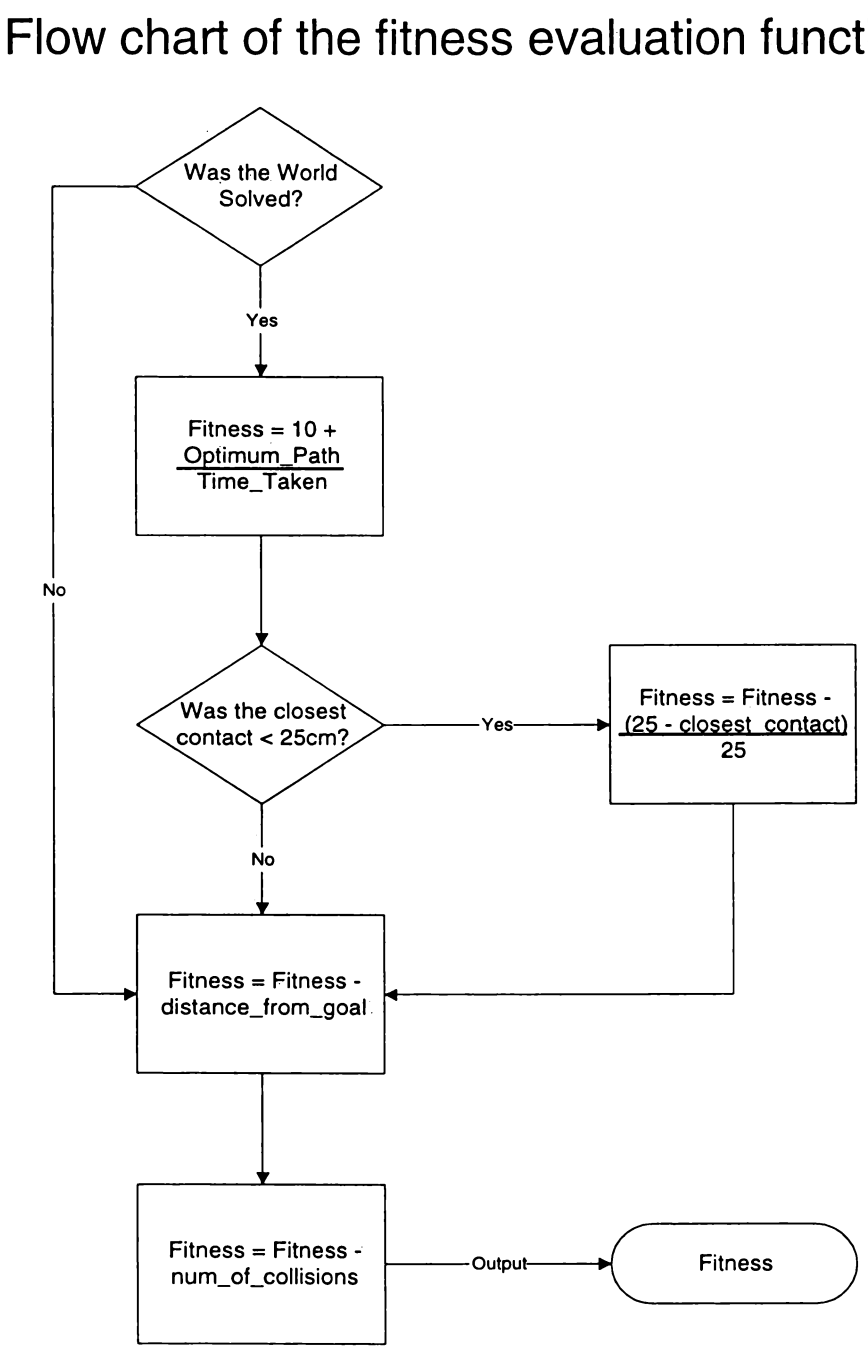

Figure 5-3 Flow Chart showing the calculation of the fitness value as a 'measure-of-proficiency' of the potential solution.

The fitness function shown in Figure 5-3 is a simple measure of the proficiency of the task considering that solving a world can take up to 6000 individual actions. In the simulations performed, a potential solution was evaluated over twenty worlds randomly selected from a library of 400 worlds. The obstacle densities of the worlds used ranged from 15-30%.

5.4.3 Selection and Reproduction

Once all the potential solutions have been evaluated they can be ranked in order of their overall "fitness." The selection process that determines which of the solutions is allowed to reproduce is based on the ranking of the solution. Only the top 25% of the population is allowed to reproduce in each generation. The bottom 25% of the population is 'removed' to make room for the newly-formed solutions.

Mates are selected for each potential solution within this upper quartile. The mate is randomly selected from a small number of potential solutions that have similar rankings. This selection method means that the 'fittest' will mate with other 'highly-fit' solutions while a 'mediocre' solution will normally mate with another 'mediocre' solution.

Reproduction consists of every set of parents producing two children. These children receive their parent's genetic information through a process similar to meiosis. The chromosomes of both parents are replicated and a crossover is performed at some point along the chromosome as shown in Figure 5-4. This means that the children of the parents inherit some genetic information from each parent. Therefore the newly-formed solutions will have similarities to their parents yet will probably be unique. Looking at this from a spatial point of view, the children represent two new points in the search space of the problem. The children are related to the parents in the sense that their positions within the search space of the problem are in hyper-planes between the two parental locations. This exchange of genetic information is the primary means of converging on the 'fittest' solution in the search space.

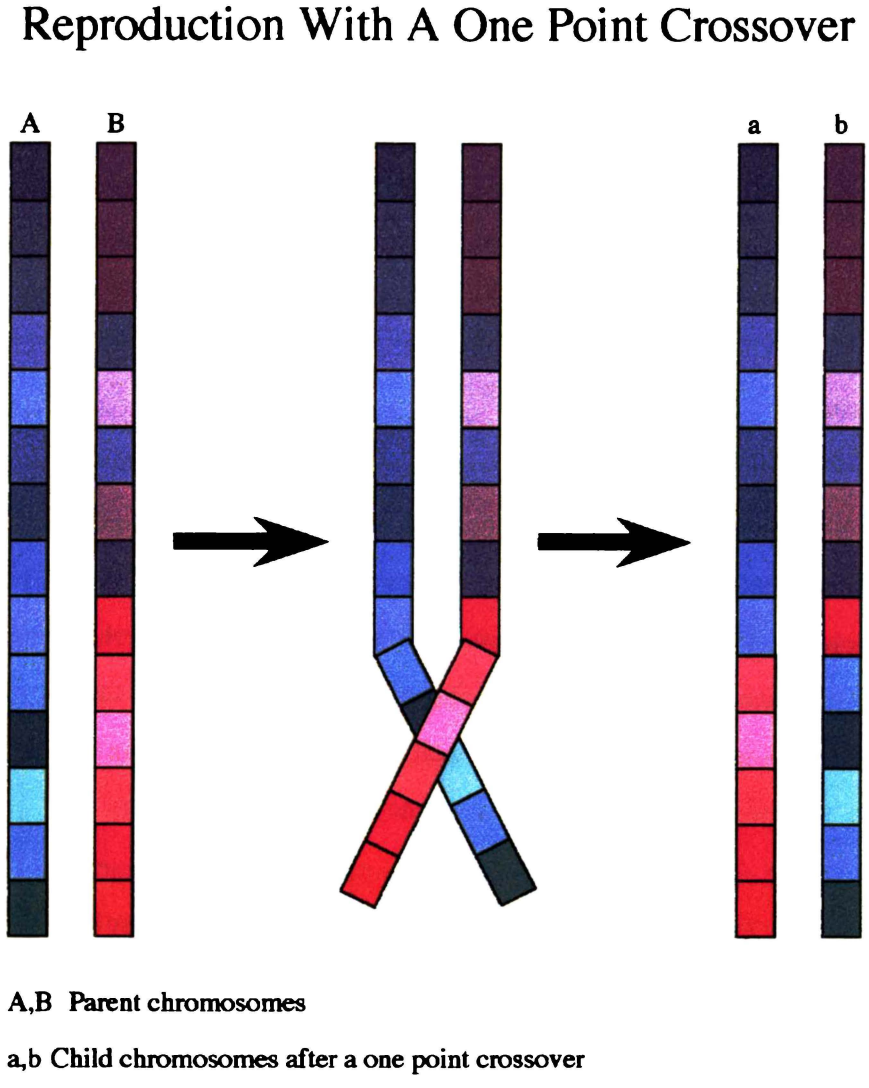

Figure 5-4 Chromosome reproduction with a one point crossover.

In the process of reproduction there is also a probability of mutations occurring in the genes of newly-formed solutions. These mutations can affect any part of the gene, from altering the connection type (and hence function) of the gene to modifying its weight or activation values. The probability of a gene's connection type being mutated is set at 0.2% per generation, while the probability of a mutation in the weighting or activation values of the gene is 2% per generation. Mutations are primarily used to reintroduce genetic material into the population to help diversify the genetic content of the population. This process slows convergence and promotes a wider search of the solution space. For these simulations, the mutations were performed on the genes at the unsigned integer level, as opposed to mutating individual bits within the gene's bit string.

5.4.4 Symbiotic Evolution

Since genetic algorithms do not require explicit credit assignment to individual actions, they belong to the general class of reinforcement learning algorithms. Standard supervised learning techniques are not applicable to this type of control problem as the domain information necessary to generate target outputs would be extremely difficult to obtain. However, while reinforcement learning methods require less a priori knowledge than supervised techniques, they generally require a large number of training episodes and extensive CPU time.

Normally, evolutionary strategies operate on a population of full solutions to the problem. In symbiotic evolution [Moriarty & Miikkulainen 94], each population member is only a partial solution. The goal of each individual is to form a partial solution that can be combined with other partial solutions currently in the population to build an effective full solution. In the case of a neural network, the population consists of individual neurons, and full solutions are complete neural networks. Because single neurons rely on other neurons in the population to achieve high fitness levels, they must maintain a mutualistic symbiotic relationship.

The fitness of an individual partial solution is calculated by summing the fitness values of all possible combinations of that partial solution with other current partial solutions and dividing by the total number of combinations as is shown in Equation 5-1.

Equation 5-1

$$f_{avg} = \frac{\sum_{i=1}^{n\max} f_i}{n\max}$$

f_{avg} = average fitness of the partial solution

f_i = fitness of a neural network that the partial solution is a member of

$n\max$ = the total of all possible combinations of that partial solution with other current partial solutions

Thus the individual fitness value reflects the average fitness of the full solutions in which the individual participated. In practice, however, evaluating all possible full solutions is intractable. Hence the average fitness values are approximated by evaluating n random subsets of partial solutions (n full solutions).

Partial solutions can be characterised as specialisations. Instead of solving the entire problem, partial solutions will specialise towards one aspect of the problem. Specialisation ensures diversity which prevents convergence of the population. A single partial solution cannot "take over" a population since, to achieve high fitness values, there must be other specialisations present. If a specialisation becomes too prevalent, its members will not always be combined with other specialisations in the population. Thus, redundant partial solutions do not always receive the benefit of other specialisations and will incur lower fitness evaluations. Evolutionary pressures are therefore present to select against members of dominant specialisations. This is quite different from standard evolutionary approaches, which converge the population, hopefully at the global optimum, but often at a local one. In symbiotic evolution, solutions are found in diverse, unconverged populations.

Separate specialisations will optimise different objective functions. Evolution will, in effect, conduct separate, parallel searches in each specialisation. This concurrent optimisation creates a faster, more efficient search, which allows the population to discover better solutions faster, and to more difficult problems.

5.4.5 Method

Symbiotic evolution is used in these simulations due to the complex nature of the task being optimised. Thus, the population that evolves from generation to generation does not actually contain a number of full solutions (i.e. networks), rather, the population consists of 200 chromosomes (i.e. neurons). In every generation, 500 networks are sampled and evaluated. The networks are sampled by randomly selecting chromosomes from the population until the accumulated bit string equals that of the genotype as

defined in 5.3.1. This newly-formed network is then evaluated at the task and the fitness is assigned to each of the participating chromosomes.

At the end of the generation, each chromosome will have participated in twelve to thirteen networks on average. Every chromosome will have a fitness reflecting the overall performance of the networks in which it participated. These average fitness values are then used in the selection and reproduction processes to generate the new population of chromosomes.

A block diagram of the simulation is shown in Figure 5-5.

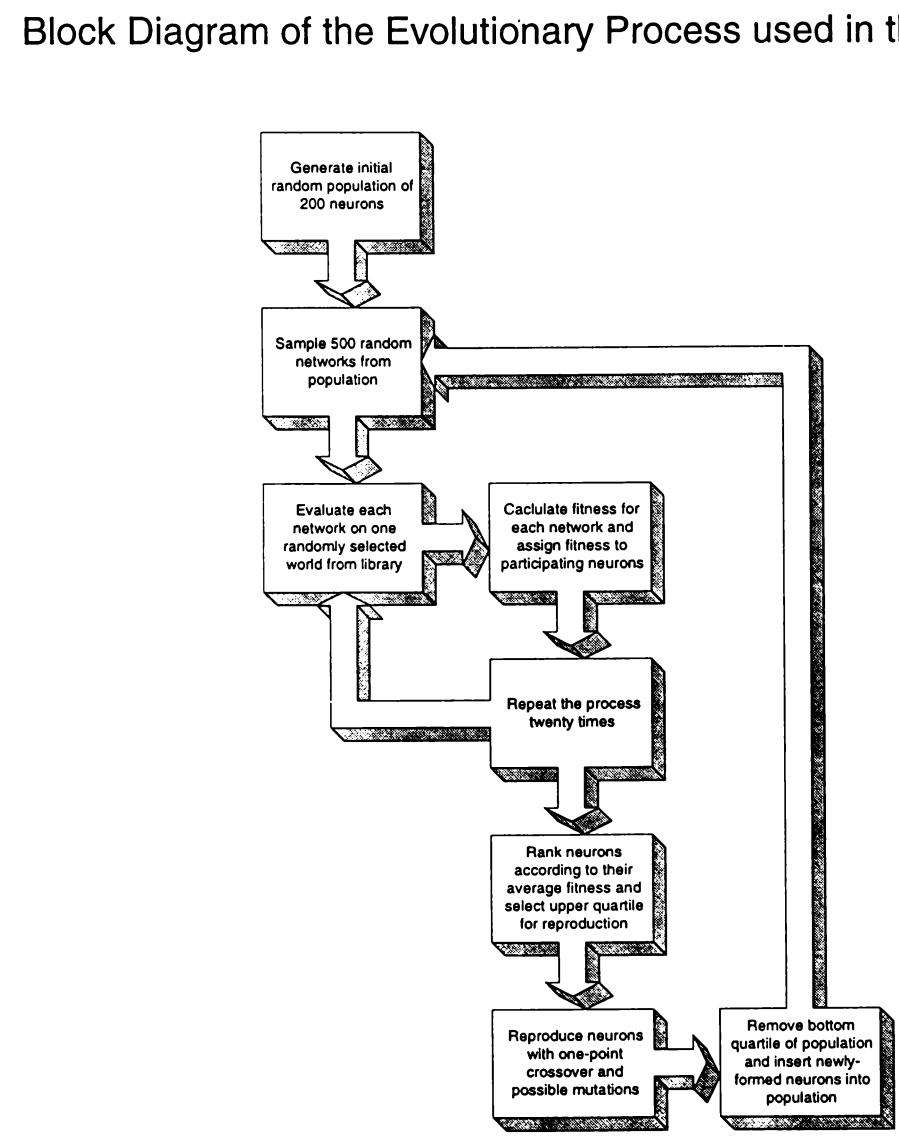

Figure 5-5 Block diagram showing the evolutionary process used in the simulations.

5.4.6 Implementation

The data structure used to represent a neuron is shown below:

```
struct Neurons{
  unsigned char    gene[14];
  unsigned short weight[14];
  unsigned short  a_max[14];
  unsigned short  a_min[14];
};
```

This structure requires 784 bits instead of 728 as stated in Section 5.3.2. While only four bits are needed to represent the connection type (gene) they are, however, stored in an unsigned char (eight bits wide), hence the extra 56 bits.

The "C" code for the neural network controller is shown below:

```
Inputs are normalised.
input[0] = environment.density / 360;
input[1] = environment.clutter / 10;
input[2] = robot.ofd / 250;     /* robot.ofd = f in Equation 4-6 */
input[3] = environment.cleartogoal - 0.5;
input[4] = environment.sensesgoal - 0.5;
input[5] = environment.goalnearby - 0.5;
input[6] = environment.wander;
input[7] = environment.nomovement - 0.5;
input[8] = 0.5;
for (j=0;j<networksize;j++) {
  neuronvalue = 0;
  for (k=0;k<neuronsize;k++) {
    x = neuron[j].gene[k];
    if (x < 128) {
      x = x % numinputs;
      y = (double) -1.0 + neuron[j].weight[k] / 32767;
      y = y * input[x];
      amax = (double) -1.0 + neuron[j].a_max[k] / 32767;
      amin = (double) -1.0 + neuron[j].a_min[k] / 32767;
      if ((amax>y) && (amin<y)) neuronvalue += y;
    }
  }
  for (k=0;k<neuronsize;k++) {
    x = neuron[j].gene[k];
    if (x > 127) {
      x = x % numoutputs;
      y = (double) -1.0 + neuron[j].weight[k] / 32767;
      y = neuronvalue * y;
      amax = (double) -1.0 + neuron[j].a_max[k] / 32767;
      amin = (double) -1.0 + neuron[j].a_min[k] / 32767;
      if ((amax>y) && (amin<y)) output[x] += y;
    }
  }
}
TGG = output[0] * 10;
EGG = output[1] * 10;
OG  = output[2] * 10;
GG  = output[3] * 10;
NG  = output[4] * 10;
```

Due to the fact that the AMR is often in the same position over many control cycles, a performance improvement can be made by pre-calculating the sensory information and storing it in an array. A 220 x 220 array was chosen as this resulted in a spatial resolution of three centimetres, which was considered an adequate resolution considering that the sensor data is only accurate to seven bits (±2cm). The structure is shown below:

```
struct Map{
  unsigned char rangedata[360];
  short              goalrange;
} xycoord[220][220];
```

To further improve the performance of the neural network, the code needed to be multi-threaded. This was achieved by designing the algorithm to output its results to file after evaluating all 500 networks on a single world. A shell script was then used to spawn the processes over the department's cluster of DEC Alphas. Each process evaluated the sample of networks for their particular world. When twenty worlds had been evaluated, the script spawned another process to collate all the output files and perform the 'Selection and Reproduction'. This resulted in the creation the next generation of neurons.

5.5 Performance

Reinforcement learning algorithms are typically CPU intensive due to the large number of training episodes required and the domain of AMR navigation is certainly no exception. As an example, the training time for the proposed simulation is approximately 3000 CPU hours on a DEC Alpha 2000/300 computer!

To enable the simulation to be completed, the code was designed to be multi-threaded (see Section 5.4.6) allowing it to run on all the departmental DEC Alphas concurrently. The DEC Alphas share a common file-system and the implementation incurred minimal

overheads. With three DEC Alpha 2000/300 computers and two DEC Alpha 3000/300 computers the simulation was completed in eighteen days!

The simulation was terminated after 100 generations had been evaluated. The average number of worlds solved, the corresponding time taken, and the efficiency was recorded for each generation. Figure 5-6 shows the evolutionary progress of the simulation. The oscillations are primarily due to the fact that only twenty random worlds, from a library of 400, are used to evaluate the sampled neural networks. This results in some generations having more difficult worlds to solve than others. However, this did ensure that a diverse range of environments were presented to the genetic algorithm while reducing considerably the total time taken to find the optimal solution.

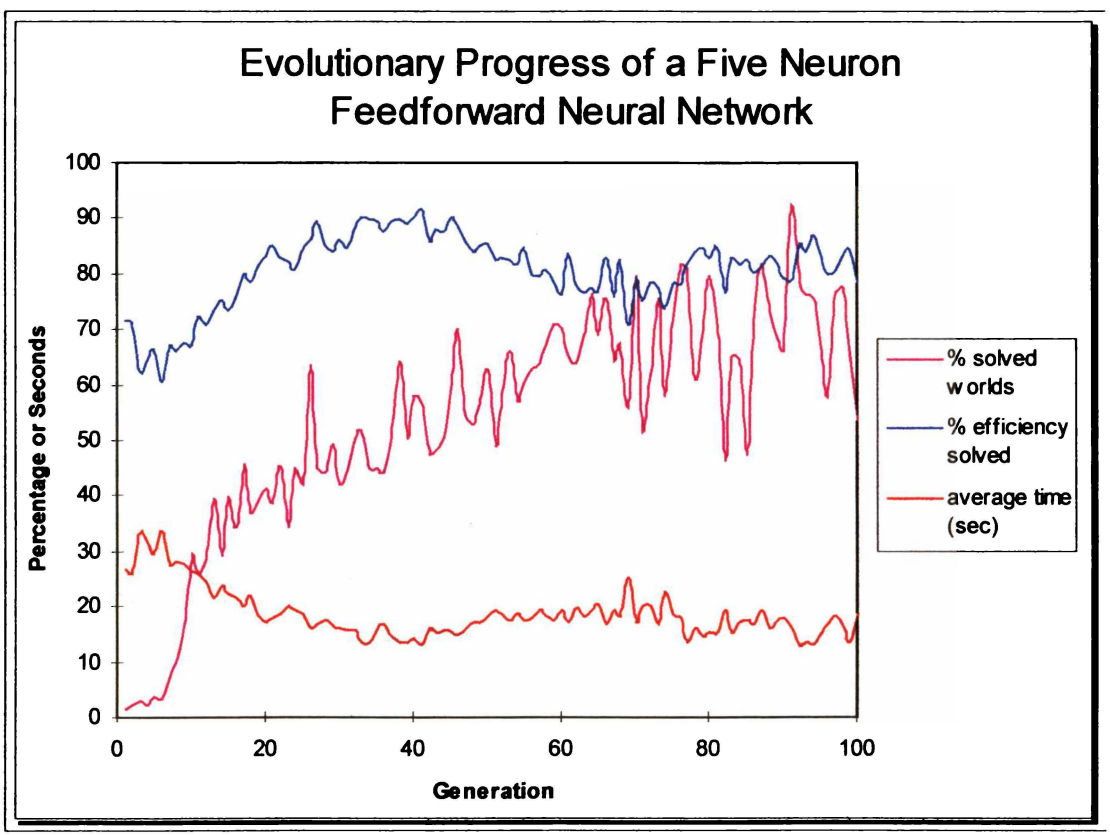

Figure 5-6 Evolutionary progress over 100 generations of a five neuron feedforward neural network.

The best network for each generation was also recorded and evaluated using the same library of worlds as used in Chapter Four. The best network formed was found to be in the fifty-ninth generation. Table 5-1 shows the performance of the network over the entire range of environment obstacle densities.

Performance of Neural Network Controller			
Obstacle Density	Percentage of Worlds Solved	Percentage Efficiency of Solved Worlds	Average Time (sec)
15%	98.8	81	16.6
20%	93.0	75	23.4
25%	85.4	73	31.7
30%	82.2	70	38.6
35%	80.4	68	41.6

Table 5-1 Performance of the neural network controller for each of the different obstacle density environments.

Figure 5-7 displays a comparison between the evolved neural network, the adaptive reactive controller, and the optimum static reactive controller. The neural network and the adaptive reactive controller both have similar inputs and outputs, yet the evolved controller solves a similar number of worlds approximately ten percent faster. This comparison demonstrates that the evolved neural network does indeed outperform the hand-designed case-based reasoning system.

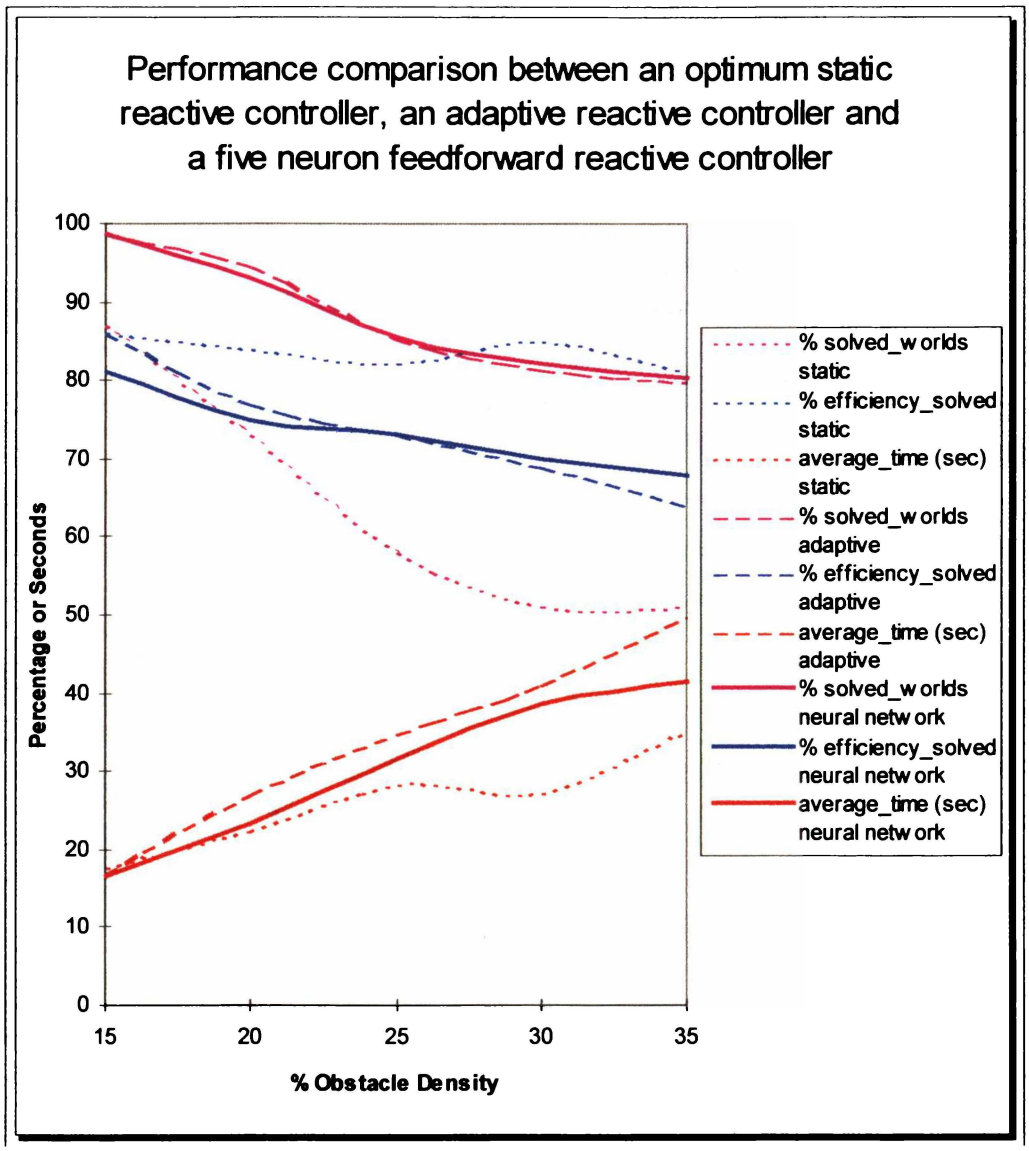

Figure 5-7 Performance comparison between an optimum static reactive controller, an adaptive reactive controller, and the evolved five neuron feedforward neural network controller.

As a comparison to Figures 4-5 and 4-8, Figure 5-8 shows the paths the AMR took for the first six worlds in the 15% obstacle density library of worlds. As with the CBR system all the worlds were solved, however, the paths taken are in some cases longer. This is supported in the results where the efficiency of solving worlds in the 15% obstacle density library decreased from eighty-six percent to eighty-one percent.

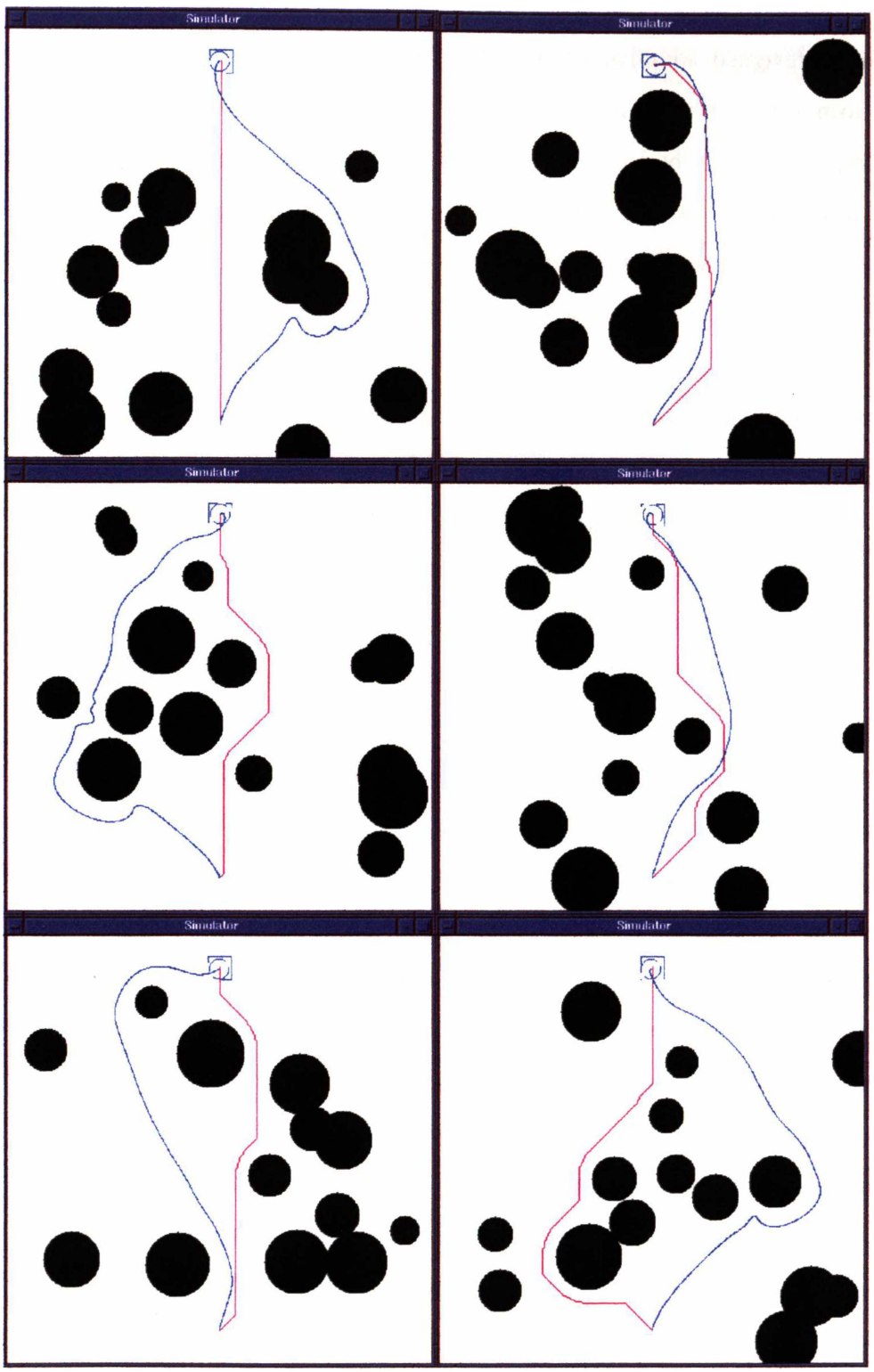

Figure 5-8 Plots of the optimal (magenta) and actual (blue) trajectories to reach the goal for the first six worlds at 15% obstacle density using the five neuron feedforward neural network.

5.6 Neural Network Analysis

The hand-designed adaptive reactive controller from Section 4.2 uses eight different cases to navigate through the environment. Upon examination of the neural network's outputs, approximately 120 different sets[1] were recorded as the AMR attempted to solve a variety of worlds. However, these sets could be grouped into five distinct classes. Two of the classes were similar to those designed for the adaptive reactive controller; namely the hugging and random behaviours.

Examination of the neural network's inputs also showed a marked difference to the adaptive reactive controller. The network manages to achieve this level of performance while not making use of the first two environmental parameters, namely the obstacle density and the obstacle clutter parameters. The activation values for these two inputs on every neuron had evolved to a state that is out of the parameter's range of values. Therefore these two parameters contribute nothing to the output units of the network.

The performance of the system is somewhat limited by the pre-processed environment information that is provided to the input layer of the neural network. Another limiting factor is the output layer which can only vary the five control parameters instead of controlling the heading of the AMR or controlling the torque applied to each of its wheels. However, removing these constraints greatly increases the search space of the problem, resulting in an evolutionary task that is not a viable option given our department's current computing resources.

The failures with this controller are similar to the previous controllers in that the AMR ends up either trapped in a 'potential minimum' as shown in Figure 5-9, or perpetually oscillating between two or more such states. As cited in Section 4.2.6 some form of memory is required to prevent the AMR from getting 'stuck' in these situations. A

---

1 The outputs of this neural network are continuous. They were logged each control cycle and then classified into different sets when any output varied by more than 0.1. Using such a fine resolution resulted in approximately 120 sets being classified. However, many of these sets were very similar (due to the fine resolution initially selected) and could be grouped into five distinct classes.

feedforward neural network by definition has no ability to store information and therefore cannot evolve to such a state.

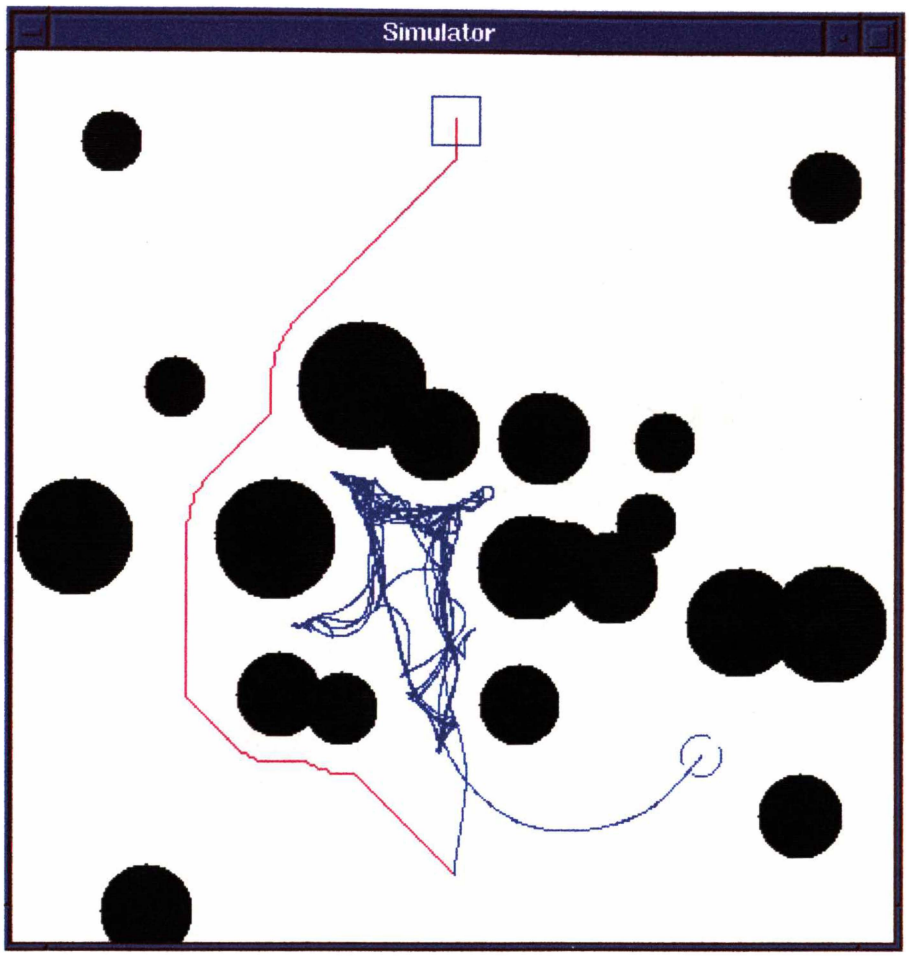

Figure 5-9 Trajectory of the AMR in World No. 135, Obstacle density 15%. This world still presents a problem to the AMR as the feed forward controller has no 'memory.'

5.7 Recurrent Neural Networks

A recurrent neural network is an architecture that incorporates feedback connections. Figure 5-10 shows an example of the connection structure of a five neuron fully recurrent neural network. In this example, every neuron is connected to every other neuron in the network, hence the term 'fully recurrent.' This feedback allows for recursive computation and the ability to represent state information, which could be considered a form of memory. Consequently, this type of network, upon favourable evolution, could potentially solve more complex worlds than the simpler feedforward neural network.

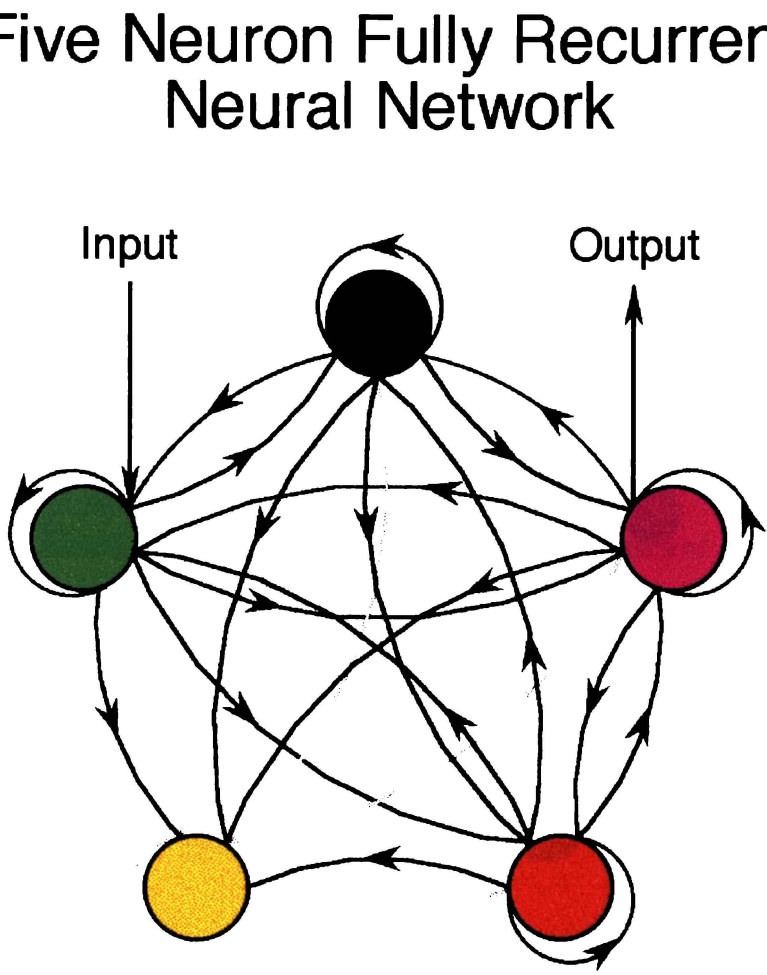

Figure 5-10 An example of the connection structure of a fully recurrent five neuron network with one input and one output.

An architecture to evolve a variable size recurrent network, with an upper limit of fifty neurons, was implemented. Raw sensory data were presented at the inputs, resulting in twenty-five different inputs. This was possible by reducing the effective resolution of the laser range-data from one degree to fifteen degrees. An output heading angle was the only output unit in the network. The simulation (which at first sight seemed possible) was started, however, the population made no evolutionary progress.

Upon examining the search space of this simulation it become readily apparent as to the reason why there was no convergence in the 'evolving' population. The fifty neuron fully recurrent neural network had a genotype in the order of 100,000+ bits. The search space was therefore equivalent to $2^{100,000}$! The solution space to this problem is so small compared to the search space that evolutionary techniques are no longer a viable

process. This technique would not, even if all the computing resources currently available on this planet were utilised, be able to find a solution in our lifetime!

This illustrates the scalability problems encountered when trying to apply genetic algorithms to optimise problems with very large search spaces. Of course, compared to the human brain, which has ~50-100 billion neurons, a fifty neuron network is rather trivial. Even so, this technique is still unable to optimise such a problem. These evolutionary simulations show that if complex **robust** behaviour is desired, then a hand-designed control system is the only viable option available.

If it was possible to design a more complex dynamic fitness function then this problem could undoubtedly be optimised using a genetic algorithm. However, this is the equivalent to hand-designing a controller, because, such knowledge is simply being used to reduce the 'effective' search space of the problem until it is capable of being simulated and solved.

5.8 Genetic Coding In Nature

As a comparison, the smallest self-replicating organisms in nature are viruses. The smallest of these viruses have genomes in the order of 3500 base pairs. There are four different bases used in DNA or RNA, so this is equivalent to a binary bit string of 7000 bits. The original optimised five neuron feedforward network had a genome of 3640 bits. Viruses, while self-replicating are parasitic in that they require a host to provide the necessary machinery (such as ribosomes) to facilitate replication. This enables them to exist with such small genomes that do not contain the necessary genetic information to produce the extra proteins required to build ribosomes.

At the other end of the biological scale is a human being. Our genotype is in the order of $3x10^9$ base pairs. Every human then, is a unique sample out of a search space that would be $2^{6,000,000,000}$ in size! If we are the product of evolution, then statistically, we're quite a miracle....

Chapter 6

6. Creationist Approach

6.1 Introduction

Having been unable to produce a satisfactory controller through the use of evolutionary techniques, a creationist technique is now further developed. Earlier experiments have shown that failures were due to the AMR's inability to recognise areas already explored. Hence, for an AMR to function adequately in a complex environment it requires some form of spatial representation of its environment.

As discussed in Section 2.2 recent research has produced two fundamental paradigms for modelling real world environments: the grid-based (metric) paradigm and the topological paradigm. Table 2-1 shows that these two approaches to robot mapping exhibit orthogonal strengths and weaknesses. A topological approach was chosen because it permits efficient planning, has a low space complexity and provides a convenient representation for problem solvers. These factors are critical when trying to preserve the reactive nature of the AMR.

6.2 Topological Maps

A topological map can be described from graph theory [Smith 87] as a series of nodes (or vertices) that correspond to distinct situations, places, or landmarks that are connected by edges (or arcs) if there exists a direct path between them. However, in the AMR's domain, what constitutes a distinct situation? The AMR is equipped with a 360 degree laser range-finder and scanning at 100 hertz results in 35 kilobytes of range information every second. Obviously, to keep the performance of the AMR reactive, most of this information must be discarded, and therefore the number of bytes used to describe a node must be kept to a minimum.

If a topological map is to be useful, it has to provide information on how to navigate from one place to another. As a topological map consists of nodes linked together by edges, the complexity of each node should be such that the AMR can move from node to node by using the edge linking them. To preserve the reactive nature of the AMR, an environment needs to be described with as few nodes as possible, yet these nodes in themselves cannot be spatially complex.

6.2.1 Distinguishable Features

To provide the most reliable topological map, the designer has to pick appropriate distinguishable features from the AMR's environment on which to base the map. These features must be able to be characterised using the AMR's sensory system. With the use of a 360 degree laser range-finder, sensory information is entirely geometrical. Consequently, distinguishable features are the geometrical shapes of obstacles and their relationship to other obstacles. Ideally, the selected feature(s) should be resolvable independent of the viewing perspective of the AMR. Therefore, using the geometrical shape of an obstacle is not adequate, because its shape varies as the AMR's perspective shifts as it moves to a new position in the environment.

The smallest vector between two obstacles, however, is a feature of the environment that not only can be viewed from many different perspectives but can be described with two variables, bearing and length. To determine the smallest vector between two obstacles, requires the obstacles themselves to be resolved from the 360 degree laser range-finder scan. An obstacle is resolved from the sensor data by:

1. searching for large changes in adjacent range values.
2. searching for large changes in the angle of the scanned surface.

An example is shown in Figure 6-1. The AMR laser range-finder scan is shown as red dots. The green vectors represent the closest points of every resolved obstacle in the scan. The white vector points to the mid-point of the smallest vector between the two closest obstacles that define the current distinguishable place.

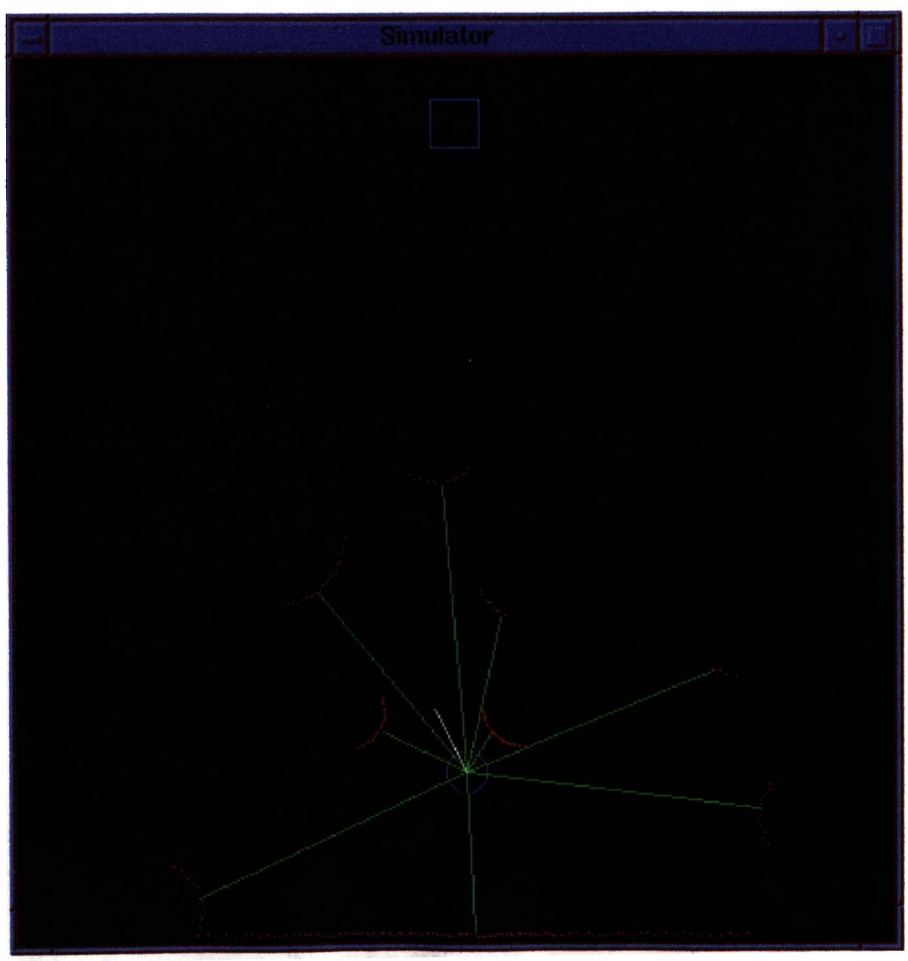

Figure 6-1 depicts the AMR in World No. 135. The red dots show a laser range-finder scan. The green vectors represent the closest points of every obstacle resolved from the scan. The white vector points to the mid-point of the two selected obstacles which define the current distinguishable place.

Once the desired features on which to base the topological map have been defined, the next issue is to determine the rules that define which node the AMR is currently associating itself with. In this case, the feature used is the smallest vector between two obstacles, therefore, selecting the two closest obstacles relative to the AMR could define the current node with which the AMR is associated with. This node will then be the position the AMR considers itself to be in with respect to its map.

Figure 6-2 shows a topological map of world No. 135. The map was produced by calculating the smallest vector between the two selected obstacles at every pixel position in the world. The various colours (based on the vector's bearing) display the different areas within which the AMR associates itself to a particular node. However, some of the large areas of colour, especially around the edges of the room, do not constitute the AMR associating itself to a single node. These areas are where there are two or more adjacent nodes that have similar vector bearings.

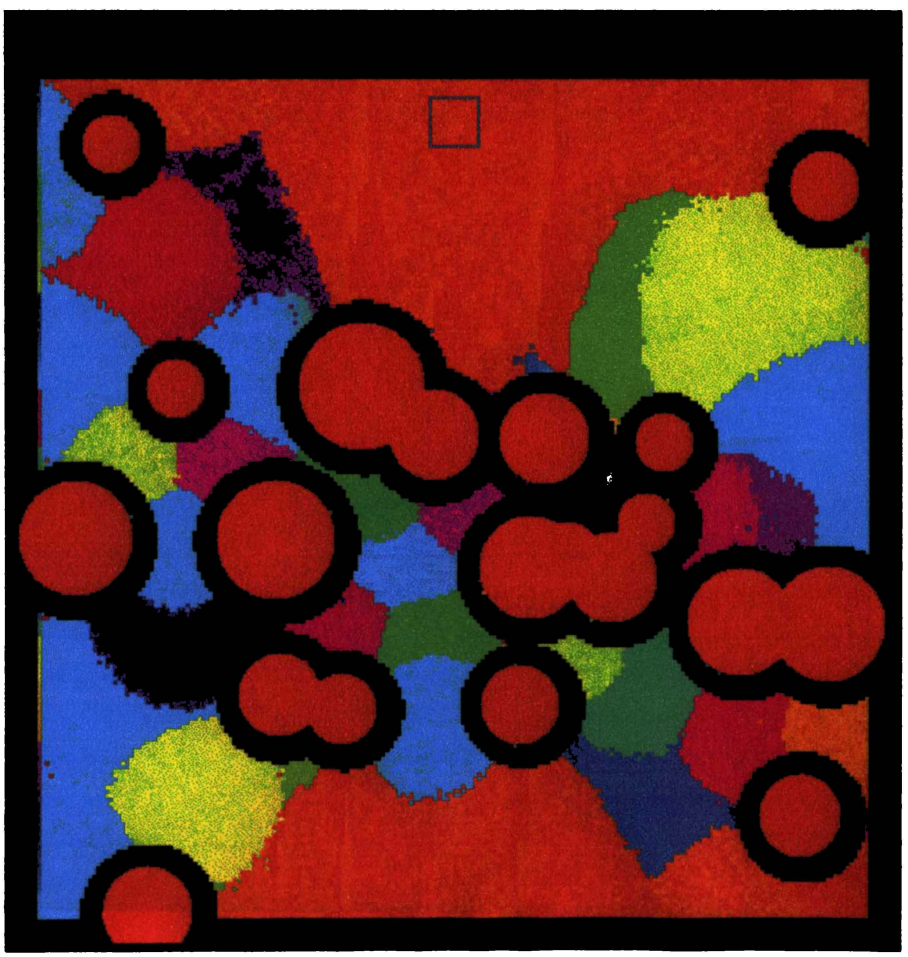

Figure 6-2 shows a topological map of World No. 135. The various colours based on the vector's bearing reveal the different nodes and the area within which they can be perceived in the environment.

Ideally each of these areas should be a solid colour. Such an area would represent a vector that is static when viewed from all perspectives. The AMR would then easily know which node it was associating its position with in the map. However, as can been seen from the plot, most areas have a blend of colour, revealing that as the perspective changes, the smallest vector between the obstacles moves around. Therefore, each node in the map also stores the variance of the smallest vector to enable the AMR to compute whether or not the current sensor data corresponds to the node that the AMR is associating itself with. The use of circular obstacles accentuates this problem as there are no edges. Consequently, the AMR is more susceptible to this perspective problem in such environments.

In Figure 6-3, the mid-points (blue areas) of the vectors are plotted for the environment. The fact that each node is not represented as a point but as a small area again shows the effect of the perspective problem.

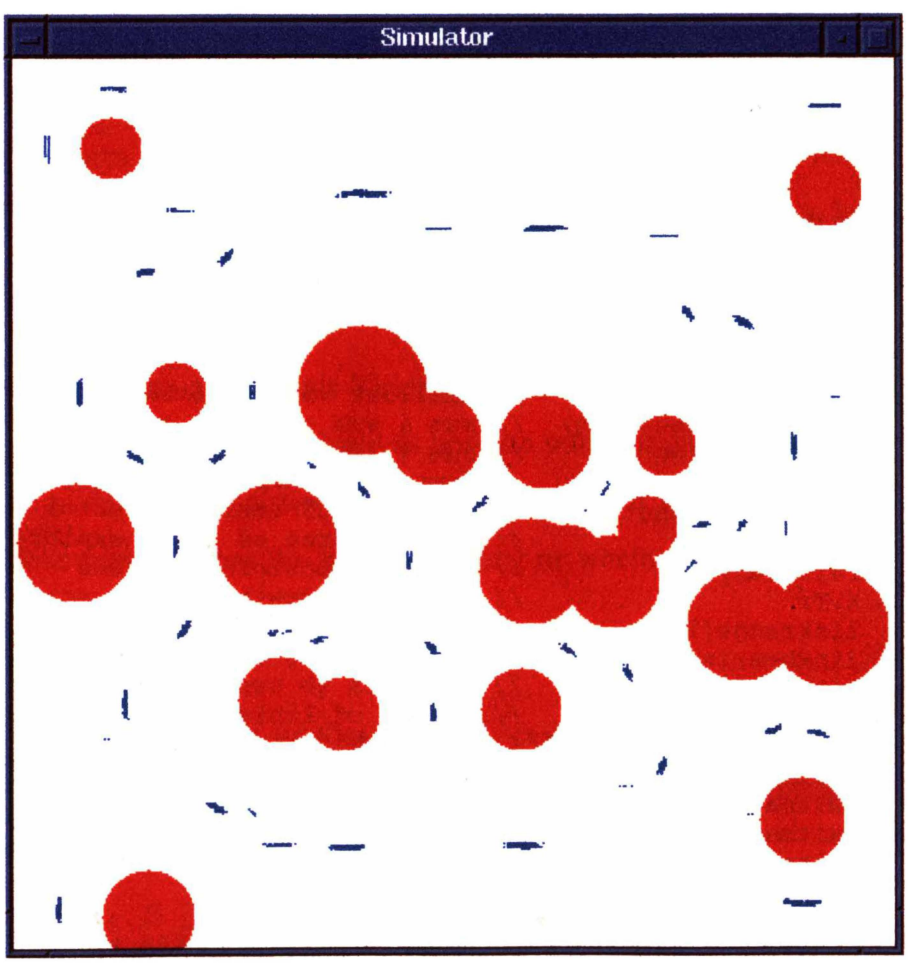

Figure 6-3 The blue areas define the mid-point of the vector between the current two closest obstacles for every position in World No. 135.

6.2.2 Construction of the Topological Map

A topological map consists of nodes (distinguishable features) that are connected by edges if there exists a direct path between them. If the environment that the map was representing had no circular paths in it (i.e. you could *not* move in a circle around an object(s) until you returned to your starting position) then the edges linking nodes together would not have to store metrical information. They would just be a link that represented one node as being adjacent to another. In a real world environment this is not a valid assumption. If the AMR is to recognise old nodes when moving in circular paths, then the topological map must have a geometrical structure.

In the topological map each node contains a list of the other nodes to which it is linked. Also contained is the vector information that links the middle of the node to the middle of the attached nodes. This information is determined as the AMR moves from one node to the next. The vector pointing to the middle of the old node is subtracted from the vector that points to the middle of the new node that the AMR has entered. The resultant vector is the link between the old node and the new node.

Shown below is the data structure for a single node in the topological map. This is data structure is not optimised.

```
struct Node{
  short  bearing,length;       // smallest vector between two obstacles
  double Sbearing,SSbearing;   // sum & sum squared; to calc. variance
  double Slength,SSlength;     // sum & sum squared; to calc. variance
  long   N;                    // number of samples; to calc. variance
  double Vbearing,Vlength;     // variance of smallest vector
  short  n,e,s,w;              // size of area as it is explored
  long   relx,rely;            // relative position of node
  long   X,Y;                  // used for drawing map to screen
  short  linkrange[20];        // link range info to connected nodes
  short  linkbearing[20];      // link bearing info to connected nodes
  short  linkmap[20];          // pointers to connected node's struct
  short  nlinks;               // number of links to/from this node
  short  exitmap[8];           // The 'exit' arrays store potential
  short  exitmapl[8];          // links that may exist to other nodes,
  short  exitmapvb[8];         // see Section 6.2.4
  short  exitmapvl[8];
  short  exitmask[8];
  short  exittrys[8];
  short  exitprob[8];
  short  mask;                 // used in path planning through map
  short  perspective;          // Set if AMR has a good perspective
};                             // of smallest vector, see Section 6.2.3
```

6.2.3 Self-Localisation within the Environment

To build an accurate geometrical structure into the topological map, the AMR requires accurate absolute positional information. The AMR is equipped with shaft encoders which provide position information, yet because of wheel slippage, there is an accumulative error in these data. This problem also occurs when racing a micromouse. A micromouse uses shaft encoder position information to determine which square it is currently in. However, the accumulative error is corrected by calibrating the positional information with the edge of side-walls in a maze. When a micromouse encounters an edge in the side-wall, it is either the start or finish of a wall section. Since the maze is a symmetrical environment of pre-determined dimensions, the micromouse can make use of this fact and calibrate its shaft encoders as long as the current error is less than half the length of a wall.

The same technique can be applied with the AMR's position information. Even though a real world is an environment of unknown dimensions, the technique is really that of *self*-localising the position of the AMR using natural features within the environment. These are features which are static and can be used to calibrate the AMR's position information. The smallest vector between two obstacles which is being used to construct the topological map is one such feature.

When the AMR enters an old node it uses the position of the node as determined by the AMR's laser range-finder, to calibrate its position information. While the AMR moves through the node it can compare its velocity with that of the node. This enables the AMR to determine whether the node is drifting or whether it is stationary and has a good perspective of it.

With this method of self-localisation and calibration, the AMR can construct topological maps that can resolve a circular path and update the appropriate old node rather than construct a new one.

6.2.4 Exits

Having a map of the environment that is incomplete is of no use to the AMR unless it contains possible exits. If the goal of the AMR is not located in the current map then the AMR must explore to find a path to the goal. The case-based controller of the AMR enables it to do this. However, when the AMR is trapped in a potential minimum, it must consult its map to decide where the next best place to search from is.

The reliable determination of exits from a node in the map is extremely important. False exits waste time as the AMR attempts to navigate through a wall, while not resolving a possible exit could end in failure for the AMR to reach its goal if the unresolved exit was the only way to get there.

Exits are determined from the smallest vectors calculated between adjacent obstacles in the AMR's laser range-finder scan. If the vector is larger than the AMR's diameter then it is marked down as a possible exit. Depending on which part of the node the AMR is in, its perception of the exits of that node can differ. To provide more accurate knowledge of the exits from a node, a probability is associated with each possible exit. The further the AMR has travelled in the direction of a possible exit, the stronger the weighting for the exit. This technique enables the AMR to remove possible exits that were detected in one part of the node, yet do not exist, because they were resolved when the AMR had an inadequate perspective.

Figure 6-4 shows a block diagram of the AMR's control system with the integration of the topological mapping system. This is also shown in pseudo-code form below.

```
struct Obstv            // obstacle vector data structure
{
  short bearing,range;
  short bearing_min,bearing_max;
}obstv[15];

struct Features
{
  short a,b;            // stores which obstacles define this feature
  short perspective;    // =1 if AMR has a good perspective of feature
  double bearing,length;// smallest vector between obstacles a&b
  double nvb,nvr;       // vector from AMR to mid-point between a&b
}features[50];

unsigned char rangedata[360];
```

```
while (robot.position != goal.position)
{
  Get_RangeData(rangedata); // see Section 3.3.1
  short numobstv = Resolve_Obstacles(rangedata,obstv);
  short numfeatures = Find_Features(numobstv,obstv,features);

  // The next function simply finds the feature with two closest
  // obstacles.

  short feature_num = Classify_Area(numfeatures,features);
  Update_Topological_Map(features,feature_num);
  Calculate_Environment_Information(); // see Section 4.2.2
  if (current_behaviour_is_failing()) SolveMap(); // see Section 6.2.5
  Update_Robots_VHP(); //  see Section 3.3.3
}


short Resolve_Obstacles(unsigned char rangedata[],Obstv obstv[])
{
  //  refer to Section 6.2.1
  short num_obstacles = 0;
  for(j=0;j<360;j++)
  {
    if (abs(rangedata[j+1] - rangedata[j]) > robot.diameter))
    {
      // Start of an obstacle…
      obstv[num_obstacles].bearing_min = Find_bearing_min();
      obstv[num_obstacles].bearing_max = Find_bearing_max();
      obstv[num_obstacles].bearing = Find_bearing();
      obstv[num_obstacles].range = Find_range();
      num_obstacles++;
    }
  }
  for (j=0;j<num_obstacles;j++)
  {
    if (find_large_angle_change(obstv[j]))
    {
      split_obstacle_into_two();
    }
  }
  return num_obstacles;
}


short Find_Features(short numobstv,Obstv obstv[],Features f[])
{
  short num_features = 0;
  short min_exit_width = 40;  // 40 cm  robot diameter = 30 cm
  for (a=0;a<(numvectr-1);a++)
  {
    for (b=a+1;b<numvectr;b++)
    {
      short minwidth = find_minimum_width_between(obstv[a],obstv[b]);
      if (minwidth >= min_exit_width)
      {
        get_feature_params(obstv[a],obstv[b],f[num_features]);
        num_features++;
      }
    }
  }
  return num_features;
}
```

```
Update_Topological_Map(Features f[],short fnum)
{
  if (abs(f[fnum].bearing - current_node.bearing) >
                              (current_node.Vbearing * 2 + 10))
  {
    if (abs(f[fnum].length - current_node.length) >
                              (current_node.Vlength * 2 + 20))
    {
      // This function creates a new node using the information of the
      // feature and links it back to the last current_node seeing
      // that was the last node the AMR was in.

      Node* node = Create_New_Node(f[fnum],current_node);

      // This next function just checks to see if the new node is in
      // the same position as one of the old nodes…

      current_node = Resolve_and_Circular_Paths(node);
    }
  }
  // This function computes the new variance of the current node
  // due to the addition of this feature's parameters

  add_feature_params_to_current_node(f[num],current_node);

  // This function updates the exit information of the node
  // based on the features extracted.

  update_current_node_exit_info(features,current_node);
}
```

6.2.5 Path-Planning

In Chapter 4 the adaptive reactive controller contained a random and a repulsion type of behaviour. These behaviours were included in the case library to enable the AMR to escape from potential minima and were only used when the other behaviours failed.

Now, however, instead of switching to these behaviours, the AMR switches to a new case. This new case supplies a series of temporary goals for the AMR to travel to which correspond to nodes along a path created by the path-planning routine. This results in the topological map only being used by the AMR when its reactive behaviours fail.

The path-planning routine functions in a similar way to a maze-solver [Brightwell 93]. The starting point is the node in which the AMR is currently situated. All possible exits are examined and assigned a weighting.

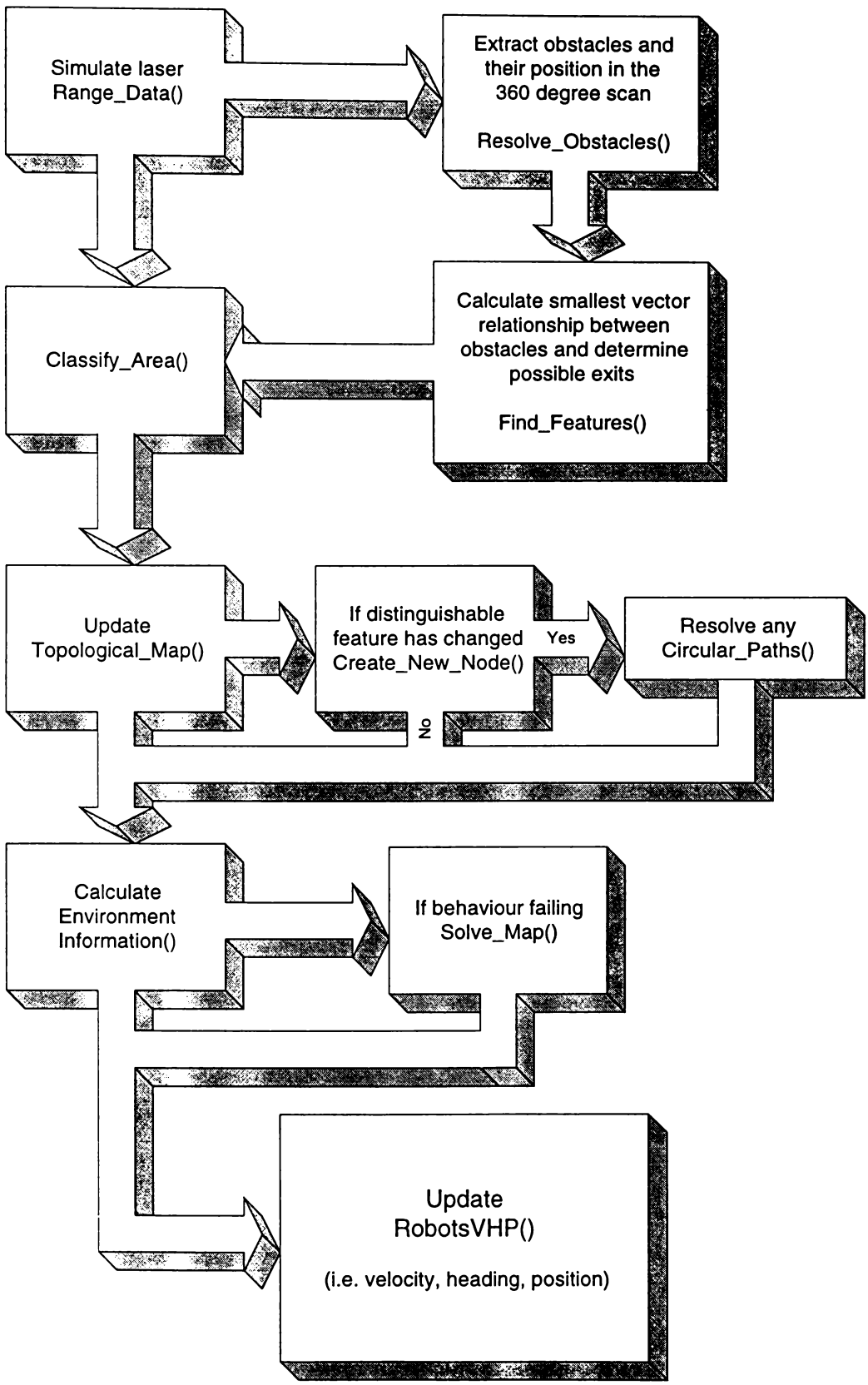

Figure 6-4 Block diagram of the AMR's control system.

The weighting as shown in Equation 6-1 is determined by the distance the node is from the goal, how far the AMR is from the node (zero in this case), and how similar the exit direction is to that of the goal direction. This process is then repeated with all the adjacent nodes. The exit with the lowest weighting is then selected, and a path is planned to the node that the exit is associated with. The final segment of the path includes the vector of the exit. The vector information describing the linkage of the nodes provides the information for the AMR on how to traverse from one node to another.

Equation 6-1 Formula used to determine the weightings for each node.

$$Weight_{node} = Link\_Weight_{node} + Exit\_Weight_{node}$$

$$Link\_Weight_{node} = \sum_{starting\_node}^{node} (1 + node\_linkrange)$$

$$Exit\_Weight_{node} = goal\_range_{node} + 100 \times \text{int}\left(\frac{\left| goal\_bearing_{node} - exit\_bearing_{node} \right|}{22.5} \right) + goal\_in\_sight$$

$$goal\_in\_sight = \begin{cases} -300 & \text{for } Clear\text{-}to\text{-}goal = 1 \text{ or } Senses\text{-}goal = 1 \\ 0 & \text{otherwise} \end{cases}$$

$\sum_{starting\_node}^{node}$ = sum of all nodes that are required to link the starting node to the current node

$node\_linkrange$ = distance from one node to another in centimetres

$goal\_range_{node}$ = straight line distance from node to goal position in centimetres

$goal\_bearing_{node}$ = bearing from node to goal position in degrees

$exit\_bearing_{node}$ = exit direction from node in degrees

6.2.6 Masking Exits

To prevent the AMR from repetitively travelling along the same path determined by an exit specified in the one of the nodes, a means of masking the exit must be employed. This situation occurs when the AMR has travelled to a dead end and must find a new area to search.

Exits are masked using three different procedures. Firstly, if a link has been made to a node in the direction of the exit then the exit is masked. Secondly, if a node has a similar characterisation (i.e. similar smallest vector between two obstacles) as the exit and is in the approximate position defined by the exit yet is not connected to the exit's node, then this exit is also masked. Thirdly, if a resolved exit passes through the side of a large adjacent node, then this exit is also masked. The pseudo-code for these three methods is shown below.

```
// 1st method

MaskExit1(Node* node)
{
 for (j=0;j<node->nlinks;j++)
 {
  linked_exit = (node->linkbearing[j] + 23) / 45;
  if (linked_exit == 8) linked_exit = 0;
  node->exitmask[linked_exit] = 1;
 }
}

// 2nd method

MaskExit2(Node* current_node,short exit)
{
 Node* node = current_node;
 short exitbearing = current_node.exitmap(exit);
 short exitlength = current_node.exitmapl(exit);
 bool finished = false;
 while (!finished)
 {
  finished = get_next_linked(node);
  if (difference(node->bearing,exitbearing) < (node->Vbearing*2+10))
   if (difference(node->length,exitlength) < (node->Vlength*2+20))
    if (difference(position(node),position(exit)) < 30) // 30 cm
    {
     current_node->exitmask[exit] = 2;
     finished = true;
    }
 }
}

// 3rd method

MaskExit3(Node* node,short exit)
{
 vector vectors[4];
 vector ev = calculate_exit_vector(node,exit);
 for (j=0;j<node.nlinks;j++)
 {
  xypos n = find_North_most_position(node.linkmap[j]);
  xypos e = find_East_most_position(node.linkmap[j]);
  xypos s = find_South_most_position(node.linkmap[j]);
  xypos w = find_West_most_position(node.linkmap[j]);
  vectors = calculate_vectors_to_NESW_positions(n,e,s,w);
  if (exit_is_within_arc_swept_out_by_vectors(ev.bearing,vectors))
   if (exitrange_is_greater_than_min_vectors(ev.range,vectors)
    node->exitmask[exit] = 3;
 }
}
```

Using these algorithms, exits are tried and marked off as they are explored, resulting in the AMR building a topological map of the environment and finding a path to the goal. However, if the AMR resolves a spurious exit, then this exit must also be masked to prevent the AMR continually trying to navigate along this exit path. This situation is handled by recording the number of attempts the AMR tries to navigate using the exit. If after two attempts, the exit has not been masked by one of the above procedures, then the exit is masked anyway.

6.2.7 Trajectory Optimisation

To improve the performance of the AMR navigating through known terrain, its trajectory is optimised. This is done by examining the nodes in the path that has been planned. If more than one node is within the AMR's sensor range and there is a clear path to each of them, then the trajectory is altered to drive straight to the furthermost node. The pseudo-code for this optimisation is shown below. If the AMR navigates into unknown territory or through nodes that are not in path planned, then the AMR calculates a new path and optimises the new trajectory.

```
struct Path  // can store up to 200 way-points
{
  Node* path[200];
  short bearing[200];
  short length[200];
  bool  mask[200];
  short last_entry_in_path;
} p;


for(j=p.last_entry_in_path-1;j>=0;j--)//last entry is 1st node in path
{
  Node* node = p.path(j);
  short bearing = get_bearing_from_current_pos_to_node(node);
  short range = get_range_from_current_pos_to_node(node);
  if (rangedata[bearing] > range)
  {
    p.mask[j+1] = true;
    p.bearing[j] = bearing;
    p.range[j] = range;
  }
  else break;
}
```

6.3 Performance

The controller was evaluated using the same 500 worlds from each class of obstacle densities as used in Chapters Four and Five. If the AMR failed to reach the goal after 200 seconds it was recorded as having failed the world. The results are shown in Table 6-1.

Performance of Adaptive Reactive Controller with Memory			
Obstacle Density	Percentage of Worlds Solved	Percentage Efficiency of Solved Worlds	Average Time (sec)
15%	100	86	17.3
20%	98.6	78	26.4
25%	98.6	70	36.4
30%	97.2	66	41.8
35%	98.0	67	39.7

Table 6-1 Performance of the adaptive reactive controller with memory for each of the different obstacle density environments.

As can be seen from Figure 6-5, the performance of this controller is a great improvement over that of an adaptive reactive controller that can not construct a spatial representation of its environment. Though the AMR still fails some worlds in the more difficult libraries, analysis of the topological maps created, revealed that in these worlds the AMR was still exploring exits when the 200 second limit was reached and that there were still unexplored exits that would have lead to the goal. Upon increasing the time limit, these worlds were subsequently solved.

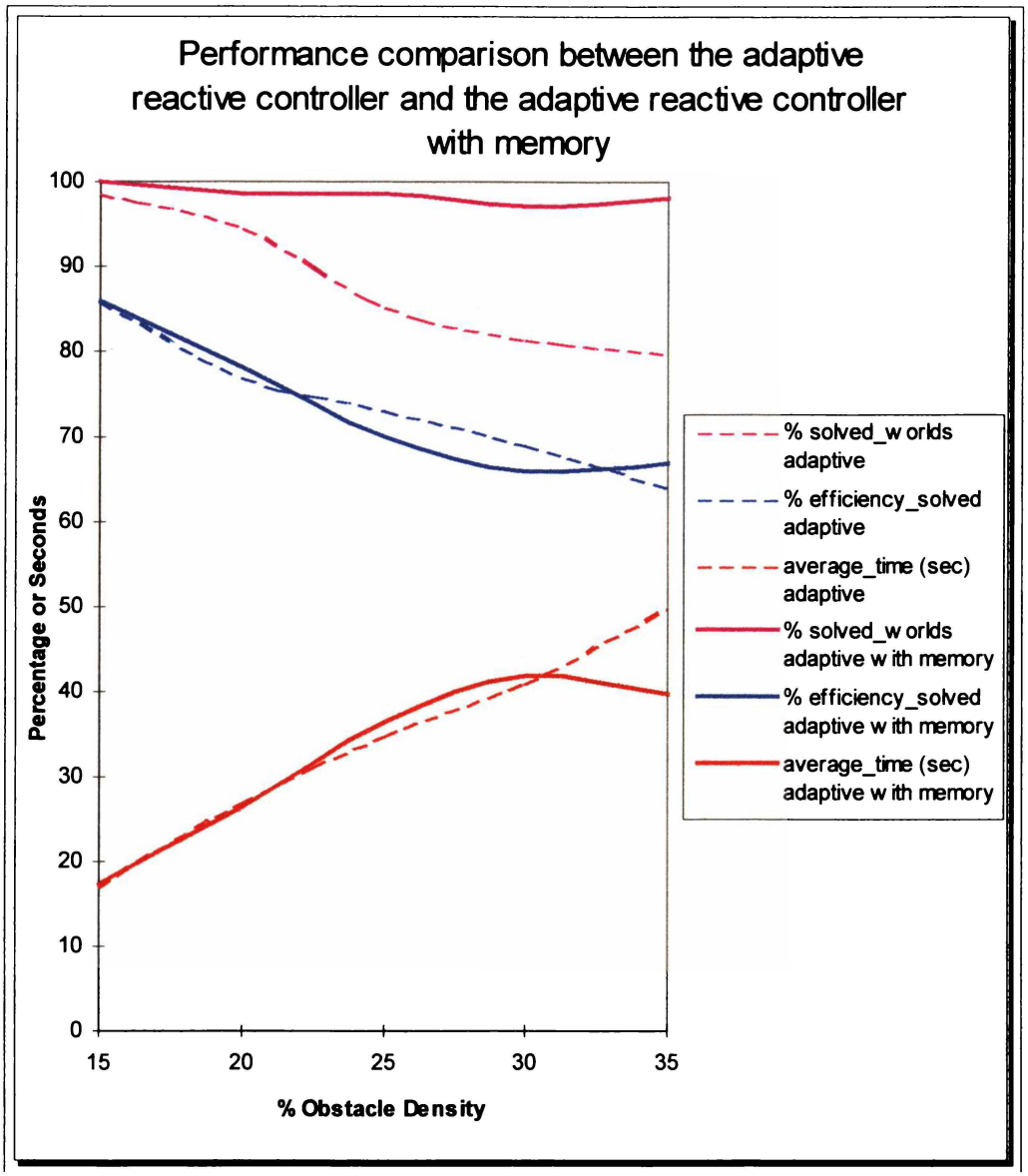

Figure 6-5 Performance comparison between the adaptive reactive controller and the adaptive reactive controller with memory.

The AMR performs worst in the 30% obstacle density worlds instead of the 35% obstacle density worlds. Though the difference in performance is statistically insignificant, given the sample size, there are features of these worlds that correlate with the time taken to solve them. In the 30% obstacle density worlds, the AMR resolved an average of 20.26 nodes per world. This is slightly less than the 35% obstacle density worlds in which the AMR resolved an average of 20.82 nodes per world. The 30% obstacle density worlds also had the highest average number of links per node as can be seen from Figure 6-6. These data show that the complexity of the environments peak in this particular sample of 30% obstacle density worlds. The number of links per node show that at 35% obstacle density the number of possible exits to search is less than in

the 30% class of worlds. Consequently, the average time taken to solve the worlds is worst at the 30% obstacle density worlds.

Analysis of Topological Maps Generated			
Obstacle Density	Average Number of Nodes per World	Average Number of Links per Node	Average Time (sec)
15%	10.97	1.94	17.3
20%	14.25	2.09	26.4
25%	17.38	2.16	36.4
30%	20.26	2.25	41.8
35%	20.82	2.23	39.7

Table 6-2 Analysis of topological maps generated in each class of worlds.

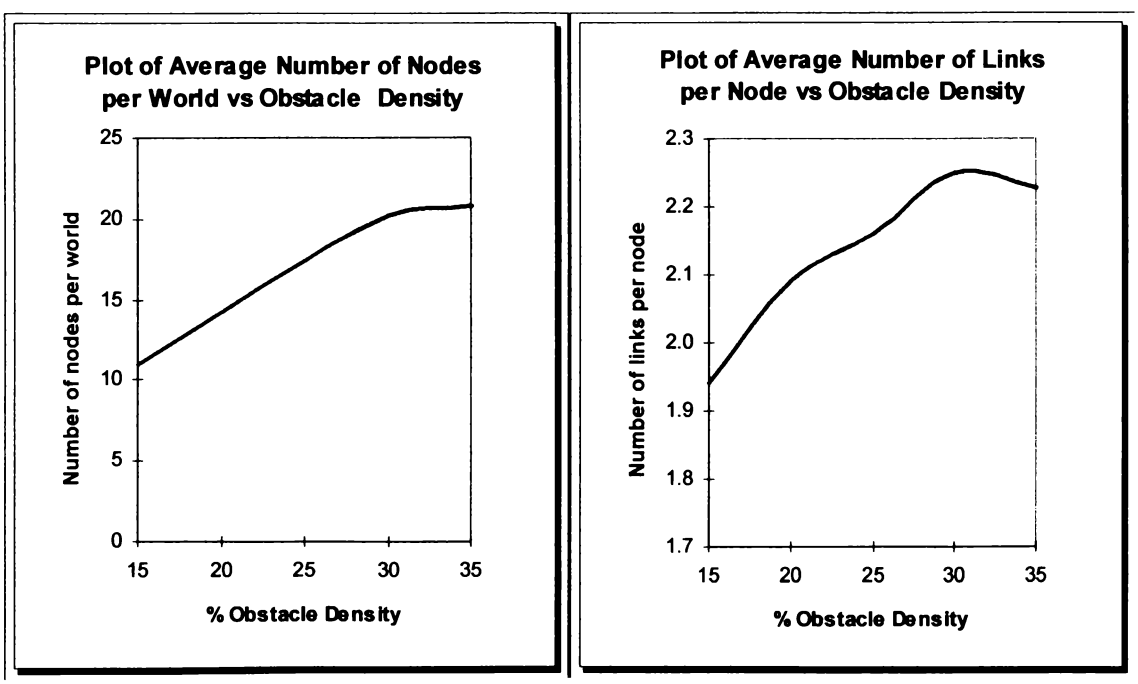

Figure 6-6 shows the increasing complexity of the worlds as the obstacle density is increased. The complexity peaks at the 30% obstacle density worlds.

6.3.1 Behaviour in Complex Worlds

To demonstrate the AMR's behaviour in a complex world such as Figure 4-9, a series of snapshots at ten second intervals were taken. The first snapshot in each interval shows the AMR with its current laser range-finder scan and the trajectory taken up to that point. The second snapshot shows the topological map generated for having travelled that trajectory. The different colours again reflect the vectors that distinguish each node as in Figure 6-2. The sizes of the nodes in some cases differ to those determined in Figure 6-2, owing to the fact that the AMR has only explored part of the node. The snapshots are shown in Figures 6-7 through 6-30. The performance of the AMR in the world is summarised in Table 6-3.

Points of Interest

Figure 6-9: The AMR has started using its topological map to explore possible exits as the reactive controller failed. This is noted by the magenta colour of the AMR.

Figure 6-15: Every possible exit in the 'bottleneck' has been explored. The AMR will continue to explore the remaining exit possibilities (i.e. possible exits in nodes one and two).

Figure 6-25: Having explored east of its starting position and finding no path to the goal, the AMR now begins searching to the west and discovers an exit in the north direction.

Figure 6-29: The complete trajectory taken by the AMR to reach the goal. The obstacles have been drawn in as a reference.

Figure 6-30: The complete topological map generated by the AMR upon reaching the goal.

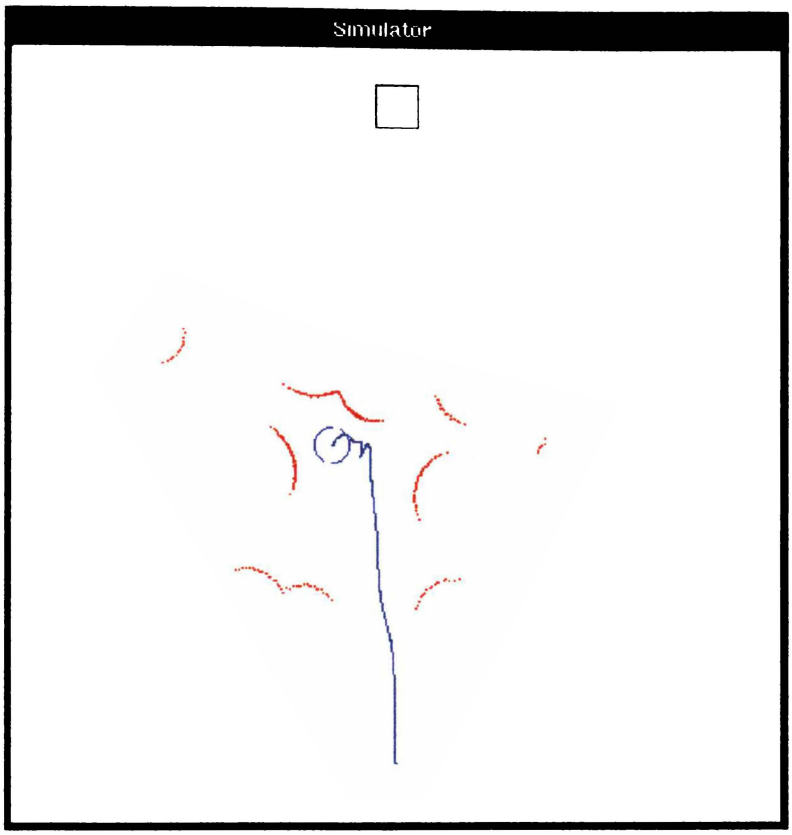

Figure 6-7 The position and trajectory of the AMR after ten seconds of exploring in World No. 135, Obstacle density 15%.

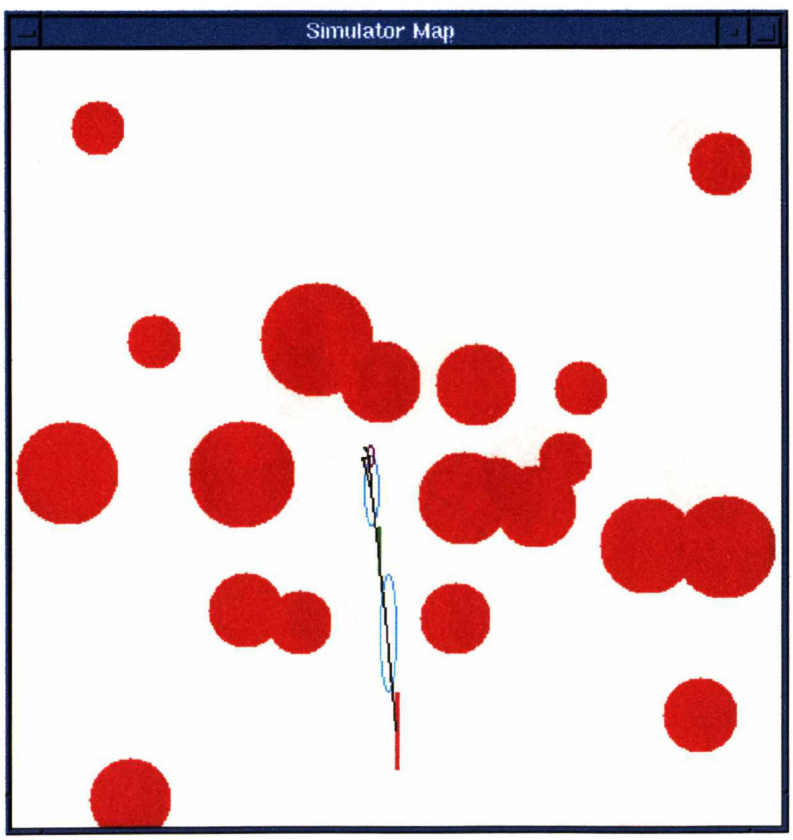

Figure 6-8 The topological map generated by the AMR after ten seconds of exploring in World No. 135, Obstacle density 15%.

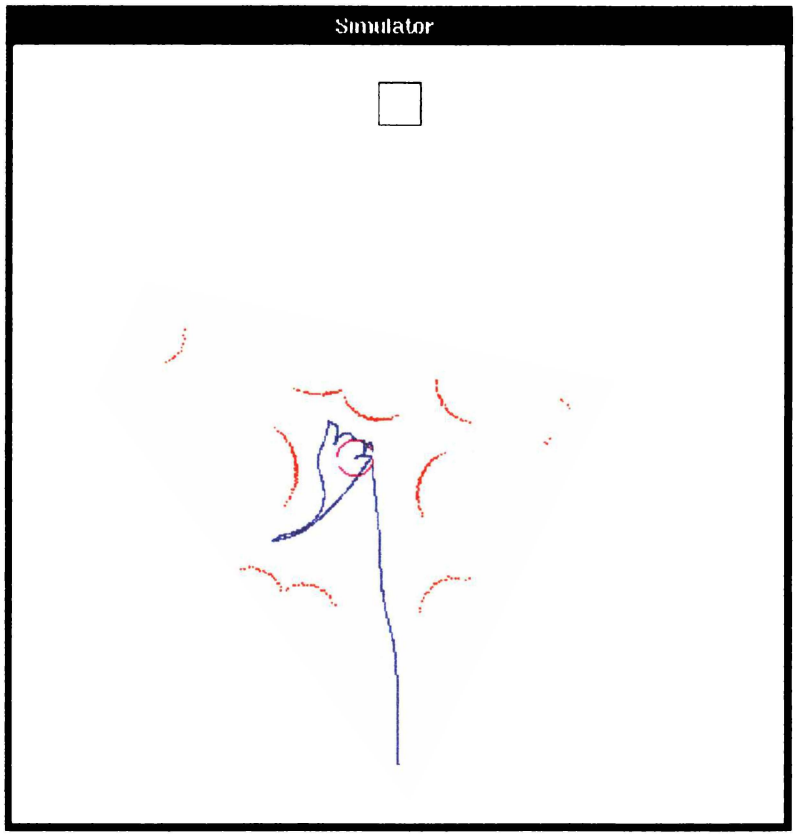

Figure 6-9 The position and trajectory of the AMR after twenty seconds of exploring in World No. 135, Obstacle density 15%.

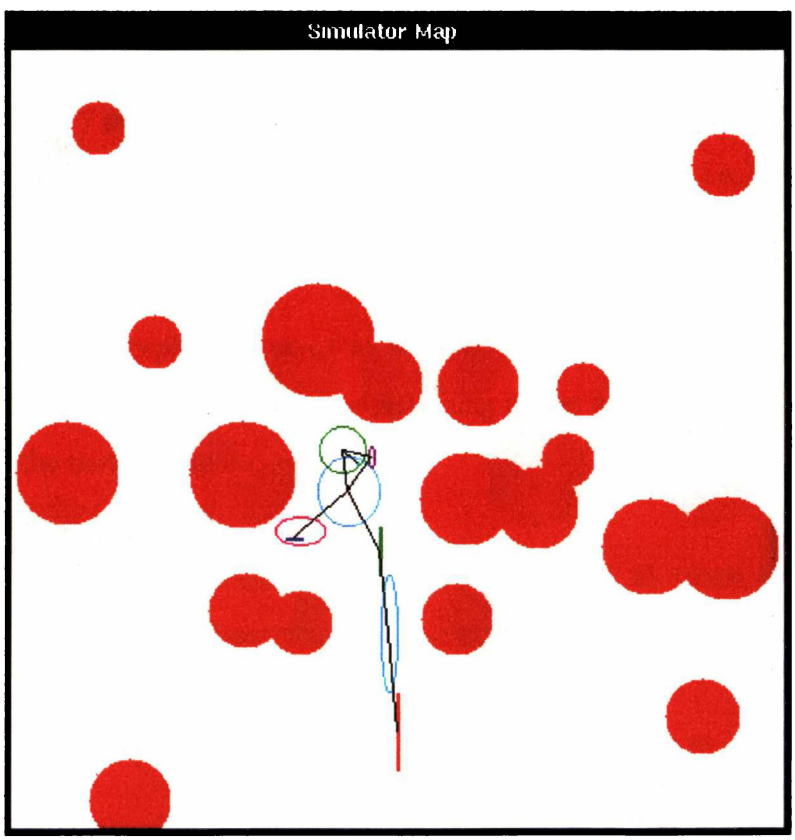

Figure 6-10 The topological map generated by the AMR after twenty seconds of exploring in World No. 135, Obstacle density 15%.

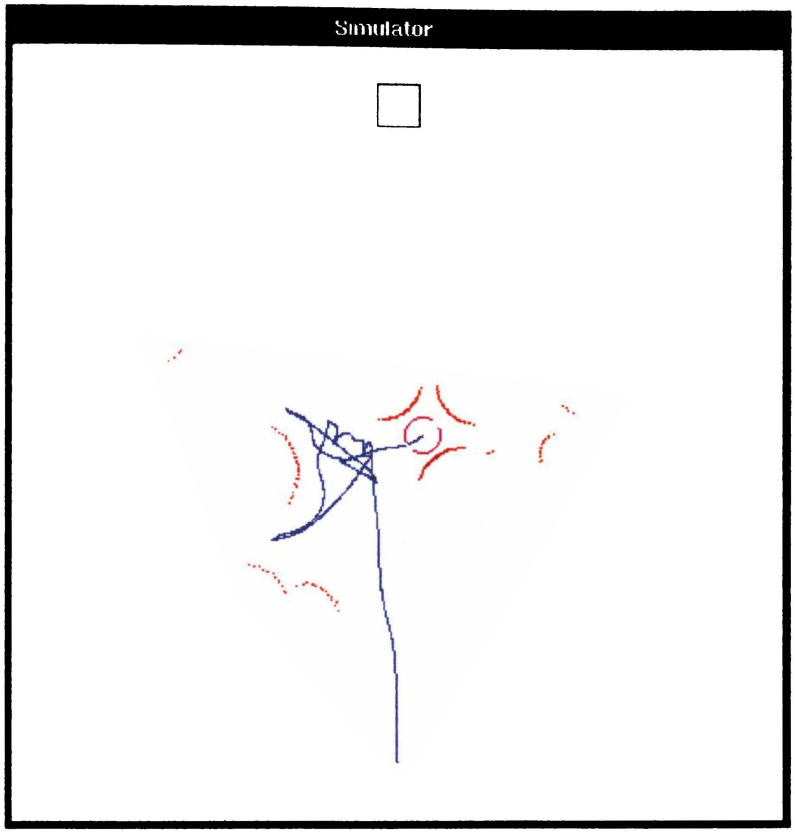

Figure 6-11 The position and trajectory of the AMR after thirty seconds of exploring in World No. 135, Obstacle density 15%.

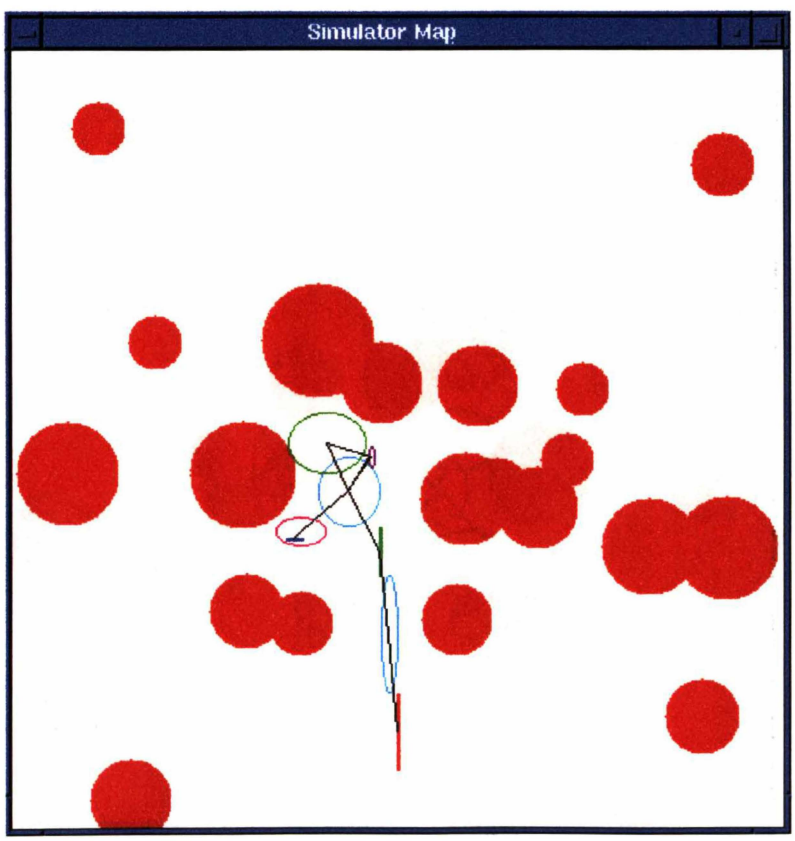

Figure 6-12 The topological map generated by the AMR after thirty seconds of exploring in World No. 135, Obstacle density 15%.

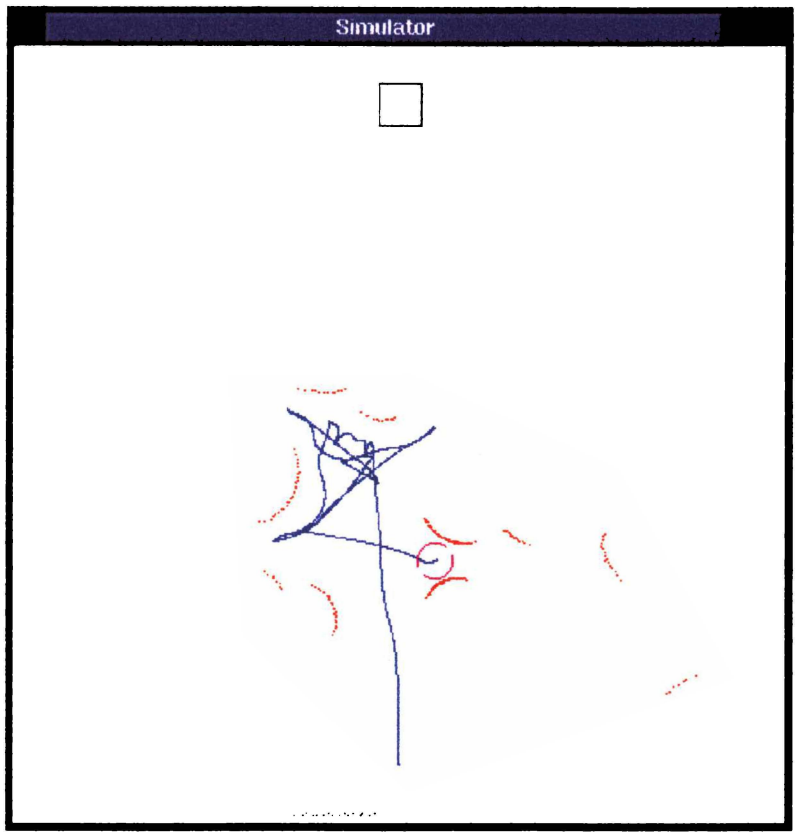

Figure 6-13 The position and trajectory of the AMR after forty seconds of exploring in World No. 135, Obstacle density 15%.

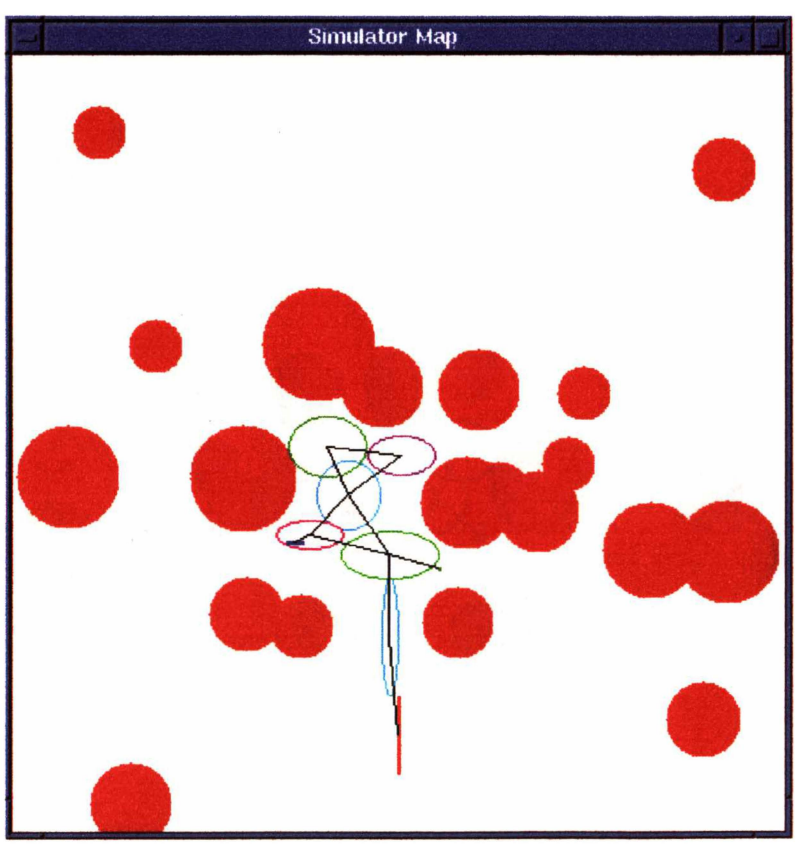

Figure 6-14 The topological map generated by the AMR after forty seconds of exploring in World No. 135, Obstacle density 15%.

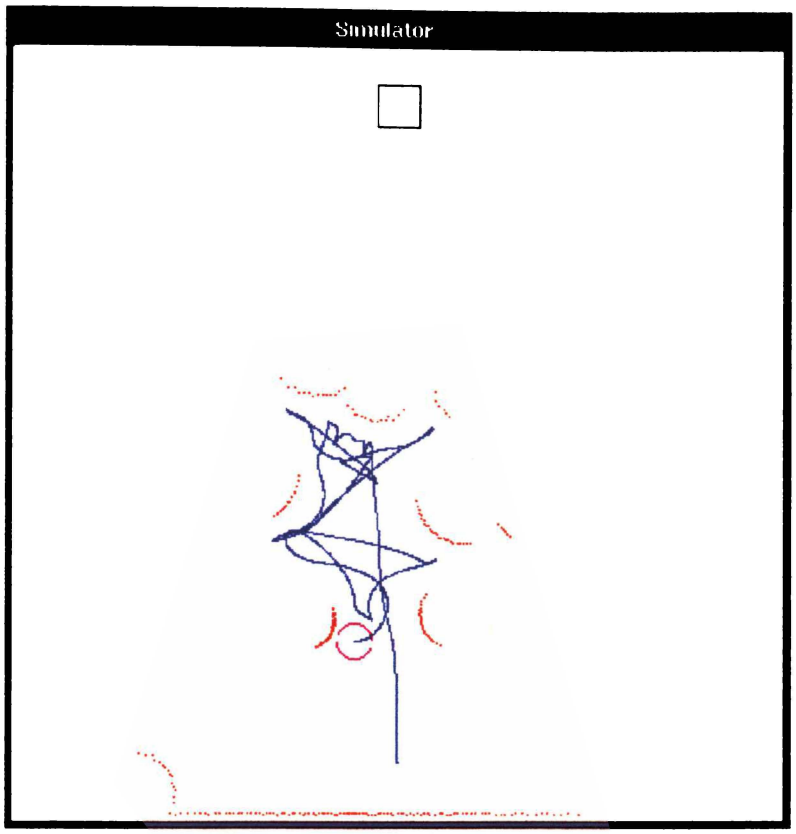

Figure 6-15 The position and trajectory of the AMR after fifty seconds of exploring in World No. 135, Obstacle density 15%.

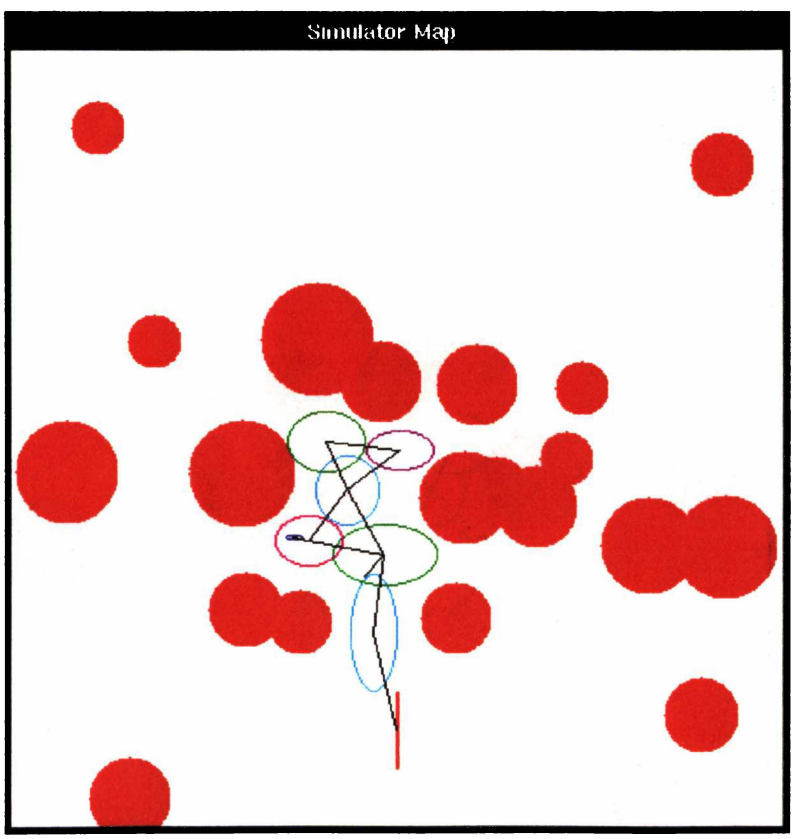

Figure 6-16 The topological map generated by the AMR after fifty seconds of exploring in World No. 135, Obstacle density 15%.

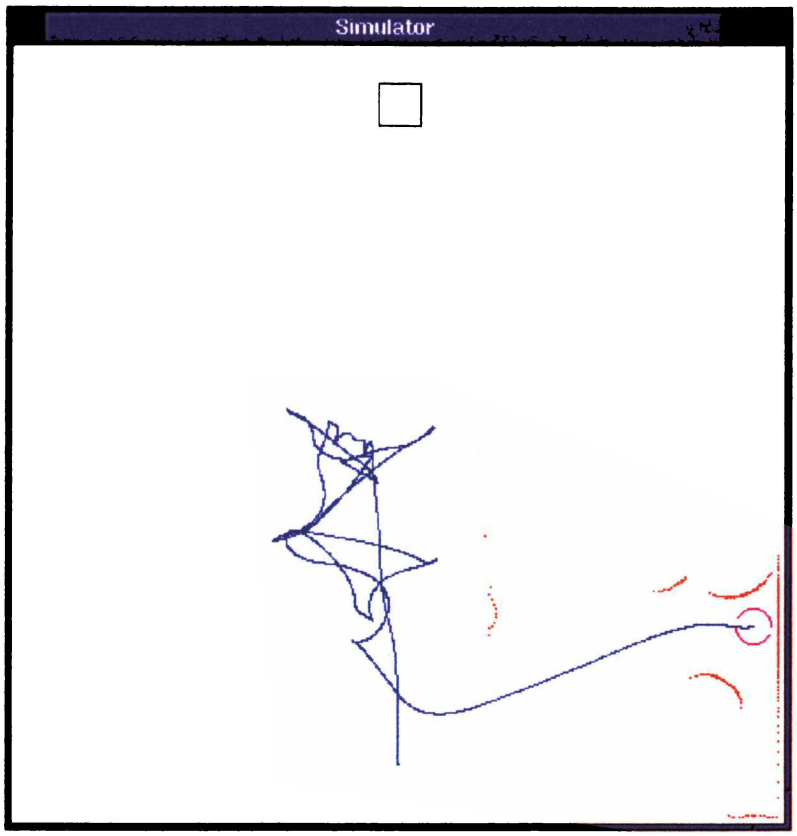

Figure 6-17 The position and trajectory of the AMR after sixty seconds of exploring in World No. 135, Obstacle density 15%.

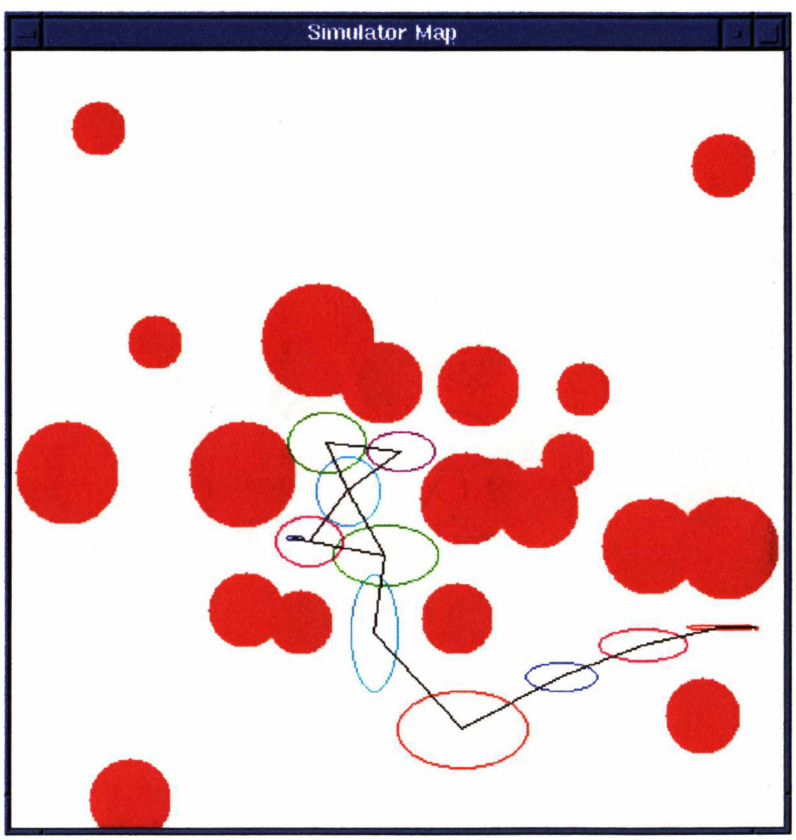

Figure 6-18 The topological map generated by the AMR after sixty seconds of exploring in World No. 135, Obstacle density 15%.

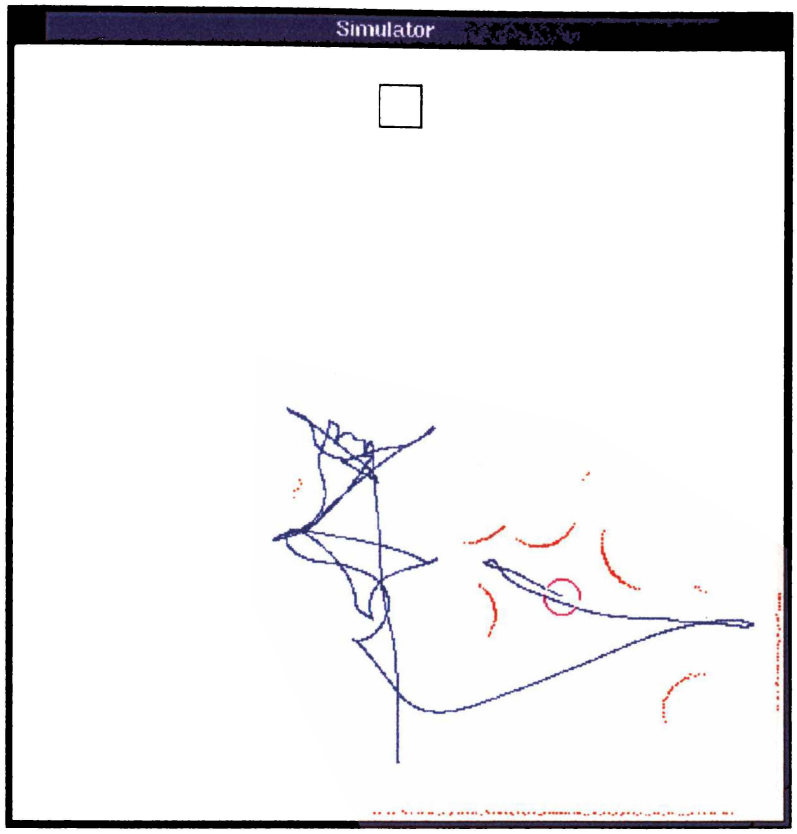

Figure 6-19 The position and trajectory of the AMR after seventy seconds of exploring in World No. 135, Obstacle density 15%.

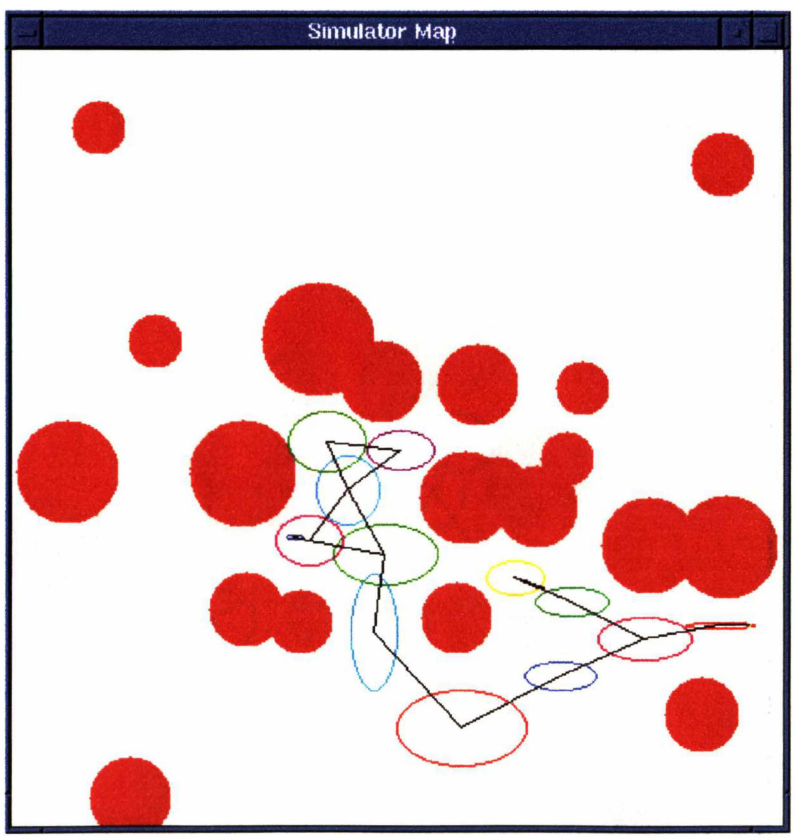

Figure 6-20 The topological map generated by the AMR after seventy seconds of exploring in World No. 135, Obstacle density 15%.

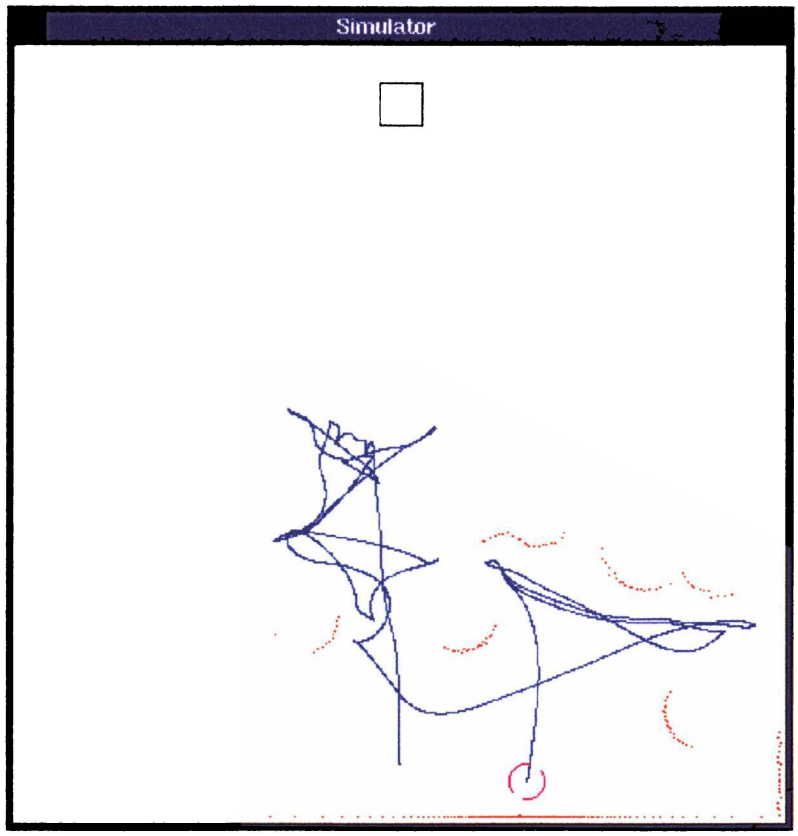

Figure 6-21 The position and trajectory of the AMR after eighty seconds of exploring in World No. 135, Obstacle density 15%.

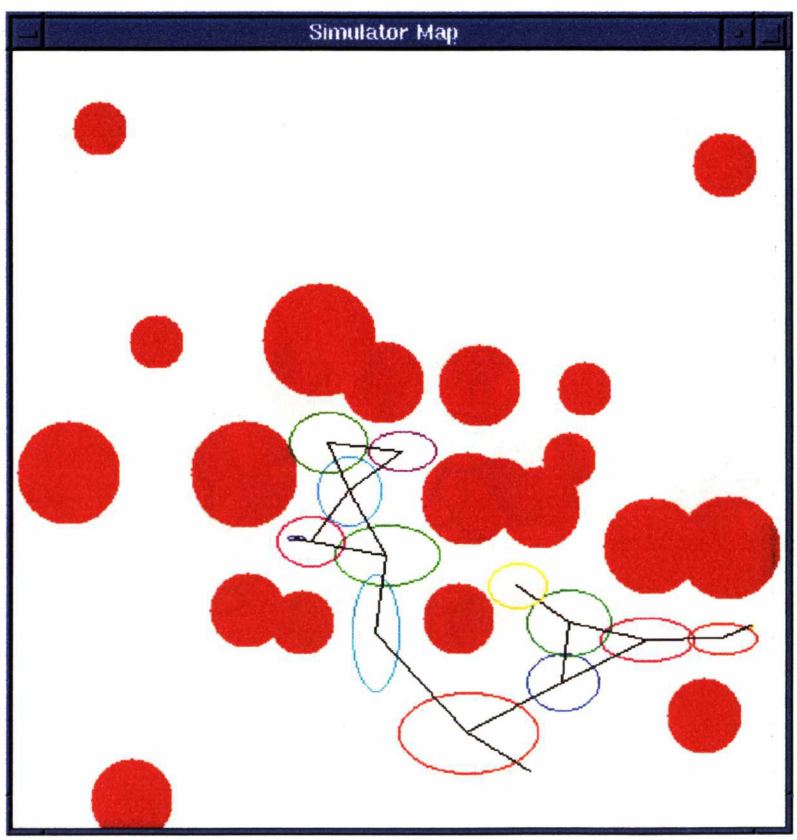

Figure 6-22 The topological map generated by the AMR after eighty seconds of exploring in World No. 135, Obstacle density 15%.

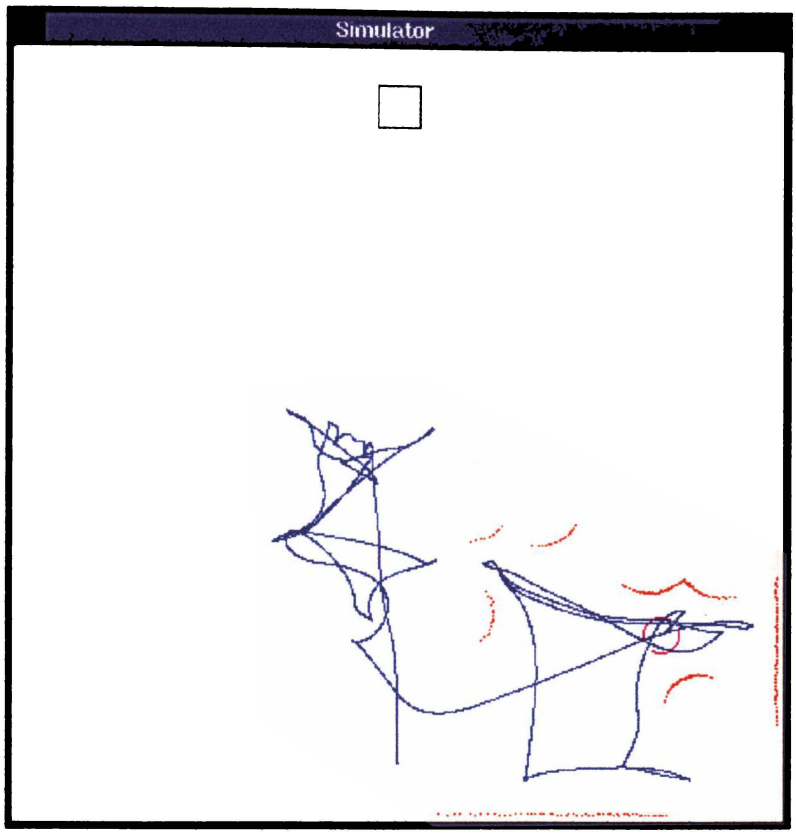

Figure 6-23 The position and trajectory of the AMR after ninety seconds of exploring in World No. 135, Obstacle density 15%.

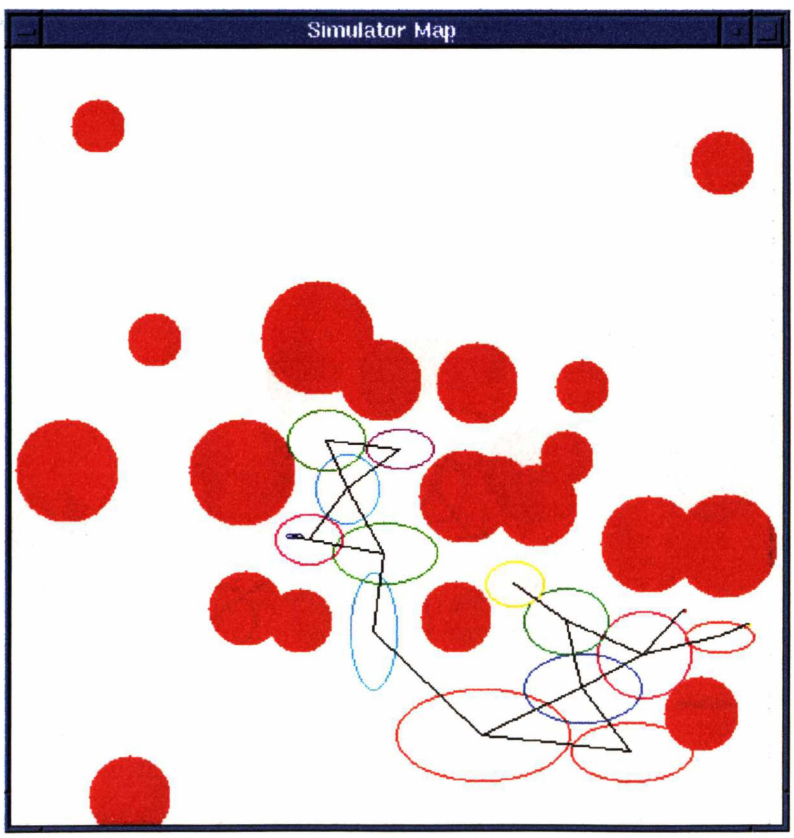

Figure 6-24 The topological map generated by the AMR after ninety seconds of exploring in World No. 135, Obstacle density 15%.

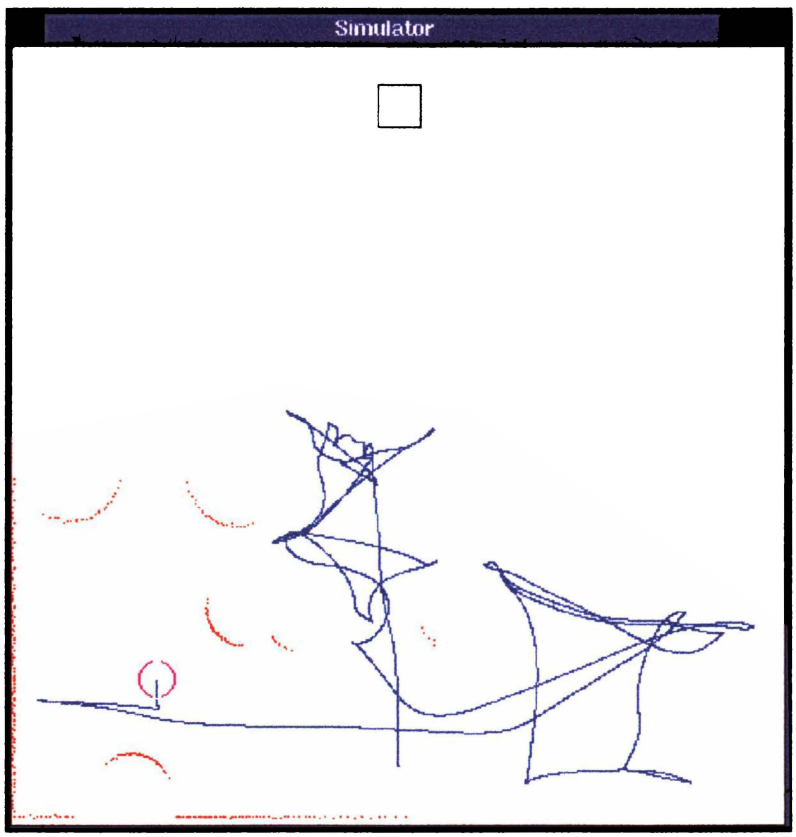

Figure 6-25 The position and trajectory of the AMR after 100 seconds of exploring in World No. 135, Obstacle density 15%.

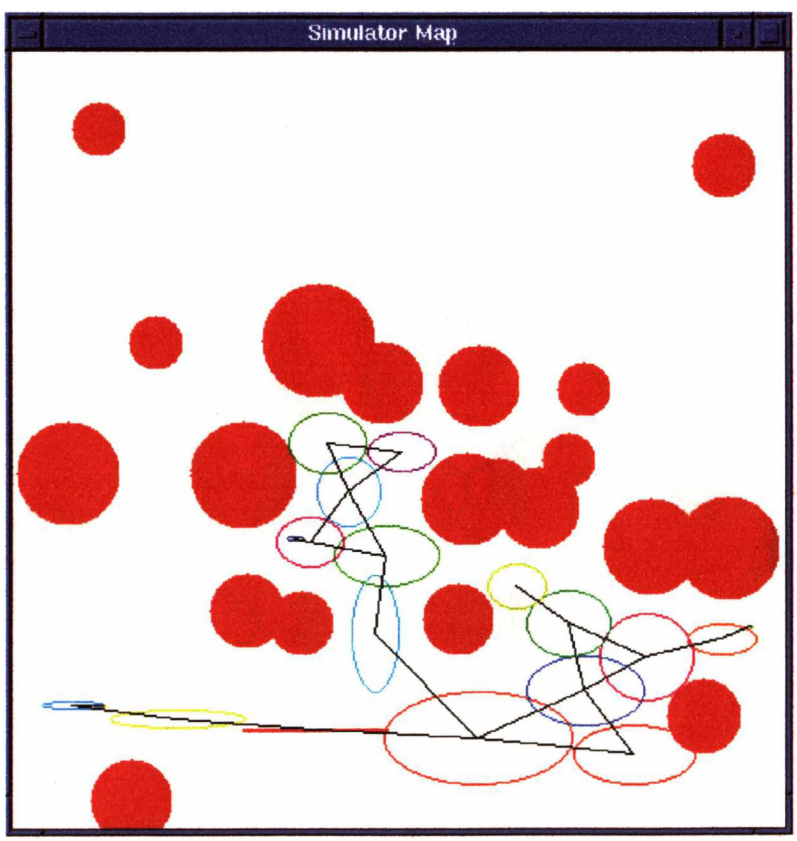

Figure 6-26 The topological map generated by the AMR after 100 seconds of exploring in World No. 135, Obstacle density 15%.

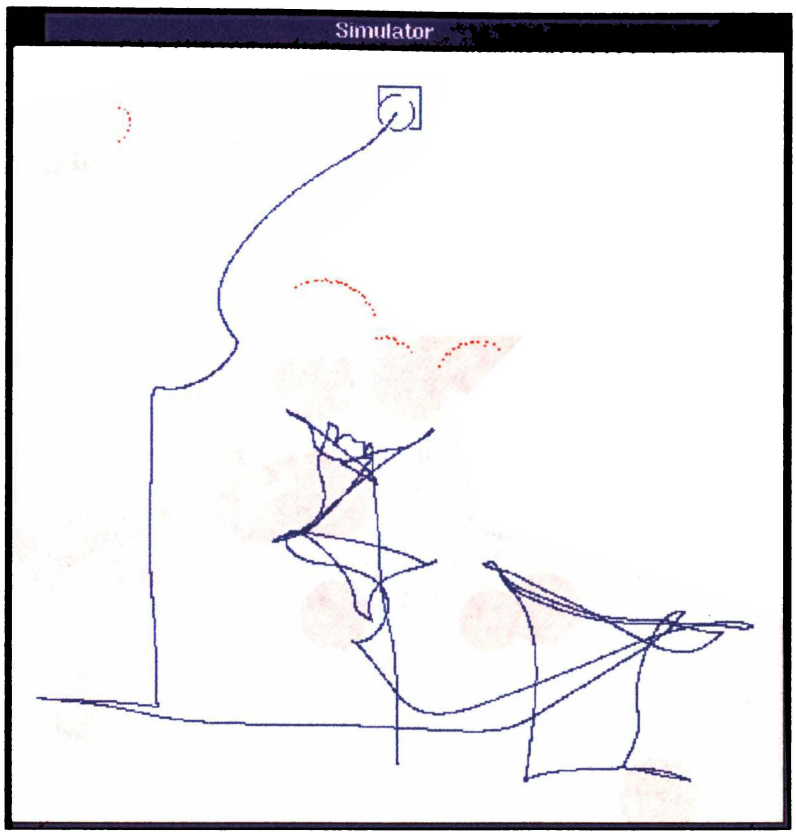

Figure 6-27 The position and trajectory of the AMR after 110 seconds of exploring in World No. 135, Obstacle density 15%.

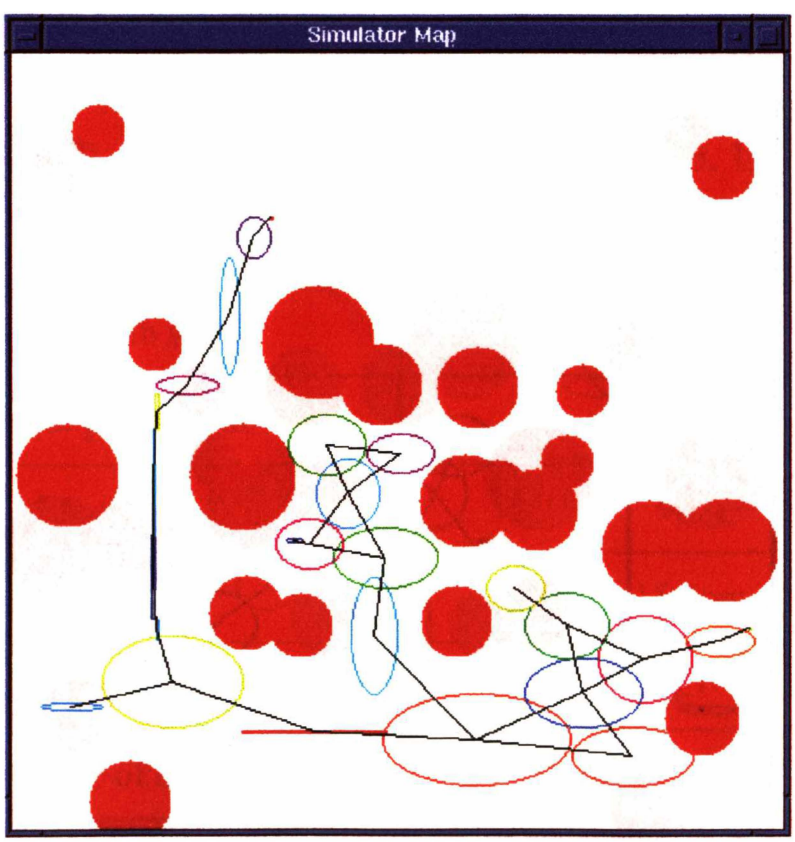

Figure 6-28 The topological map generated by the AMR after 110 seconds of exploring in World No. 135, Obstacle density 15%.

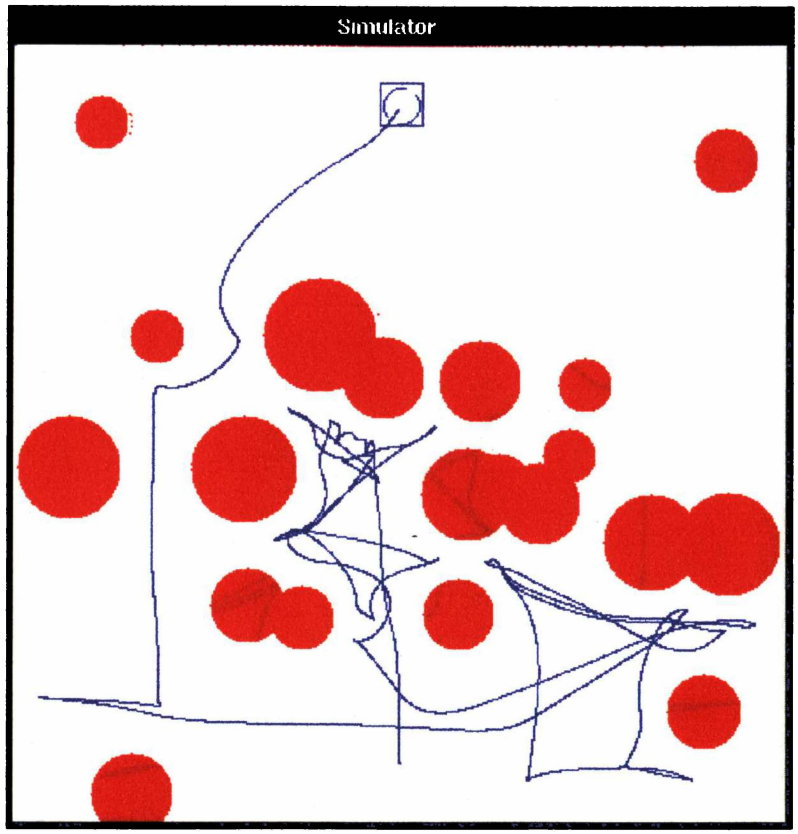

Figure 6-29 The position and trajectory of the AMR after 110.9 seconds of exploring in World No. 135, Obstacle density 15%.

Figure 6-30 The topological map generated by the AMR after 110.9 seconds of exploring in World No. 135, Obstacle density 15%.

AMR performance in World No. 135, Obstacle density 15%	
Optimal path length	7.6m
Actual path length	44.9m
Efficiency	16%
Time taken	110.9sec
Number of nodes generated	26
Average number of links per node	2.23

Table 6-3 The performance of the AMR in World No. 135, Obstacle density 15%.

If the AMR is restarted in the world without re-initialising its memory it should plan a path directly to the goal and navigate directly there. This is shown in Figures 6-31 through 6-34. The performance of the AMR with the map already constructed is shown in Table 6-4.

AMR performance in *mapped* World No. 135 Obstacle density 15%	
Optimal path length	7.6m
Actual path length	7.9m
Efficiency	96%
Time taken	15.0sec
Number of nodes generated	27
Average number of links per node	2.37

Table 6-4 The performance of the AMR in World No. 135, Obstacle density 15%. However, the map constructed in the previous simulation is provided and allows the AMR to navigate directly to the goal.

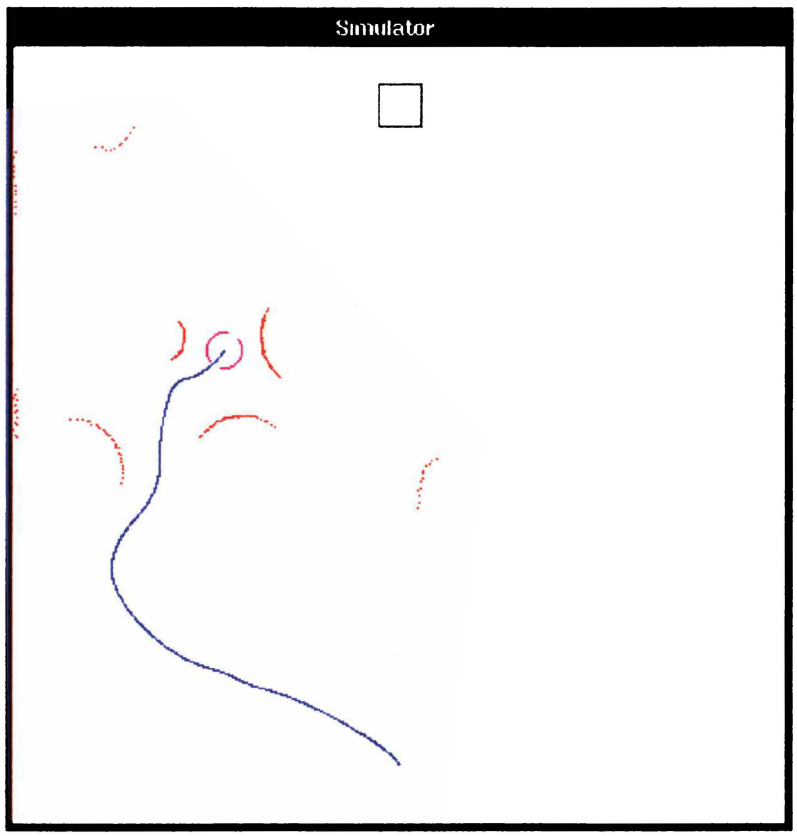

Figure 6-31 The position and trajectory of the AMR after ten seconds of navigating with a map in World No. 135, Obstacle density 15%.

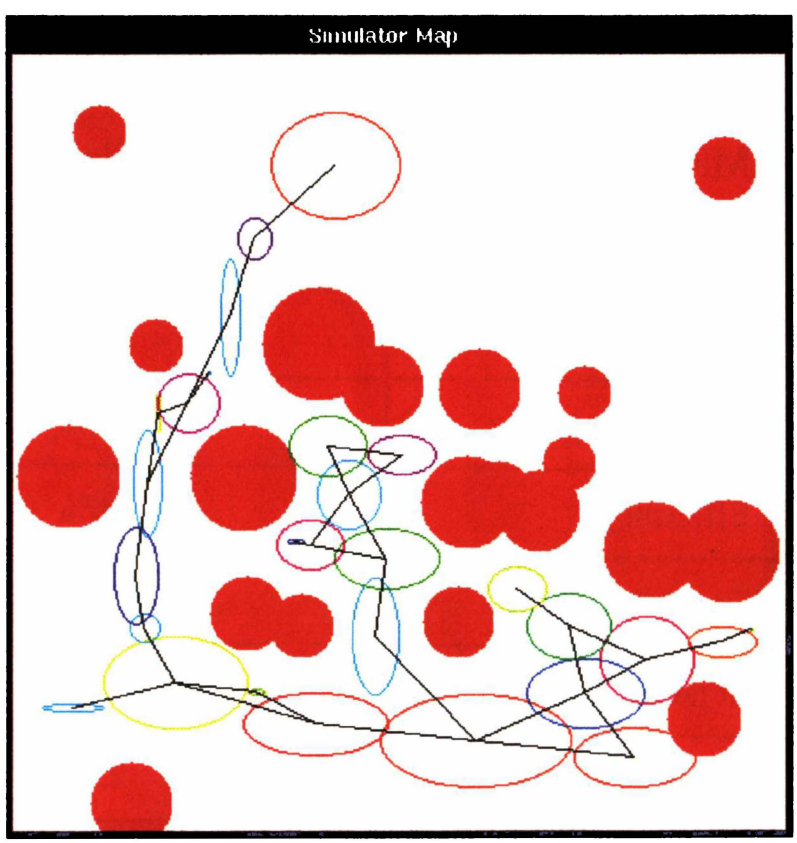

Figure 6-32 The topological map from the last simulation with new information integrated into it from ten seconds of navigating in World No. 135, Obstacle density 15%.

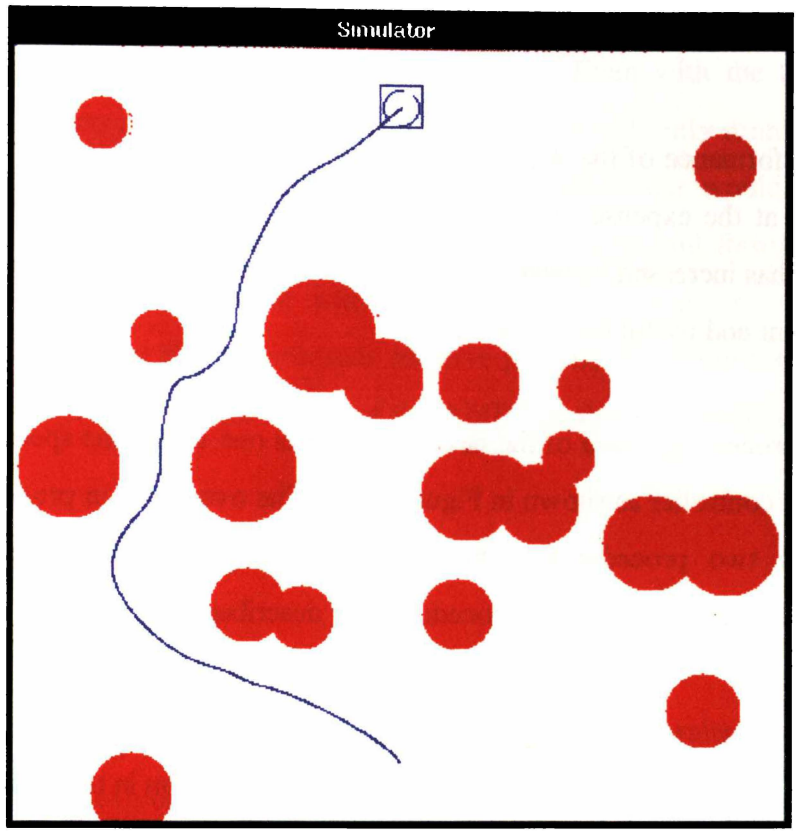

Figure 6-33 The position and trajectory of the AMR after 15.0 seconds of navigating with a map in World No. 135, Obstacle density 15%.

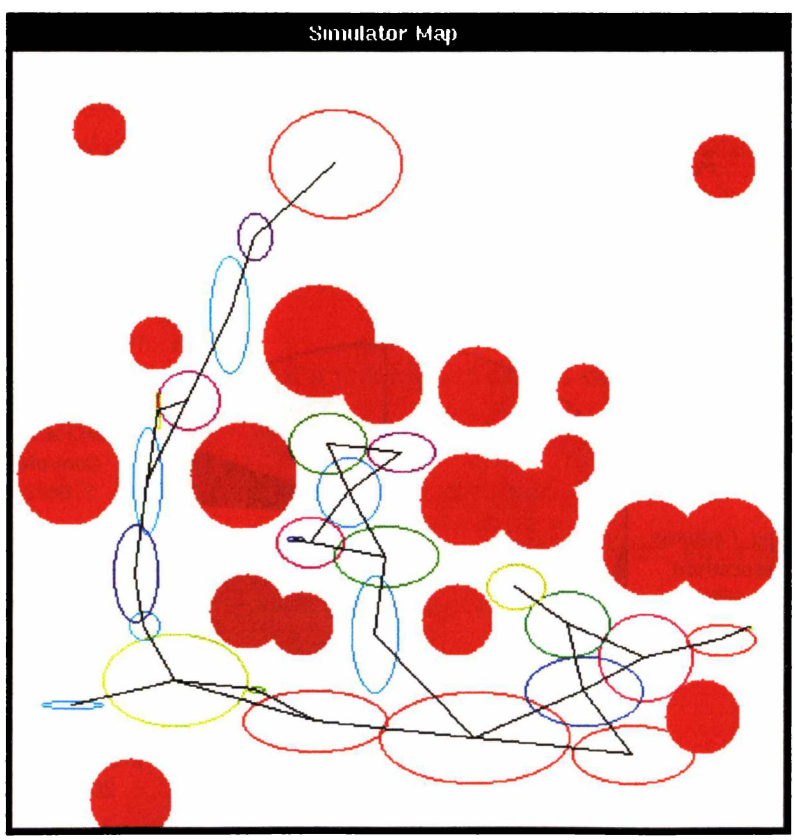

Figure 6-34 The topological map from the last simulation with new information integrated into it from 15.0 seconds of navigating in World No. 135, Obstacle density 15%.

6.3.2 'Reactiveness' of the AMR

While the performance of the AMR has been improved with respect to solving worlds, this has been at the expense of the AMR's 'reactiveness.' The CPU time required to solve worlds has increased because of the algorithms used to transform the sensory data into an efficient and useful representation.

Overall, the processing speed of the new controller is one fourth the speed of the earlier CBR reactive controller as shown in Figure 6-35. The overheads in processing time are consumed by two procedures, namely, the *Resolve_Obstacles* procedure and the *Find_Features* procedure. These procedures are described below:

- *Resolve_Obstacles* This procedure takes the 360 byte array data from the laser range-finder and extracts out the obstacles and their position in the scan.
- *Find_Features* This procedure takes the obstacles computed and determines the smallest vector relationship between various obstacles in the scan and also determines possible exits between obstacles.

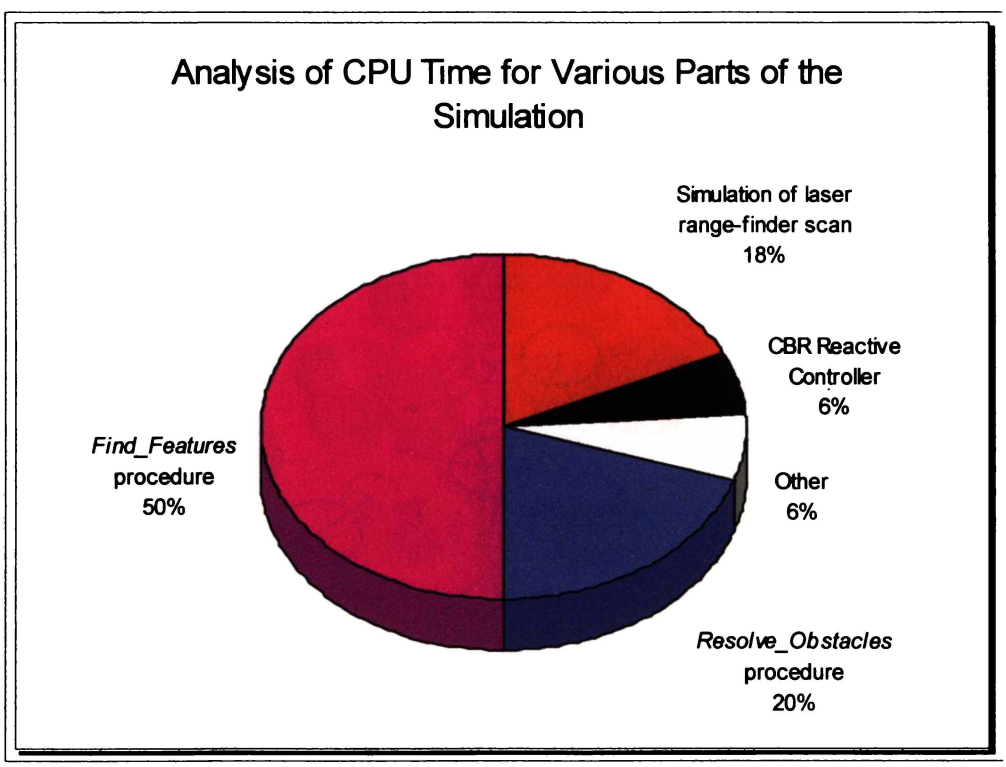

Figure 6-35 Analysis of CPU time for various parts of the simulation.

This simulation takes 187 seconds to complete when running on a DEC Alpha 2000/300 and scanning the environment at 100 hertz. Even with the time taken to simulate the laser range-finder subtracted, the simulation is still only running at 70% of 'real-time.' Halving the scanning rate of the laser range-finder would improve the performance, however, on a physical AMR the *Find_Features* and *Resolve_Obstacles* procedures could be implemented and optimised on a low-power digital signal processing (DSP) chip to provide adequate reactive performance from the AMR. This would not only improve the performance of the AMR but also reduce the data-flow into the core CPU.

6.3.3 Memory Requirements

The new controller requires more memory than the previously developed controllers. While the AMR was exploring World No. 135, Obstacle density 15%, 3.8 megabytes of sensory data were generated. Currently, 318 bytes of memory are required to store each node that the AMR resolves (see Section 6.2.2). For the AMR exploring in this world, twenty-six nodes were resolved before the AMR reached the goal. This equates to only eight kilobytes of memory. However, the structure storing the information of each node is not optimised and it is possible to reduce the size of the structure to 150 bytes without a significant loss in performance, resulting in only four kilobytes required for this simulation.

The amount of memory used to store a topological map is dependent on the size of the nodes resolved which are determined by the complexity of the environment. In the case of the 15% Obstacle density class of worlds, the space to memory ratio equates to approximately three square meters per kilobyte. Again, if the structure storing the nodal information was optimised this space to memory ratio could be doubled.

Chapter 7

7. Conclusion

7.1 Contributions

This aim of this thesis has been to provide quantitative results measuring the performance of various control systems. Four different types of controllers have been developed and tested, beginning with the simplest of reactive controllers through to an adaptive reactive controller with the ability to construct and use topological maps for navigational purposes. In situations where a controller's performance has been inadequate, the cause was determined and a solution has been integrated into the next control system developed.

Both evolutionary and creationist philosophies have been investigated regarding the design technique of such control systems. Evolutionary methods have shown use in generating controllers in domains where the search space of the problem is small. However, in the domain of AMR navigation through complex environments the search space of the problem is too large and the problem becomes intractable. A neural

network with a bit string length of 3640 bits required 3000 CPU hours on a DEC Alpha 2000/300 computer to evolve. The resultant network's performance, while better than that of an adaptive reactive controller, still did not provide robust control and a memory or spatial representation of the AMR's environment was determined as being essential to the problem. Such a system could not be designed through evolutionary techniques so a creationist approach was adopted and a controller with memory was developed.

This memory was provided to the AMR in the form of a topological map of the environment. To provide the AMR with the ability to self-localise itself within the environment, distinguishable features were extracted from the sensory data. The features used were the smallest vectors between pairs of obstacles, and these provided the information to construct the nodes in the map. Each node was linked to each other by the vector relationship between the mid-points of each adjacent node. Exit information for each node and a means of masking the exits that had been explored was developed. The resulting controller was simulated and achieved the desired performance criterion, namely, robust navigation.

7.2 Limitations and Future Work

Currently, the controller developed in Chapter Six can only create accurate topological maps in a static environment. However, the same techniques can still be applied to a topological mapping system that will function in a dynamic environment. There are two main issues to deal with in such an environment. Firstly, moving obstacles need to be detected. If possible, these obstacles should be omitted when constructing a map, as they are transient features. Secondly, there will be semi-permanent obstacles such as furniture that may be moved from time to time and doors that will open and close.

7.2.1 Obstacle Detection

The first issue in adapting this system so as to function in a dynamic environment is determining which obstacles are static and which are dynamic. This could be achieved by comparing successive laser range-finder scans, and by subtracting off the AMR's velocity, compare the positions of the obstacles from each scan to determine if any obstacle is moving [Wilson 97]. The time between scans must be great enough to resolve the movement of an obstacle out of the noise in the laser range-data.

Once a moving obstacle has been detected, it can be masked/ignored when creating or referencing the AMR's map. The obstacle's mask could decay over time so that if it stops moving, the AMR would not think it part of the surrounding static environment until the mask had decayed below some threshold. If the obstacle began moving again then this would reset the mask.

Such a system would require that a list of obstacles be tracked from scan to scan. The following data structure could now represent the obstacles.

```
struct Obstv
{
  bool current;
  float move_mask;
  short bearing, range;
  short bearing_min, bearing_max;
}obstv[15];
```

The pseudo-code from page 81 could be revised as follows to generate the data for the new Obstv structure.

```
short Resolve_Obstacles(unsigned char rangedata[],Obstv obstv[])
{
  //  refer to Section 6.2.1
  short num_obstacles = 0;
  Obstv tmpobstv;
  float decay = 0.01;  // arbitrary value
  for(j=0;j<15;j++)
  {
    obstv[j].current = false;
    if (obstv[j].range != 0) num_obstacles++;
  }
```

```
  for(j=0;j<360;j++)
  {
    if (abs(rangedata[j+1] - rangedata[j]) > robot.diameter))
    {
      // Start of an obstacle…
      tmpobstv.bearing_min = Find_bearing_min();
      tmpobstv.bearing_max = Find_bearing_max();
      tmpobstv.bearing = Find_bearing();
      tmpobstv.range = Find_range();
      tmpobstv.current = true;
      short oldindex = MatchObstacleWithOldScan(obstv,tmpobstv);
      if (oldindex != -1)
      {
        if (Moving(tmpobstv,obstv[oldindex])
        {
          tmpobstv.move_mask = 1;
        }
        else tmpobstv.move_mask = obstv[oldindex].move_mask - decay;
        obstv[oldindex] = tmpobstv;
      }
      else
      {
        tmpobstv.move_mask = 0;
        obstv[num_obstacles] = tmpobstv;
        num_obstacles++;
      }
    }
  }
  num_obstacles = SortAndRemoveNOTCurrentObstacles(obstv);
  for (j=0;j<num_obstacles;j++)
  {
    if (!obstv[j].move_mask && find_large_angle_change(obstv[j]))
    {
      split_obstacle_into_two();
    }
  }
  return num_obstacles;
}
```

7.2.2 Variable Confidence Links and Exits

The second issue highlighted is that of semi-permanent obstacles. In these situations the AMR will construct a topological map based on certain obstacles only to return to the same area to find that one or more obstacles have been moved or removed or there may be the addition of one or more new obstacles. A door being closed would be a common example of such a situation. This would result in the removal of a path that once existed to another part of the AMR's map.

Given the current system, the AMR would mark the exit that once existed as being blocked off and would never consider it again in future path planning even if the door was opened some time later. However, if there were already a link established through

to a node on the other side of the door, then the AMR would continually try to navigate along the link to the node. While the reactive controller would prevent the AMR from colliding with the door, the AMR would fail to achieve its goal even if there was another longer path still available to it.

To overcome this problem, the node's links and exits could have a variable confidence level as is used by Yamauchi and Beer in their adaptive place network [Yamauchi & Beer 96]. In their system, whenever a link is traversed successfully, the confidence of the link is increased to $\lambda_{link} + (1 - \lambda_{link})c_{old}$ where λ_{link} is the link learning rate (0.5 in their experiments) and c_{old} is the old confidence. However, if the link was not traversed successfully within a given time, the new link confidence is decreased to $(1 - \lambda_{link})c_{old}$.

Using such a system would allow old links to be removed if they are blocked by obstacles or if the destination node no longer exists (due to changes in the environment). Obviously within a changing environment there will be new nodes being created in place of old nodes. These old nodes can then be deleted once there are no links connecting them to any other node.

7.2.3 Sensor Selection

Further research into the field of laser range-finders would be necessary before low-cost AMRs could be realised. Another approach to building a low-cost AMR would involve using different sensors to generate topological maps of the environment using the same techniques described in this thesis. The choice of distinguishable features from the sensory data is dependent on the sensory system used. However, a robust topological map can still be created if the appropriate features can be extracted. As mentioned in Section 6.2.1, these features should be resolvable independent of the viewing perspective of the AMR.

Appendix A
Case Library

As discussed in Section 4.2.3, the simulator was configured with a case library consisting of eight cases. Their descriptions and associated parameter values follow below. In the list of control parameters, the noise persistence is shown in tenths of a second while the sensing distance is in centimetres. These units were selected so as to avoid using real numbers.

1. Clearfield In an open environment, the AMR should pay no attention to obstacles (since there will not be any), increase the goal gain, lower the noise gain and noise persistence.

ENVIRONMENT INFORMATION	
Parameter	**Value**
Clutter	0
Density	0
Proximity	-1
Wander	0
Clear-to-goal	-1
Senses-goal	-1
Goal-nearby	-1
Nomovement	0

CONTROL PARAMETERS	
Parameter	**Value**
Goal gain	6
Obstacle gain	3
Noise gain	0
Noise persistence	250
Sensing distance	250
Temporary goal gain	0
Exit goal gain	0

2. Ballooning When there are relatively few obstacles, the AMR attempts to swing around them in a wide way (increase obstacle gain).

ENVIRONMENT INFORMATION	
Parameter	**Value**
Clutter	2
Density	35
Proximity	0
Wander	0.05
Clear-to-goal	1
Senses-goal	-1
Goal-nearby	0
Nomovement	0

CONTROL PARAMETERS	
Parameter	**Value**
Goal gain	6
Obstacle gain	7
Noise gain	0
Noise persistence	250
Sensing distance	250
Temporary goal gain	0
Exit goal gain	2

3. Squeezing When there are many obstacles, the AMR attempts to find a path by squeezing between obstacles (lower obstacle gain, increase goal gain and exit goal gain).

ENVIRONMENT INFORMATION	
Parameter	**Value**
Clutter	4
Density	60
Proximity	1
Wander	0.15
Clear-to-goal	-1
Senses-goal	-1
Goal-nearby	-1
Nomovement	0

CONTROL PARAMETERS	
Parameter	**Value**
Goal gain	7
Obstacle gain	3
Noise gain	0
Noise persistence	250
Sensing distance	250
Temporary goal gain	0
Exit goal gain	6

4. Hugging When there are many obstacles and the AMR is currently faced with an obstacle directly in its path, it attempts to hug the side of the obstacle as it makes its way around it.

ENVIRONMENT INFORMATION	
Parameter	**Value**
Clutter	6
Density	54
Proximity	1
Wander	0.45
Clear-to-goal	-1
Senses-goal	-1
Goal-nearby	-1
Nomovement	0

CONTROL PARAMETERS	
Parameter	**Value**
Goal gain	7
Obstacle gain	4
Noise gain	1
Noise persistence	250
Sensing distance	250
Temporary goal gain	2
Exit goal gain	0

5. Shooting Regardless of the number and size of the obstacles surrounding the AMR, if the system sees its goal and there are no obstacles in the way, it adopts an extreme version of the Clearfield strategy and goes directly to it.

ENVIRONMENT INFORMATION	
Parameter	**Value**
Clutter	5
Density	45
Proximity	0
Wander	0.1
Clear-to-goal	-1
Senses-goal	1
Goal-nearby	-1
Nomovement	0

CONTROL PARAMETERS	
Parameter	**Value**
Goal gain	6
Obstacle gain	5
Noise gain	0
Noise persistence	250
Sensing distance	250
Temporary goal gain	0
Exit goal gain	0

6. Random The AMR raises the noise gain, obstacle gain and exit goal gain, leaves the goal gain at a medium level, and wanders for a period of time.

ENVIRONMENT INFORMATION	
Parameter	**Value**
Clutter	5
Density	75
Proximity	1
Wander	0.65
Clear-to-goal	-1
Senses-goal	-1
Goal-nearby	-1
Nomovement	1

CONTROL PARAMETERS	
Parameter	**Value**
Goal gain	5
Obstacle gain	6
Noise gain	5
Noise persistence	250
Sensing distance	250
Temporary goal gain	0
Exit goal gain	4

7. Repulsion In certain situations, the system considers moving away from the goal for a period of time. If, for example, the AMR senses the goal and there are obstacles surrounding it, it may decide to 'back away' for a distance before attempting to get to the goal. This is accomplished by setting the goal gain to a negative amount and raising the exit goal gain.

ENVIRONMENT INFORMATION	
Parameter	**Value**
Clutter	3
Density	80
Proximity	1
Wander	0.9
Clear-to-goal	-1
Senses-goal	-1
Goal-nearby	-1
Nomovement	1

CONTROL PARAMETERS	
Parameter	**Value**
Goal gain	-3
Obstacle gain	5
Noise gain	2
Noise persistence	250
Sensing distance	150
Temporary goal gain	0
Exit goal gain	6

8. Wallcrawling If there is an obstacle the system cannot seem to get around by Hugging and no progress is being made, the AMR will try to follow along the wall of the obstacle to try to get around it. This is accomplished by raising the temporary goal gain and lowering the goal gain, and allows the AMR to travel away from the goal while following the wall of an obstacle.

ENVIRONMENT INFORMATION	
Parameter	**Value**
Clutter	0
Density	0
Proximity	0
Wander	0
Clear-to-goal	0
Senses-goal	0
Goal-nearby	0
Nomovement	0

CONTROL PARAMETERS	
Parameter	**Value**
Goal gain	2
Obstacle gain	3
Noise gain	0
Noise persistence	250
Sensing distance	45
Temporary goal gain	9
Exit goal gain	0

Bibliography

[Arkin 89] R. Arkin. 1989. "Motor schema-based mobile robot navigation." *International Journal of Robotics Research,* 8:92-112.

[Brightwell 93] G. Brightwell. 1993. "Design and construction of micromouse." MSc Thesis, Department of Physics and Electronic Engineering, University of Waikato.

[Brooks 86] R. Brooks. 1986. "A robust layered control system for a mobile robot." *IEEE Journal of Robotics and Automation*, 2:14-23.

[Brooks 89] R. Brooks. 1989. "A robot that walks: Emergent behavior from a carefully evolved network." *Neural Computation*, 1:253-262.

[Connell 87] J. Connell. 1987. "Creature building using the subsumption architecture." *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-87)*, pp. 1124-1126.

[Engelson & McDermott 92] S. Engelson and D. McDermott. 1992. "Error correction in mobile robot map learning." *Proceedings 1992 IEEE International Conference on Robotics and Automation*, pp. 2555-2560.

[Fox et al. 97] D. Fox, W. Burgard, and S. Thrun. 1997. "The dynamic window approach to collision avoidance." *IEEE Robotics and Automation,* 4(1).

[Goldberg 89] D. Goldberg. 1989. "Genetic algorithms in search, optimization and machine learning." Reading, MA: Addison-Wesley.

[Harvey et al. 92] I. Harvery, P. Husbands, and D. Cliff. 1992. "Issues in evolutionary robotics." *Cognitive Science Research Paper,* Serial No. CSRP 219, University of Sussex.

[Holland 75] J. Holland. 1975. "Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control and artificial intelligence." Ann Arbor, MI: University of Michigan Press.

[Kolodner 92] J. Kolodner. 1992. "Case-based reasoning." Morgan Kaufmann, San Mateo, CA, 1992.

[Kortenkamp & Weymouth 94] D. Kortenkamp and T. Weymouth. 1994. "Topological mapping for mobile robots using a combination of sonar and vision sensing." *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94),* pp. 979-984.

[Koza 92] J. Koza. 1992. "Genetic programming: On the programming of computers by means of natural selection." MIT Press.

[Kuipers & Byun 91] B. Kuipers and Y. Byun. 1991. "A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations." *Journal of Robotics and Autonomous Systems,* 8:47-63.

[Maes 91] P. Maes. 1991. "Situated agents can have goals." *Designing Autonomous Agents,* Pattie Maes, Ed., Cambridge, MA: MIT Press, pp. 49-70.

[Mataric 92] M. Mataric. 1992. "Integration of representation into goal-driven behavior-based robots," *IEEE Transactions on Robotics and Automation,* 8:304-312.

[Mataric 94] M. Mataric. 1994. "Interaction and intelligent behavior." Technical Report AI-TR-1495, AI-Lab, MIT.

[Meeden 94] L. Meeden. 1994. "Towards planning: Incremental investigations into adaptive robot control." PhD Thesis, Department of Computer Science, Indiana University.

[Moravec & Elfes 85] H. Moravec and A. Elfes. 1985. "High resolution maps from wide angle sonar." *Proceedings of the IEEE International Conference on Robotics and Automation*, St. Louis, MO, pp. 116-121.

[Moriarty & Miikkulainen 94] D. Moriarty and R. Miikkulainen. 1994. "Efficient reinforcement learning through symbiotic evolution." Technical Report AI94-224, Department of Computer Sciences, University of Texas at Austin.

[Pearl 88] J. Pearl. 1988. "Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference." San Mateo, CA: Morgan Kaufmann.

[Pierce & Kuipers 94] D. Pierce and B. Kuipers. 1994. "Learning to explore and build maps." *Proceedings Twelfth NCAI,* AAAI, pp. 1264-1271.

[Ram et al. 92] A. Ram, R. Arkin, K. Moorman, and J. Clark. 1992. "Case-based reactive navigation: A case-based method for on-line selection and adaptation of reactive control parameters in autonomous robotic systems." Technical Report GIT-CC-92/57, College of Computing, Georgia Institute of Technology.

[Rosenblatt & Payton 89] J. Rosenblatt and D. Payton. 1989. "A fine-grained alternative to the subsumption architecture for mobile robot control." *Proceedings of the International Joint Conference on Neural Networks.*

[Saffiotti et al. 93] A. Saffiotti, E. Ruspini, and K. Konolige. 1993. "Blending reactivity and goal-directedness in a fuzzy controller." *Proceedings of the IEEE International Conference on Fuzzy Systems,* San Francisco, CA, pp. 134-139.

[Smith 87] H. Smith. 1987. "Data structures – Form and function" Harcourt Brace Jovanovich, Publishers, pp. 294-295.

[Thrun 93] S. Thrun. 1993. "Exploration and model building in mobile robot domains." *Proceedings of the ICNN-93*, San Francisco, CA, pp 175-180, IEEE Neural Network Council.

[Thrun & Bucken 96] S. Thrun and A. Bucken. 1996. "Learning maps for indoor mobile robot navigation." Technical Report CMU-CS-96-121, School of Computer Science, Carnegie Mellon University.

[Thrun et al. 98a] S. Thrun, A. Bucken, W. Burgard, D. Fox, T. Frohlinghaus, D. Hennig, T. Hofmann, M. Krell, and T. Schmidt. 1998. "Map learning and high-speed navigation in RHINO." AI-based Mobile Robots: Case studies of successful robot systems. Cambridge, MA: MIT Press.

[Thrun et al. 98b] W. Burgard, A. Cremers, D. Fox, D. Hahnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun. 1998. "The interactive museum tour-guide robot." *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-94)*

[Wilson 97] A. Wilson. 1997. "Mapping of an environment using a laser-scanning system." MSc Thesis, Department of Physics and Electronic Engineering, University of Waikato.

[Yamauchi & Beer 96] B. Yamauchi and R. Beer. 1996. "Spatial learning for navigation in dynamic environments." *IEEE Transactions on Systems, Man, and Cybernetics,* Special Issue on Learning Autonomous Robots, Vol. 26, No. 3, pp. 496-505.