Title A layered control architecture for mobile robot navigation

Name Jiancheng Qiu

A Layered Control Architecture for Mobile Robot Navigation

by

Jiancheng Qiu

A Thesis submitted to the Univeristy Research Degree Committee in fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Robotics

Faculty of Science, Design and Technology
University of Luton

January, 1998

Abstract

This thesis addresses the problem of how to control an autonomous mobile robot navigation in indoor environments, in the face of sensor noise, imprecise information, uncertainty and limited response time. The thesis argues that the effective control of autonomous mobile robots can be achieved by organising low level and higher level control activities into a layered architecture. The low level reactive control allows the robot to respond to contingencies quickly. The higher level control allows the robot to make longer term decisions and arranges appropriate sequences for a task execution.

The thesis describes the design and implementation of a two layer control architecture, a task template based sequencing layer and a fuzzy behaviour based low level control layer. The sequencing layer works at the pace of the higher level of abstraction, interprets a task plan, mediates and monitors the controlling activities. While the low level performs fast computation in response to dynamic changes in the real world and carries out robust control under uncertainty.

The organisation and fusion of fuzzy behaviours are described extensively for the construction of a low level control system. A learning methodology is also developed to systematically learn fuzzy behaviours and the behaviour selection network and therefore solve the difficulties in configuring the low level control layer.

A two layer control system has been implemented and used to control a simulated mobile robot performing two tasks in simulated indoor environments. The effectiveness of the layered control and learning methodology is demonstrated through the traces of controlling activities at the two different levels. The results also show a general design methodology that the high level should be used to guide the robot's actions while the low level takes care of detailed control in the face of sensor noise and environment uncertainty in real time.

Acknowledgements

First and foremost I would like to thank my director of studies, Mick Walters, for three years of advice and friendship. Mick gave me the freedom to pursue my own ideas and made it possible for me to concentrate on this research. He also managed to bring me back to the track when my ideas went too wild. Most recently, he gave me the pushes I needed to get everything finished and written down. Without his valuable guidance and full support, it would be impossible to complete the work in three years.

Second, I would like to thank Dr. Roger Harvey, who unfortunately left us half year ago. I have benefited immensely from his rigorous scientic research method and his encouragement. It was mainly for his effort that I had chances to present some of the research results in international conferences under difficult circumstances. He will be sorely missed.

Third, I am also extremely grateful to Dr. John Chisholm, my external supervisor for supervising my study, reading and commenting on an early draft of my thesis, especially as he has not been so well over the last year. His comments helped shape the final draft thesis greatly.

Special thanks go to Dr. Kemal Ahmed and Mr. David Wilkingson for their diligence in reading the draft of this thesis and their valuable discussions.

I would like to thank my examination commitee. Dr. Huosheng Hu served as the external examiner and Dr. Alfred Vella as the internal examiner. Their invaluable comments and suggestions gave me the guidances in completing the final thesis in a short time.

I would also like to thank Steve Arkurst, Elaine Walsh and Faculty of Design and Technology for providing me with a PhD studentship and valuable teaching experience during the course of this work.

This thesis would not be possible without a great deal of advice and help from many people. I cannot possibly list them all here, and I apologize for any I might have missed.

Thanks to my family for their love and support. I would especially like to thank my father, who instilled in me, at a very young age, a love of learning. Dad, thanks for all the stories, homeworks and love. Dad, rests in peace.

Finally, I would like to thank my wife, Hua. Marrying her was the smartest thing I ever did. I am forever indebted to her, for giving me so much love, hope, and support.

Table of Content

List of Figures

List of Tables

Chapter 1 Introduction

This thesis is about the navigation control for a mobile robot with possible applications in indoor environments. As a way to identify and understand some of general mobile robot navigation problems, the thesis will use driving a car as an equivalent example and keep the analogy between driving a car and mobile robot control in the following discussions.

Driving a car is a difficult task which takes people a considerable amount of time to master. The car's velocity and heading must be constantly controlled to avoid collisions while at the same time leading towards a destination. The driver must respond quickly to unexpected contingencies like children running across a road, while at the same time planning ahead to decide, for example, whether to get gas at this exit or wait for the next one.

Unlike a car driven by human, we will deal with autonomous mobile robots which move about on their own. An autonomous mobile robot navigating about in the world might correspond better to a person walking towards a destination to complete a task than driving on a road. However, driving seems to have more physical similarities with controlling a mobile robot as they all involve mechanical actuators. Many problems encountered in driving to a destination are similarly presented in mobile robot control. The mobile robot does not correspond to the driver alone, but rather to the car and driver as an integrated system. The driver can be considered as corresponding to the robot's computer and sensory system, the car to the robot's actuators.

Driving is hard for three fundamental reasons. First, the amount of time available to decide what to do is limited. If a child runs in front of a car, the driver only has a few seconds to react. Even processes which take a longer time such as planning a route from a map have time limits. It does a person no good to spend much of his vacation time to plan an optimal route for the beach. Second, the world is largely unpredictable: animals run in front of cars, roads are closed for repair, traffic signs have been removed, maps are approximate and imprecise and not always up-to-date, and the weather may change suddenly. This usually makes it impossible to plan a complete and reliable course of action in advance. Actions have to be taken at the moment the situations arise which cannot be perceived by a plan. Third, sensors and actuators are not perfect. Sensory information includes noise and sometimes may be totally wrong. Mechanical actuators are imprecise and can fail. Wheel slips on the floor. Error accumulation can turn a series of small imprecision into failure. A small direction error can lead to a big position displacement.

These problems are fundamental problems and they cannot be solved by engineering techniques so far. No matter how powerful a computer people build, a finite amount of time will allow only a finite amount of computation. No matter how good a sensor may be there is always information that it cannot deliver because the relevant situation is hidden behind a wall or around a corner. No matter how many theories people may use in describing our physical world, some aspects of the world still cannot be predicted. Limited computation and sensing capabilities make the problems even worse.

This thesis addresses the problem of how to design a control mechanism for an indoor autonomous mobile robot that will allow it to reliably operate in the real world in the face of these problems, namely:

. Sensor Noise;

. Limited computation time;

. Uncertainty, approximity and imprecision.

The behaviour of the robot should display three characteristics. Firstly it should be reactive; that is, it should be able to respond quickly to unexpected contingencies, such as collisions with obstacles. Second, it should be task-oriented; that is , the robot should choose the right sequences of actions which lead it to achieve its task goals. Third, the robot's behaviour should be robust and reliable. It should be able to move on its own and be confident that it will survive while achieving its goals under sensor noise, uncertainty and imprecision.

1.1 Issues in Real-World Navigation

Examining further our car driving analogy can help to identify the sorts of processes that are needed get a mobile robot to move around in the real world, whether it be across office building, construction floor or across country. The following sections highlight the main themes which the thesis will address.

1.1.1 Local Sensor Data vs. World Model

Navigation, whether it be a hallway or on a motorway, involves local sensory input, a plan and a map in order to control a physical system progress towards the navigation goal. How to use local sensory input and a world model varies, but in humans, using either extreme affects their performance. Human can't drive with their eyes closed. However, even with their eyes open, most drivers will have trouble finding way in London when there is no knowledge of roads beyond what can be seen directly.

Local sensor data are necessary for a number of reasons. First, the world is a dynamic and unpredictable place. Cars on streets and people in hallways move in unpredictable ways, and in order to navigate without collisions it is necessary to monitor the surroundings constantly. Even if the world were perfectly static, sensors would still be necessary. Because of mechanical uncertainty, it is impossible to build a system which navigates reliably without some sort of feedback. Therefore, local sensory information is necessary, at least, to provide feedback for the low level control mechanism driving the robot.

A world model is often needed in addition to local sensor data. Sometimes the information necessary to decide what to do to achieve a goal is simply not available to local sensors. It is similar to the way that one relies on instructions and maps, (that is, plans and world model), to decide the direction to drive when there is no such information locally.

1.1.2 Dealing with Sensor Noise

While sensory information is necessary, it is often noisy or wrong. It is not unusual that people sometimes misread traffic signs. Robots are particularly vulnerable to noise because their sensors are typically of much lower quality than those human possess. Due to erroneous sensor data or a changing world, the robot's world model may be wrong as well. The robot's actions would certainly be wrong based on the incorrect world model. An effective control system should be able to detect and correct erroneous information in the world model to guide the robot's actions. At the same time, the robot should still work properly under the noisy local sensor data.

1.1.3 Dealing with Imprecise Information

A mobile robot requires some prior knowledge in obtaining a navigation goal. Such knowledge is perhaps a map which may not be accurate because of the changing world. Some features of an environment can be modified, a passage way indicated in the map may be blocked and no longer in use, a door the robot is about to cross is actually positioned half a meter left to the one in the map. Things can even be worse when imprecise information is mingled with noise sensor data. The robot control system should have a way to accommodate such imprecise knowledge and still be able to navigate successfully.

1.1.4 Fast Response Time

All computations in navigation are time-limited, but some are more limited than others, such as avoiding emerging collisions. When a driver steps on his brakes in front of an emerging car from a side road there are typically only a few seconds available before colliding. An autonomous mobile robot must operate at the pace of the world and respond quickly to unexpected situations in order to survive in the real world. A robot that can do the right thing is useless if it does it too late. Slow response will certainly result in unfavourable navigation actions and even damage to the robot.

On the other hand, sometimes there are decisions that can and do take longer time to make. The decision to go down M1 rather than A418 from Luton to Bristol requires many minutes of poring over a map. However, even this decision has a deadline; it would not be very useful to decide to use A418 once already started on M1. For a robot, it must do the whole task in a reasonable time to be cost-effective.

1.1.5 Plans

Much of the early work on mobile robot planning usually make the assumption that the world can be predicted infinitely into the future. High level decisions are often made under such assumptions. A plan is constructed as a computer program, a step-by-step algorithm which may be executed by an autonomous robot. The plan tries to account for every detail of mobile robot interactions and instructs every movement of the robot. Such a method may be reasonable in an engineered environment completely under the control of the planner. In a complex , dynamic real world, an assumption of predictability is no longer held. It is not possible to predict the future in detail. When a planner cannot project future states of the world in complete detail, it cannot construct a detailed plan or the plan constructed is not going to work. Many detailed actions in the plan will have to be left unspecified until the situation in which they are required arises and the details of the world state can be determined by direct observation. This means that a plan must be sketchy. I simply cannot tell if a duck will run in front of my car when I set off to visit my friends. When it does, I step on the brake, change the gear and stop the car to let the duck get away. The sequences of actions are not planned before I set off. They belong to a sketchy plan I have in my mind: go to Old Bedford Road; drive till Six Form College; turn left to New Bedford Road.... The sequences of the actions have been embedded in my driving skills and are brought into action when the situation requires. Similarly, a robot acting in a complex, dynamic world must be controlled with flexible plans that allow many low level actions to depend on the actual situations, encountered at execution time. Constructing a control mechanism to support a sketchy task plan and low level detailed actions is the central topic of the thesis.

1.2 Actions and Behaviours

It is common in the AI literature to decompose actions hierarchically. High level actions made by high level decisions consist of sequences of low level actions. The hierarchy ends at the lowest level with primitive actions which can be directly executed by the robot actuators. "Turning the wheel" is a good example of a primitive actions for a person driving down the road. Similarly, it is fairly straightforward to "turn the wheel" on a mobile robot because the "wheel" is usually directly connected to a motor over which the robot exercises direct control. However, "turn the wheel" cannot be executed by the motor unless the direction and amount of angle to turn are also provided. Here, two example primitive actions involving turning are "turn left" and "turn right". Others may include "increase speed", "decrease speed", "raise arm", "lower arm", etc. For a mobile robot, a primitive action cannot be initiated without its control parameters calculated and provided by a processing module.

We can see a primitive action is invoked to create a desired control behaviour in the robot movement and serve the specific purpose. It is natural to use the term "behaviour" to refer to this specific purpose processing module which creates the parameters and invokes primitive actions. When a driver turns the wheel left and presses on brake, he implicitly exhibits collision avoidance "behaviour" to prevent colliding with an approaching car from the right. Driving to London is an activity which requires a collection of driving behaviours. To keep moving on the M1 at certain speed ranges, a driver will step on the accelerator pedal constantly. To avoid collision, the driver will change speed and direction. To take a correct position, the driver will follow traffic lanes. These driving behaviours fall in two different categories: reactive behaviours and purposeful behaviours. Driving to London is an activity which needs the support of purposeful driving behaviours, such as following correct lanes. However, following a correct lane alone cannot guarantee a safe arrival. There is a constant need to avoid collision and change speed during the course. These driving behaviours do not belong to the activity "Driving to London" but are parts of one's innate driving knowledge and can be activated whenever required. The "Driving to London" activity needs the support of both driving behaviours. Similarly, a robot control system can be better organised, based on special purpose controlling behaviours. Task-oriented behaviours support the purpose of completing the robot task, while reactive behaviours take care of local interactions. These two types of behaviours need to be composed to resolve the conflicting control actions. The major part of this thesis will be about how to organise these specific purpose control behaviours.

1.3 High Level Activity and Low Level Activity

While low level activities can directly control a robot, they have to be guided by high level control activities. I cannot drive to my friend's home in Bristol without figuring out the approximate steps how to get there. The sketchy plan is produced by carefully reading a road map and simple instructions from my friend. I have to follow these steps during my driving while constantly engaging in controlling the car. These two types of activity lead me to a safe arrival. However, these activities have a number of different characteristics. A high level activity is initiated by a high level decision. High level activities can also include computational processes which produce decisions. Low level activities carry out a goal provided by a high level activity. Driving down the M1 to Junction 6A is the first step of my going to Bristol and involves a number of my driving behaviours; Keep Moving, Avoid Collision, Follow Lane. A high level activity need more computation time to decide and is less time critical. A low level activity has to produce fast results and control the robot in real time. A high level activity has a slower pace but its effect lasts longer. Once I commit myself to be on the M1, Driving down the M1 to Junction 6A will last more than 10 minutes. By contrast, once on the M1, pressing on the brake commits me to go slower only for a few seconds. I can also quickly regain my speed at any time.

High level activity and low level activity are two parts of an integrated problem. They can interact with each other and the environment in very complex ways. Driving to Bristol and stepping on the brake to avoid hitting the front car near Junction 6A are related somehow. I would not have had to step on the brake if I had not been driving to Bristol. On the other hand, my stepping on the brake was not simply a direct consequence of driving to Bristol either. If the car had not slowed so suddenly, I could have driven to Bristol without pressing on the brake at that point. Pressing on the brake does help me to arrive in Bristol safely later. Controlling a mobile robot needs careful organisation of these different types of activity. Their differences in processing time and functions need to be taken into account in designing the robot control architecture. A layered model is adopted in this thesis as a more efficient control approach.

1.4 Imprecision and Heuristic Control

Humans all share one property, we are not good at describing our behaviour with mathematical models. Instead, we use heuristic experience or knowledge and symbolic terms to describe our daily activities and make decision. "On approaching the T junction, slow down" is a typical example of such a heuristic rule to instruct a student driver. We can well understand and master such kind of control knowledge. Such knowledge is often imprecise or vague and impossible or very difficult to be represented in a vigorous mathematical model. See the above example of "approaching the T junction" again, the instructor may advise his driving student when to apply the brake to slow down. Would he say, "Begin braking 74 feet from the T junction"? or would his advise be more like "Applying the brake pretty soon"? The answer is the latter, of course. The former instruction is too precise to be implemented. Looking at a similar control rule for a collision avoidance behaviour of a mobile robot, the rule “if an object is less than 2m in the front, then reduce the speed to 20mm/s” would be much more difficult to be realised than the rule “if an object is close in the front, then reduce the speed to very slow” because of noisy sensor, wheel error and environment uncertainty. In driving, humans use many such heuristic rules and imprecise knowledge to effectively control a car. Our correct behaviours of controlling a car are implicitly built on such linguistic rules, which are derived from empirical observations and heuristic knowledge. If there is a car in front, slow down. If a car approaches from the front right, turn to the left to avoid collision. These are rudimentary control knowledge. In controlling a mobile robot, one of the key problems is how to provide a method for handling such kind of heuristic knowledge, while at the same time coping with the imprecision, uncertainty and sensor noise which occur in the real world.

1.5 Learning

An intelligent system should learn or adapt. Learning means change, either the change of control parameters or the organisation of the control system. It is difficult to design a complete control system without the need for further modifications and improvement. A new driver will still need to learn. How to drive on motorway, how to drive in fog conditions, etc. are all the new things he needs to master. Through continuous learning, one can be more certain to drive safely. For a complex control system like a mobile robot, learning is even more important. There are two main reasons. First, a mobile robot is built by human. Humans have shortcomings, either their knowledge of controlling a mobile robot is not complete, or they are unable to postulate all aspects of the control problems. The worse thing is that humans will make mistakes. Learning is a process of change which allows such inherited mistakes and incompleteness to be exposed and changed. Second, the environments used to demonstrate the controlling of a mobile robot are limited. The real world is largely unpredictable. Conditions and situations presented in one environment may not be available in another one and vice-versa. This causes the problem that some aspects of controlling problems cannot be investigated because the conditions are not present. Learning should be introduced to overcome, or at least reduce, such difficulties.

1.6 Outline of a Two Layer Control Architecture

The proceeding sections have described some problems and intuitions about the natures of mobile robot control. This thesis is mainly about how to translate these intuitions into a computational mechanism for controlling an autonomous mobile robot, namely indoor navigation. This section gives an overview of how this has been done.

To summarise briefly, the central topic of the thesis is that a mobile robot navigation system should be supported by a layered control architecture. High level activity and low level activity have different computational requirements and need to be organised differently. The low level activity works at real time and involves reactive and purposeful actions which can be supported by special purpose computational modules, called behaviours. Some of the behaviours are responsible for achieving goals assigned by high level decisions. Others are required to interact with the dynamic real world and help achieve high level tasks. These behaviours rely on local sensors and goal information and perform fast computation. They work together to complete tasks given by high level decisions in the face of sensor noise, uncertainty and imprecision. Fuzzy logic control is exploited to build the two types of behaviours in the low level control layer. With the flexibility and symbolic natures of fuzzy control rules, heuristic control knowledge is effectively organised into the low level control mechanism of a mobile robot.

On the other hand, this low level activity alone is inadequate to navigate a mobile robot. It needs the guidance from a high level control activity. Navigation involves the execution of a task plans which consists of a sequences of steps for the robot to complete. A task plan is better to be organised as a sketchy plan with many details of control actions taken care of by the low level control layer. A high level activity organises the correct sequences of controlling activities in the low level control layer, monitors the progress of the task execution and even performs high level planning when necessary. Planning will not be discussed in the thesis. It is assumed that task plans have already been produced by a planning system or human. A Reactive Action Package(RAP)[Firby89] like control structure is used as basic blocks for organising high level activity.

A two layer control architecture, MARCO[†], consisting of a task template(modified RAP) based higher level control layer and a fuzzy behaviour-based low level control layer, is proposed. The two-layer control architecture belongs to the lower two levels of a so called three layer architecture, deliberative/sequence/reactive[Hasemann95]. The sequencing layer initiates, monitors and terminates the controlling activities in the low level control layer, while the low level control layer executes tasks initiated by the sequencing layer. The two layers co-operate to complete a navigation task with the sequencing layer being in charge.

This thesis also describes the way to effectively organise a low level control layer for indoor navigation, specifically based on indoor environment features. A learning methodology is discussed to automatically learn the low level control layer and reduce the efforts and difficulties involved in the design of such a control structure.

1.7 Summary

This section gives a brief review of the main points of the thesis.

1.7.1 The Problem

This thesis addresses the problem of how to organise an effective control mechanism for autonomous mobile robot navigation in a real world environment in the face of

[†] Mobile Autonomous Robot COntroller.

sensor noise, uncertainty and limited response time in a way which is reactive, robust and task-directed.

1.7.2 The Assumption

A control mechanism can be based fundamentally on a layered architecture. A high level control layer can be based on a control structure which can sequence the controlling activities in the low level control layer. The low level control layer can be based on special purpose behaviours to support the task execution assigned by the high level control layer. The high level control layer organises the correct sequences of task execution while the low level control layer performs fast control actions in the face of sensor noise, uncertainty and imprecision.

A robust, reliable system for controlling an autonomous mobile robot in the real world can be obtained through effective learning processes.

1.7.3 Approach

A two layer control architecture, MARCO has been developed. This architecture includes a high level sequencing layer and a low level control layer.

The computational structures for implementing the architecture, MARCO, are studied in detail. Two heterogeneous structures, a fuzzy behaviour and a task template are defined and used as the basic building blocks of the two layers.

A simple-to-complex multistage learning methodology has been developed to automatically learn a low level control layer.

The implementation of the control architecture is demonstrated by controlling a simulated robot in performing two indoor tasks. One is a concrete slab finishing task and the other is a building security patrolling task. The robot's ability to efficiently sequence the controlling activity according to task plans and perform robust goal-directed control actions under sensor noise, approximate and imprecise information is assessed.

1.8 Thesis Outline

This thesis is divided into seven chapters. Chapter 1 is the Chapter that you are currently reading. It presents an informal descriptions of the mobile robot navigation and control problems upon which the rest of the thesis is based.

Chapter 2 reviews related work in mobile robot control and justifies the choice of a multi-layer architecture and fuzzy logic control method.

Chapter 3 describes the organisation of a two layer control architecture, MARCO and defines the data structures of the two layers' basic building blocks, fuzzy behaviour and task template.

Chapter 4 presents the organisation of a fuzzy behaviour-based low level control layer. The two major parts include the fuzzy behaviour organisation based on the sphere of influence of environment features and the behaviour selection network.

Chapter 5 describes a simple-to-complex multistage learning methodology in order for the low level control layer to learn, including learning individual fuzzy behaviours and behaviour selection network.

Chapter 6 presents two experiments using the MARCO control system to perform two tasks in simulation. The detailed traces are discussed to demonstrate the effectiveness of the control architecture.

Chapter 7 summarises and presents discussions and conclusions of the MARCO architecture and suggests some directions for future research.

Chapter 2 Review of Mobile Robot Control Architecture

2.1 Introduction

Mobile robotics research can be marked by the first appearance of autonomous mobile machine, Shakey, which was built at the Stanford Research Institute in 1968 [Nilsson69]. Shakey worked in a very carefully-engineered environment. Its navigation was helped with the special artificial features in the environment. Although it was not a successful mobile robot, Shakey encouraged further research in mobile robotics. In the seventies, the Stanford Cart[Moravec83], a wheel-driven mobile robot was developed as a testbed for indoor and outdoor navigation in an unknown environment. It was only capable of slow motion(1 meter every 10~15 minutes) and was not a success. Another project running at the same time was the "Mars Rover" project at JPL[Cox90]. It was stopped due to the lack of efficient computing and sensing technology. Probably the first success in mobile robots was Hilare developed at LAAS, France[Giralt90]. The robot was capable of perception, self-navigation, position estimation, path planning and obstacle avoidance. However, because of the length of time needed to obtain sensor data, Hilare had to spend most of its time sitting still. The most successful aspects of the project were the application of traditional sensing and AI technologies. The success of Hilare project attracted more attention to mobile robotics during the eighties. Many mobile robots were developed based on Hilare system architecture [Moravec88] [Thorpe90] [Weisbin89] [Crowley87]. This trend of the development over 15 years was mainly based on traditional sense-model-plan-act approach. Despite a lot of effort, few of these autonomous robots can carry out tasks robustly and reliably in the real world.

Controlling autonomous mobile robots is difficult for three fundamental reasons[Gat92]. First, the time available to decide what to do is limited. A mobile robot must operate in the time domain of its environment. Second, many aspects of the world are unpredictable, making it impossible to plan a complete course of actions in advance. Third, sensors cannot provide complete and accurate information about the environment. These are fundamental problems which cannot be eliminated by engineering methods. Throughout the years of building robot control systems a number of robot control architectures have been created to enable a mobile robot to work in face of these problems. Some of the more commonly known architectures are reviewed here.

2.2 Traditional Approach

The traditional Sense-Model-Plan-Act control architecture was first highlighted in the Hilare project[Giralt90]. A mobile robot control system is organised based on a single execution pipeline of information processing, proceeding from sensing, world modelling, planning to action. The control framework exists essentially in any control system. The extent to which these processes are instantiated in particular systems varies greatly. Specifically, the amount of deliberation, i.e. modelling and planning, can be strongly emphasised as in expert system control or totally neglected as in mechanical control systems. In mobile robot control, the sense-model-plan-act approach assumes a largely static or at least predictable world assumption between sensing and acting. This is a rather invalid assumption for most real world. Because of the unpredictability of the real world and imperfect sensors, it is impossible to maintain a perfect world model which is the base of its success. This often leads to errors in reasoning, planning and the final execution in the control system. With a single pipeline execution, the time needed for a mobile robot to respond to a situation is equivalent to the time taken for the information to pass through sensing, world modelling, planning and acting, resulting in poor real time performance.

Advances have been made in exploiting hierarchical structures to improve reactivity by breaking up the original single execution pipeline into a number of parallel ones. The typical approach is hierarchical decomposition which may involves vertical and horizontal decomposition. The reasons for vertical system decomposition are increasing degrees of abstraction and decreasing frequency of interactions with the environment. Reasons for horizontal system decomposition are far more reactive response and simpler modelling of low level control activities.

2.3 Behaviour Approach

The behaviour approach was first introduced by Brooks in the subsumption architecture[Brooks86] which was subsequently updated in [Brooks89]. A modified version was developed by his student Connell[Connell89].

Brook's subsumption architecture is more a design methodology than an architecture. Rather specifying a set of components and the interfaces between them, subsumption specifies a set of guidelines to be used for developing control mechanisms. To quote Mataric, "Rather than a recipe for programming robots, [the subsumption architecture] is a set of philosophical concepts about robot... design."[Mataric90].

Brook's formulation of the architecture is based on the idea of decomposing the problem of robot control by task rather than by function. Most robot control architectures are composed of functional modules which perform such processes as sensor interpretation, planning, execution monitoring, etc. Brooks argues that such a design is inherently inefficient because it requires that each functional module be powerful enough to support any task the robot may perform.

Rather than developing general functional modules, the subsumption architecture advocates the development of more specifically focused mechanism called behaviours. Each behaviour is designed to control only a single task, allowing the computation within the behaviour to be optimised for that task. Each behaviour is coupled directly to the robot's sensors and actuators. Behaviours are organised hierarchically into layers, with the lowest level behaviours responsible for maintaining the viability of the robot, and the higher levels pursuing more purposeful goals. The idea is that if the higher levels cannot provide guidance, lower levels still cause the robot to react reasonably, e.g., not bump into obstacles. When a higher level is active, it can suppress more primitive behaviours below it. Conflicts among behaviours are resolved by an arbitration mechanism.

There are some other general guidelines advocated by the architecture. Each individual computational module should be fairly simple. Implementations of the architecture have been based on simple finite-state machines. The architecture strongly opposes the use of centralised data structures which can be accessed across behaviours. Each behaviour is responsible for maintaining whatever data structures it needs. There is no centralised control as in traditional architecture.

The subsumption architecture has been highly influential in mobile robotics community, and its ideas have appeared in most of the proposed architectures to some extent. To date, the subsumption architecture has been used to build dozens of autonomous mobile robots[Thau97]. However, because of their minimal prior knowledge about the environment and lack of spatial reasoning capability, they are mostly limited to low level or very specific behaviours. Connell's can-retrieving robot can navigate in an unknown environment, locate a soda can by means of active vision system, collect the soda can in a gripper, and return to its starting position. This is an advanced behaviour. However, to accomplish the task, hardware and software are tailored specifically to recognise only soda cans, the robot cannot be readily reprogrammed to "look" for other objects. The robot sometimes took very circuitous routes and there were some places its navigation scheme couldn't reach at all[Connell89]. Mataric has demonstrated a robot that was able to construct a map of its environment and plan paths using that map[Mataric90]. However, her robot operated in a fairly simple domain(essentially an one-dimensional world) and it is not clear her method can extend to more complex tasks. Because of the lack of a world model which can be shared by all of behaviours, it is difficult to co-ordinate reactive behaviours and more purposeful behaviours. The ability of the architecture to solve more complex navigation problem is limited.

2.4 RAPs

Firby's Reactive Action Package(RAP) control system[Firby89] belongs to reactive planning approaches which typically employ task-nets to carry out a plan execution. A RAP task is an autonomous process pursuing its goal until it is achieved or all methods tried to achieve fail. RAPs can hierarchically refer to other RAPs within the associated task-nets and therefore creates a task tree during execution. The RAP system consists of four major parts, the RAP memory, the RAP library, the RAP task agenda and the RAP interpreter. The RAP memory contains the best estimates of the current world state. The RAP library is simply a collection of RAPs which are themselves collections of methods for accomplishing something. Each method is annotated with information that describes under what circumstance it is applicable. The selection of a method is followed by the execution of the task net within the method. The RAP interpreter is a program which executes a RAP program. The interpreter runs in cycles. The system starts with a set of tasks stored in a data structure called a task agenda. At the beginning of each cycle the interpreter chooses a task from the task agenda based on a set of heuristics. It then finds a RAP in the RAP library for performing that task, and chooses one of the RAP's methods based on the method annotation and the RAP memory. A method is either a primitive(discrete) action, in which case it is executed, or it is a set of tasks connected by task net, in which case these are placed on the task queue. The cycle then starts again. Each RAP keeps track of whether or not it has succeeded in accomplishing its goal, and keeps trying methods until either it succeeds, or all methods have been tried. Because a task has the execution control for only one cycle each time it is invoked, the system can respond quickly to unexpected events in the world.

The RAP system was designed to be the middle sequencing layer of a so called three layer architecture[Hasemann95]. In the original RAP architecture, the bottom layer controlled discrete actions, and the top layer was a planner which generated RAPs. One important feature of RAP systems is the need of a sensor memory which is the sole base of information about the world. The sorts of tasks which the RAP system dealt with involved many object recognition and manipulations.

RAPs provide a powerful approach to decompose task and organise the sequences of task execution, monitor and intervene the progress of a task. It also provides the efficient mechanism for error detecting and recovery. The main disadvantage of the RAP is its inability to carry out parallel tasks[Firby95].

RAP architecture has been very influential in MARCO architecture proposed by this thesis. Actually, MARCO sequencing layer is a modified RAP system, adopted to control fuzzy behaviour-based low level control layer instead of discrete actions and also allow the concurrent execution of parallel tasks.

2.5 ATLANTIS

ATLANTIS is a three layer architecture developed for mobile robot indoor and outdoor navigation[Gat92][Gat93]. ATLANTIS consists of three main components: a control layer which is responsible for low level control and decomposed horizontally into "circuit" behaviours, a sequencing layer to co-ordinate robot activities and a deliberative layer which performs time-consuming deliberative computations. Low Level control layer is a subsumption architecture with extended circuit semantics called ALFA. ALFA is a robot control programming language used to specify and compile "circuit" behaviours. The main differences from pure subsumption [Brooks89] are the replacement of layer interfaces by "circuit" channels between behaviours and the use of arbitrary message formats between the state machines and the possibility to use internal states(though not encouraged).

ATLANTIS middle sequencing layer is RAP interpreter, extended by resource handling and mutual exclusion of certain behaviours using semaphore. The sequencing layer allows the concurrent primitive controlling activities, instead of discrete primitive actions by the RAP interpreter. Another feature of ATLANTIS sequencer is the ability to dynamically change parameters to control low level control modules. The operation of deliberative layer is also controlled by the sequencing layer.

ATLANTIS is a heterogeneous robot control architecture based on the claim that a successful architecture for controlling autonomous mobile robot should be heterogeneous and asynchronous, that is, it should have components which are structured differently from one another and which operate in parallel at different levels of abstraction[Gat92]. This idea has also been adapted in this thesis, though control structures involved are different.

2.6 TCA

Simmon's Task Control Architecture(TCA)[Simmons90] is a control architecture which also supports task decomposition and concurrence. The architecture consists of a set of task-specific computational processes called modules which communicate with each other by passing messages through a central control module. The central control module routes messages dynamically among the task modules. Tasks in TCA are structured as hierarchical task trees which have parent-child relationship among messages. A task tree is similar to an expanded RAP. TCA allows concurrent executions of steps in the task tree, which can include ordinary computation as well as physical tasks. TCA includes mechanisms for enforcing temporal constraints among various steps in the task tree. TCA is actually a distributed architecture with the support of central message control. Robot control processes such as perception, planning, and execution can be localised and synchronised with message passing.

The methodology used in TCA is first to develop systems with traditional sense-model-plan-act cycles, then use the TCA facilities to add concurrence. Monitoring and error handling are also added after the code for handling normal situations is in place [Simmons90]. TCA is more traditional than other architectures in that it uses pre-written, carefully engineered tasks net with explicit control information. TCA does not specify the structure of low level control mechanism at all. In term of three layer architecture, TCA belongs to sequencing layer.

2.7 AuRA

Arkin's AuRA(Autonomous Robot Architecture)[Arkin90] is motivated by biological evidence and a potential field approach. AuRA's fundamental building block is a motor schema, adopted from Arbib's notion of motor schema[Arbib85]. "Potential fields" is first introduced by Khatib[Khatib86] and now extensively used in the robotic domain[Latombe91]. In a potential field approach, a goal is represented by a potential pseudo-force from that goal's viewpoint. For example, the goal of avoiding obstacles is represented by a potential field having maximum value around the obstacles; and the goal of reaching a given location is represented by a field having minimum value at that location. At the each point, the robot responds to a pseudo-force proportional to the vector gradient of the field. Arkin's motor schema is implemented by "potential fields", which is associated either with a goal or with an obstacle. Motor schemas are combined by vector summation, resulting in an overall potential field which controls the robot's motion. A planner modulates the motor schemas to keep the robot out of local minima which are often produced when combining such potential fields[Slack90].

AuRA is comprised of five subsystems: perception, cartographer, planner, motor control and homeostatic control. AuRA's planner is basically a path planner that generates a piece-wise linear path to the goal. This plan is then passed to the execution layer, where motor schemas to follow this path piece by piece are dynamically chosen and instantiated at execution time. AuRA is not a typical three layer architecture. Accurate world modelling is a key part of the architecture.

2.8 Situated Automata

Rosenschein and Kaelbling's situation automata theory[Kaelbling90] is a formal methodology for the design of a robot control architecture. The basic architecture consists of two components, a perception component and an action component. The perception component consists of a network of combinatorial logic gates connected to the robot's sensors as well as to its own outputs through a time-delayed feedback loop. The action component is simply a combinatorial logic array. The idea is that the perception component keeps track of the current state of the world while the action component maps that perception onto an appropriate action for achieving the robot's goals in that situation. The feedback loop in the perception component allows the robot to remember the past, and thus allow past data to be incorporated into current decisions.

Situation automata theory does not commit to analogical representation of the world. The key point of the theory is that the robot should contain just enough information to accomplish tasks in its environment and this information need not be in an analogical form. The theory has many things in common with subsumption architecture, but with the formal emphasis.

The intuition of the theory is that the situation automata approach will allow high level descriptions of environments and tasks to be compiled automatically into a reactive control mechanism with a formal basis. The current main practical success have been a suite of development tools: GAPPS, RULER, REX [Kaelbling88] [Kaelbling90]. Situated automata can be seen as a reactive control layer in three layer architecture.

2.9 Blended Behaviour Approach

Blended Behaviours[Saffiotti93] approach constitutes a new direction and are based on fuzzy rule sets and fuzzy logic composition rules in order to blend simple behaviours to form complex ones. Context-dependent blending of behaviours is accomplished using desirability functions and context rules. In this approach, low level behaviours are implemented using fuzzy logic controller. These behaviours run concurrently and their weights are reassigned according to their activation level and their fixed importance orders, i.e. priorities, in response to the state of the world and goals. A synthesised control output is produced by weighted summation of all outputs from active behaviours. Blended behaviour is similar to "potential field" approach in that the behaviours are composed through weighted summation, but with fuzzy desirability instead of potential force.

The biggest problem of the approach is the weakness in coping with local minima and error recovery similarly encountered in the "potential field" method[Slack90]. The composition nature of the robot control makes it inefficient to escape local minima. When the robot control fails, direct control strategies need to be employed to recover the robot, which will result in an inconsistent low level control structure. However, the fuzzy behaviour-based approach provides the robot with robust control abilities in facing noise sensor, uncertainty and imprecision in the real world[Saffiotti et al 93a]. Blended behaviour architecture is a low level control layer.

2.10 Hybrid Approaches

There are many other robot control architectures in the literature. Most of them are variants on the traditional sense-model-plan-act architecture where a planner constructs a plan from a world model to be executed by an execution system. The most common is some type of hierarchical exploitation on basic approach, where one planner generates a plan at a high level of abstraction which is fed to another planner to fill in details, such as NASREM[Smith89].

Another form of variation is to allow some of the processes to be parallel. The CODGER architecture[Shafer89] is a recent example of this approach. It attempts to overcome the inherent slowness of the sense-model-plan-act model by constructing incremental plans and pipelining the process. The result has been quite successful. However, like all purely traditional architectures, CODGER is still limited by the speed of the pipeline. CODGER is not a real-time system. Due to the relatively long latencies in message passing and UNIX time-sharing execution patterns, data transfer cannot be guaranteed within given time bounds. CODGER is an inappropriate architecture for systems requiring real time processing[Shafer89].

Attempts have also been made to provide the traditional approach with an execution monitoring system which monitors the execution of the plan and takes corrective action when things go wrong(e.g. [Broverman87]). The execution monitoring system simply checks the values of the robot's sensors to make sure that they fall within expected bounds. When they do not, the execution monitoring system diagnoses the problem and takes corrective action. The problem is that diagnosing the execution problem and taking the corrective action is also a very hard thing to do, involving the entire problem of deciding what to diagnose and where to start. Instead of diagnosing, some systems try to anticipate possible failures and provide required responses [Miller89] [Gat90].

Departing farther from the traditional approach than the above variations is the anticipation of possible situations and actions upon which plans, called situation-action plans, can be organised. These compiled plan approaches consider all possible situations(or at least a large subset of them) and map appropriate actions to them instead of using run time planning. Examples of the approaches are teleo-reactive trees[Nilsson94] and universal plans[Schoppers87]. Teleo-reactive trees are sets of condition $\rightarrow$ action rules which are continuously evaluated. The first action for which the energising evaluates to true is executed and continue as long as the energising condition is true. Teleo-reactive trees can be recursive and organised in hierarchies. Although the character of the teleo-reactive approach is more like that of a sequencing machine(like RAP), the continuous nature of actions and condition evaluation puts it to low level control approach. The major disadvantage is inefficiency since all conditions must be evaluated continuously. Universal plans push the situation action approaches to the extreme by generating plans which say what to do in every conceivable situation. They are decision trees which map the current world state into the next action to take. An universal plan is usually created off-line by a "reverse planning" procedure. Given a goal and a set of operators as inputs, the procedure chains backwards from the goal condition using the descriptions of the operators and generates the plan. There are a number of problems with this approach when applied in the real world. First, decision trees can grow exponentially as the numbers of actions and conditions increases. Second, universal planners require that all possible states of world be counted at plan-time. In the case of path planning, it means that a global map must be known at plan time. If a new block or a new obstacle is encountered, the system must replan from scratch. Universal plan approach belongs to single reactive control layer approach.

There are a number of other notable architectures in the literature. The architecture presented in [Noreils90] addresses the problem of recovering from task-level failures. Robo-Soar[Laird91] adapts the SOAR production system architecture to robotic applications. Soldo argues that reactive and preplanned control can be usefully combined, and presented an architecture based on a control structure called a behaviour expert which was used to control a real robot in an indoor environment [Soldo90]. Durfee presents a similar argument and a system for controlling a robot implemented as a blackboard[Durfee90]. Anderson and Donath present an architecture based on observations of animal behaviour[Anderson90]. Hammond presents an architecture which is designed to be able to take advantage of unexpected opportunities[Hammond90]. Most of these systems are homogeneous architecture. DRTA(Distributed Real Time Architecture)[Hu94] uses a transputer architecture to distribute sensing, world modeling, path planning, controlling, etc and coordinate these processes through message passing which is similar to TCA. It also bears some resemblance to AuRA in that it uses the similar structures to organise perceptual systems and global path planning. Neves and Gray[Tilford97] presented general principles for an unified control approach, including information generation, action generation and reconfiguration and implemented the approach in simulation.

Three layer architecture seem to be the current state of evolution. Three layer architectures usually employ three levels of abstraction(occassionally a fourth layer is added for servo control [Payton90]). These three layers are a deliberative layer, a sequencing layer and a reactive control layer. The deliberative layer uses classical AI representation and reasoning techniques such as temporal planning, scheduling, and resource handling. Activities on this layer correspond to long term planning. This level relies on very abstracted knowledge, highly sophisticated reasoning techniques, and extensive application domain knowledge. Two planners commonly used in this layer are IxTeT[Ghallab94] and SIPE, SIPE2[Wilkins94][Wilkins95]. The sequencing layer involves the selection of appropriate task nets and organises the correct sequences of controlling activities. A task net is a pre-written ordered set of actions, such as behaviours or operators. Task nets represent execution procedures and can have hierarchical structures. The sequencing layer selects appropriate task nets and executes them following the ordered steps within the task nets. Execution of task nets involves activation, monitoring and termination of reactive layer behaviours. The reactive layer usually consists of behaviours, performs the transition of task goals from higher level to numerical control and combines the separate behaviours to produce control output.

Several typical three layer architectures are 3T[Bonasso94, 95], ATLANTIS[Gat92], GLAIR[Hexmoor93, 95], LAAS[Chatila92][Ingrand95] and Payton architecture(4 layers, the bottom 2 roughly correspond to the reactive layer)[Payton86,90]. Another example of a heterogeneous three layer system is SSS, presented in [Connell90,92] where a human is used as a high level controller for a low level system based on the subsumption architecture.

2.11 Summary

During the last decade decisive progress on the robot control architecture has been made though major problems still need to be addressed, namely noise sensor, unpredictability, imprecision and approximity. The introduction of the reactivity approaches, with its most extreme implementation being the subsumption architecture, is probably the most important development in robot control architecture during the last 20 years.

Traditional sense-model-plan-act approach was supplemented by reactive components and seems to have lost importance because of the following reasons: (a) single pipeline execution is time consuming and cannot produce fast response in real world environment; (b) different computation mechanisms can be provided to support different processing involved; (c) most difficult problems lie in the interactions with the world and the need for fast response at lower levels of representations. However, it still is an essential part, in one form or another, in all mainstream architectures, though the difference in realisation can be huge.

Advances have been made in exploiting hierarchical structures to split the robot control system vertically into levels of hierarchies as well as the introduction of control entities, such as behaviours, reactive action package, or teleo-reactive programs, to split up the low level control horizontally into concurrent operations. The reasons for vertical decomposition are that increasing degrees of abstraction results in decreasing frequency of interactions with the environment, allowing different structures to be employed for different levels of operations. Reasons for horizontal decomposition are far more reactive response and simpler modelling of low level control activities. The current state of the robot control architecture seems to be three layer architectures which usually employs three levels of abstraction. These three layers are the deliberative layer, a sequencing layer and a reactive control layer.

This thesis concentrates on the two lower levels, sequencing layer and reactive control layer. Particularly, the research focuses on the combination of RAP-like sequencing technology with fuzzy behaviour-based low level control layer to solve the problems raised in Chapter 1. Combining a RAP-like sequencing layer to a fuzzy behaviour-based control layer is a topic which still remains unexplored. Fuzzy logic control has already been used in mobile robot navigation , especially in behaviour-based approach. In this thesis, a triangle form of fuzzy membership function and a singleton representation of fuzzy output are chosen to support a simple design of behaviours and also a fast control inference process. The review of fuzzy logic control is given in Appendix A.

Chapter 3 A Two Layer Control Architecture

This Chapter develops the computational structure required to implement a control mechanism for an autonomous mobile robot based on the concept of fuzzy logic control and a modified RAP. The control architecture consists of two parts, a low level control mechanism and a higher level sequencing mechanism. This Chapter approaches the problem from the bottom-up by considering first how to organise the low level control. The low level control means controlling primitive activities which contains no decision-making computations. These include the robot's direct interactions with the environment and the purposeful control actions in order to accomplish an predetermined goal. This Chapter describes a low level control mechanism based on fuzzy set theory. The notion of the behaviours have been adopted from Brooks's subsumption architecture[Brooks89], but behaviours are implemented using fuzzy logic control, instead of "circuit" and "potential field" methods[Gat91a][Arkin90]. The layer structure is replaced by a behaviour link network to perform co-ordination of the behaviour's activities for the robot control.

Having developed the structure for controlling primitive activities, the Chapter will go on to examine how to control higher level activities. The computational structures needed to control the higher level activities turn out to be very different from those needed to control the low level activities. An existing technology, RAP can be employed for this purpose with some modifications.

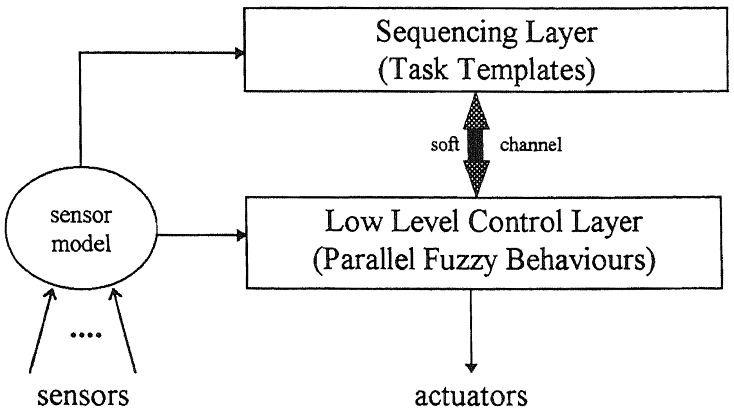


Fig. 3-1 MARCO Block Digram

Fig. 3-1 is a simple block diagram of MARCO. The sequencing layer is based on a RAP-like control structure called a task template, carrying out task sequencing and management activities. The low level control layer consists of a collection of fuzzy behaviours, running in parallel and producing a single set of outputs for controlling mobile robot actuators. An independent perceptual subsystem is responsible for world modelling and maintaining a sensor model which is accessed by both the layers. This Chapter will concentrate on the construction of the two layers. The sensor model and the perceptual subsystem will be discussed in Chapter 4.

3.1 Low Level Control Layer

This section describes the first part of the MARCO architecture, the one that controls low level robot activities. As argued in Chapters 1 and 2, a low level control mechanism should have the following characteristics:

. fast response time;

. able to cope with sensor noise, imprecise information and uncertainty;

. easy to use heuristic control knowledge.

To impose these characteristics onto low level control behaviours, fuzzy logic control provides an efficient method. To review briefly, fuzzy control is a method of controlling a system that is similar to classical process control, but differs in that it substitutes imprecise, heuristic notions for the precise numeric measures of a control model. In order to organise behaviours using fuzzy logic, the structure of a behaviour needs to be examined first.

3.1.1 Behaviour Structure

Definition 1: Behaviour

In Summers's contemporary English dictionary[Summers95], a behaviour is defined as a way of acting or reacting in a specified way. The definition is extended in this thesis for the robot application. A behaviour is defined as a way of acting or reacting in order to accomplish a single goal. To be less abstractive, it can be further stated that a robot control behaviour is a set of control strategies to formulate robot movement actions in order to accomplish a single goal. Under such definitions, some examples of robot behaviours can be Avoid Obstacle, Keep Moving, Follow Wall, Follow Corridor, Go To Position, etc. Avoid Obstacle is a single goal for the robot to achieve when there is an obstacle ahead of it. Similarly, Go To Position is a single goal that is assigned to the robot to complete. The differences between the above two behaviours are that avoiding obstacle is a direct action in facing environment contingencies, while going to position is a purposeful behaviour. The former is called a reactive behaviour and the latter a task-oriented behaviour. Whatever their functions, these behaviours are similar in several aspects. They all have input and output components. Input components can be direct sensor data or interpreted environment information, such as range and encoder data or environment features such as a door or a piece of wall. Output components constitute the output control parameters, which mainly are the speed and heading information of the robot. Each behaviour creates the output control parameters to satisfy its purpose or function. For example, a Go To Position behaviour creates a direction and speed which allows the robot to move straight towards the specified position. At the same time, Avoid Obstacle behaviour produces the speed and heading which slows down and steers the robot away from an obstacle right ahead. The above two outputs are in conflict for actually controlling the robot movements, but they are the result of the two individual behaviours for their own purposes. Later, how to resolve the contradictions will be discussed. Now, let us check the behaviour structure further. In order to create the output control parameters from input sensor or environment information, we need a control method, or a control model to connect these two parts. Fuzzy logic control is a very efficient way to perform such connection roles. In a fuzzy logic controller, a collection of fuzzy control rules is used to form the control formula. Applied to a robot behaviour, this set of fuzzy rules constitute the control strategy for a behaviour to fulfil its purpose.

3.1.2 Fuzzy behaviour Components

3.1.2.1 Fuzzy Input Variables and Output Variables

From the above discussion, we know that a robot fuzzy behaviour includes at least three parts: input, output and a fuzzy control model. Its control model has a set of fuzzy control rules. These control rules are fed with fuzzy input variables and produces a set of fuzzy output variables. The fuzzy input variables actually summarise the state of the robot for a behaviour. Each fuzzy behaviour is provided with its own set of fuzzy input variables. The values of these input variables are computed through fuzzification from extracted specific sensor information necessary for the behaviour to operate. For example, we have a simple Avoid Obstacle behaviour with three rules:

if obs_right is CLOSE then left_heading;
if obs_left is CLOSE then right_heading;
if obs_front is CLOSE then decrease_speed;

This simple behaviour relies on three fuzzy input variables, obs_right, obs_left and obs_front. These variables indicate the degree of CLOSENESS, representing the robot's state in relation to the obstacles in the environment. Suppose that the robot has three sensors in the right, left and front directions which sense the nearest obstacles in these directions. The range values to detected obstacles are fed to the fuzzy behaviour and the values of its fuzzy input variables are calculated to indicate how close these obstacles are to the robot. Depending on the fuzzy input variables, Avoid Obstacle behaviour can produce the output control values accordingly in order to move around the environment safely. Because this Avoid Obstacle behaviour is only responsible for staying away from obstacles, the fuzzy input variables indicating closeness to the robot's surrounding are sufficient for the behaviour to operate. For other types of fuzzy behaviours, their fuzzy inputs variables have to reflect the state of the robot in relation to the task that behaviour is supposed to perform. For a Go To Position behaviour, it does not need the indications of the robot's closeness to obstacles but requires the measurement of the CLOSENESS of the robot to the specified position and the deviation of the heading of the robot from that position. Similarly, different fuzzy input variables indicating relevant state information of the robot with respect to other behaviours are required for those behaviours to operate on.

In most fuzzy behaviour-based approaches[Goodridge94][Reignier94] [Garcia-Alegre93] [Sugeno89][Konolige92], the method of derivation of fuzzy input variables are similar, but the representation of fuzzy output variables can be different. Standard forms of fuzzy output variables are used in [Reignier94][Martinez93] [Skubic94]. Goodridge [Goodridge94], Garcia-Alegre[Garcia-Alegre93] and Vandorpe [Vandorpe94] used crisp values which are similar to fuzzy singleton representation of output. Konolige[Konolige92] used simplified control set to represent fuzzy output. Sugeno[Sugeno85] used another type of fuzzy control rule which has the form

if X is A_i and Y is B_i .... then

$Z = a0_i + a1_i x + a2_i y + \ldots\ldots$

The fuzzy output variable is a linear function of fuzzy input variables. As argued in Section 4, Appendix A, a singleton representation of fuzzy output has the advantages of fast computation and simplicity. In constructing MARCO's fuzzy behaviours, fuzzy output variables are represented as singleton fuzzy values to improve the real time response of low level control behaviours. At MARCO's low level control layer, only two types of output control parameters are required for the robot's control, forward/back speed and heading. These two types of fuzzy singleton values represent the differentials between current speed and angle to the desired values. For example, two typical fuzzy control values would be

if obs_front is CLOSE then slow_down(to speed 50)

and

if obs_right is CLOSE then left_heading(8 degree);

The fuzzy singleton value slow_down is the differential value between current speed and 50 and left_heading is 8 degree from current heading. The heading and speed output variables can be used in different behaviours. Some fuzzy behaviours may only need one output while others may need both of them. For example, a speed control behaviour only needs speed as an output variable, while an obstacle avoidance behaviour needs to control both the speed and the heading.

3.1.2.2 Membership Functions

In the above section, fuzzy input variables are described as the indications of the robot's state in relation to its environment from a behaviour's point of view. These state values are actually transformed from the robot's sensor information by fuzzy membership functions. Again, take Avoid Obstacle as an example. In the rule

if obs_right is CLOSE then left_heading,

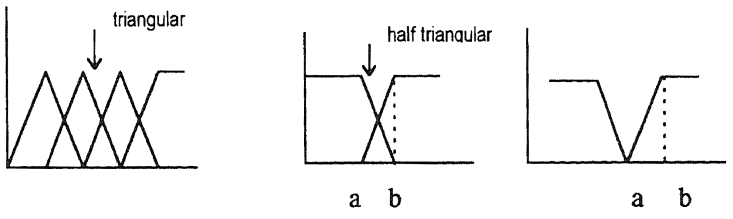


Fig. 3-2 Membership function structures used in MARCO

when fuzzy input variable obs_right, has a value 1.0, it indicates that an obstacle is very close to the robot. This results in the full escaping heading, -8 degree, of the rule output. The problem here is how to decide appropriate criteria. We can say that 0.5m is the minimum distance which indicates a full degree of CLOSENESS or 1.0m is the criteria. We can also say that 5.0m is the distance from where the degree of CLOSENESS is 0.0 or 3.0m can be the case. Even the criteria for zero and full degree is the same, a fuzzy input variable can have different values using different types of membership function. In most fuzzy behaviour implementations, a triangle form of fuzzy membership functions is used because of its easy computation and proved effectiveness[Cox94]. In MARCO, triangular and half-triangular forms of membership functions are selected as shown in Fig. 3-2.

Although, fuzzy input variables are behaviour-dependent, their membership functions have similar structures. In MARCO, the following linguistic terms listed in Table 3-1 are used to represent different fuzzy sets for different fuzzy input variables:

Table 3-1 Liguistic Terms Used in Fuzzy Rules in MARCO

speed	angle	distance	time	step
FAST	BIG	SMALL	LONG	SMALL
SLOW	POSITIVE MEDIUM	NEAR		
	NEGATIVE MEDIUM	CLOSE		
	POSITIVE BIG	POSITIVE BIG		
	NEGATIVE BIG	NEGATIVE BIG		

For the fuzzy variables *time* and *step*, there is only one linguistic term involved respectively. The terms **LONG** and **SMALL** are used in Recover Stall behaviour which only becomes active when a robot is stalled for a certain period of time or the robot moves in a very slow pace. The membership functions of the linguistic terms in Table 3-1 may look like the function graph in Fig. 3-2. In implementation, these functions take floating-point number arguments and produce a floating-point fuzzy value between 0.0 and 1.0. As mentioned earlier, the difficulties in deciding membership functions are in the choice of criteria, i.e., the a, b values as indicated in Fig. 3-2. Every fuzzy behaviour has several fuzzy control rules, each of which may include several fuzzy input variables. Because the choice of membership functions for one fuzzy variable may affect the others, to select a set of suitable a, b values manually may need much effort and a time-consuming trial and error process. Chapter 5 will be devoted to deal with the problem through an automatic learning approach.

3.1.2.3 Fuzzy Control Rules

The central component of a fuzzy behaviour is its fuzzy rules. This is where the action is: rules define how fuzzy input variables are transformed into fuzzy control values, which ultimately are combined to create a set of control parameters for the robot. The most important task of designing a fuzzy behaviour is to derive a set of effective control rules. There are many ways described in the literature[Lee90] [Sugeno85]

[Kruse94] [Cox94] on how to derive fuzzy control rules. In fuzzy logic controller design, four commonly used approaches are based on:

1. operator's experience;
2. control engineer's knowledge;
3. fuzzy modelling of the operator's control actions;
4. fuzzy modelling of the process.

Most fuzzy logic controllers are based on the knowledge and experience which are expressed in fuzzy if-then rules[Sugeno85]. Sugeno designed a fuzzy controller to control a model car and park a car into garage[Sugeno85][Sugeno89]. The fuzzy control rules are derived by modelling a driver's control actions. In driving, a driver employs subconsciously, a set of fuzzy if-then rules to control his car. We can derive such implicit control knowledge to control a mobile robot. In MARCO, fuzzy control rules for fuzzy behaviours are designed using the combination of Sugeno's method and heuristic control knowledge. However, the structure of a fuzzy rule is different because with MARCO, a fuzzy singleton value is used, instead of a linear function. In the design of a fuzzy behaviour, first a driver's control behaviour is examined. For example, to derive fuzzy control rules for an avoid obstacle behaviour, we first examine our driving actions to avoid collision with other cars or obstructions in a road. We probably will outline our driving actions like this:

> if there is obstruction at the right, turn the wheel left; if there is obstruction at the left, turn the wheel right; if there is obstruction ahead, slow down and turn left or turn right.

For a mobile robot, although the obstacles can be very different from the obstructions found in the road, their effects are similar. Therefore, the above heuristic control knowledge can be modelled into fuzzy control rules as follows:

if obs_right is CLOSE then heading_left;
if obs_left is CLOSE then heading_right;
if obs_front is CLOSE then slow_down;
if obs_front is CLOSE then heading_left or heading_right.

The term *CLOSE* is a fuzzy set which represents the closeness of a robot to obstacles from a designer's point of view. From such translation, one can see how easily fuzzy control rules can interpret heuristic control knowledge, though the above description of driving actions and the fuzzy control rules need to be refined further to design a competent avoid obstacle behaviour.

3.1.3 Fuzzy Behaviour Structure

The above sections have described the fuzzy behaviour components and their possible design. A fuzzy behaviour should have an input part, a fuzzy control model, i.e. a fuzzy ruleset, and an output part. These three components constitutes a complete fuzzy logic controller. However, a fuzzy behaviour is a complete special purpose computational module for the low level robot control, which needs other mechanisms to be able to fully engaged in the control process. In MARCO, fuzzy behaviours at the low level can be mediated by the higher sequencing layer. This means a fuzzy behaviour needs a mechanism to connect with the higher layer through which the higher level of control can monitor and influence the fuzzy behaviour. In the ATLANTIS architecture, such connections are realised through channels which are implemented as a "circuit". In a RAP, there is no such means because a RAP can directly control discrete actions. In MARCO, a soft two-way channel is designed for the higher layer to change the control parameters and also monitor the state information of fuzzy behaviours, such as maximum speed, running state, activational level and their achievement. Apart from the higher level monitoring and intervention, fuzzy behaviours also compete and co-operate with each other for the control of the robot. Therefore, a competition and co-ordination mechanism is also required in the low level control layer. For this purpose, a structure, called a behaviour link is designed for a fuzzy behaviour. This behaviour link will be described in details in the following chapter. For now, the structure of a fuzzy behaviour in MARCO architecture can be presented. A fuzzy behaviour comprises a complete fuzzy logic controller, a soft two-way channel to connect to a higher layer and a behaviour link to connect with other behaviours. The complete structure of a fuzzy behaviour is depicted in Fig. 3-3.

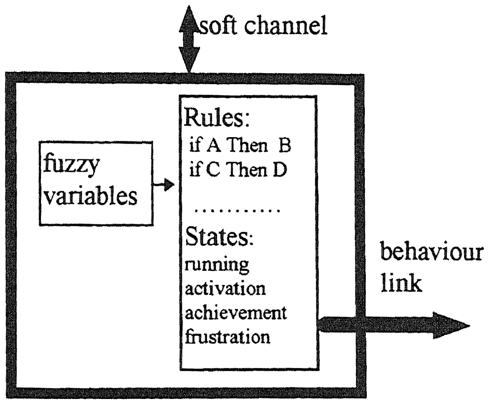


Fig. 3-3 Structure of a Fuzzy Behaviour in MARCO

3.1.4 Fuzzy Behaviour Processing Algorithm

The computational process of a fuzzy behaviour in MARCO consists of four parts. First, necessary sensor information is extracted by perceptual subsystem for the behaviour. The processing in this part varies for different behaviours. Some behaviours need little computation, while others may need longer processing. Second, fuzzy logic control process is carried out. This process includes four steps:

1. fuzzification of input sensor data to obtain the values of fuzzy input variables;
2. calculations of the weights of antecedent parts of fuzzy rules;
3. calculations of fuzzy singleton values of fuzzy control rules;
4. defuzzification.

The expense of the computation in this part also depends on individual behaviours. Simple behaviour usually takes very little computation. Complicated behaviours may take a longer time. However, the design of membership functions and the fuzzy reasoning method guarantee fast computation time. Third, behaviour state information is calculated. This process will virtually take no time because most of state values will be available already from the previous calculations. Finally, the activation levels of fuzzy behaviours are recalculated through the energy redistribution by the behaviour link network. This process will be discussed later. Fig. 3-4 presents the computational algorithm in a fuzzy behaviour.

```
get_sensor_input();
fuzzy_control_process():
  . fuzzification;
  . antecedent weight calculation;
  . output singleton calculation;
  . defuzzification.
set_behaviour_states();
redistribute_activation_energy().
```

Fig. 3-4 Processing Algorithm of a fuzzy behaviour.

3.1.5 Fuzzy Behaviour Link

The basic building block of the MARCO's low level control layer is a fuzzy behaviour. Individual fuzzy behaviours perform independent processing for their own functions. However, their control outputs have to be combined to produce one set of control output. In the above sections, a data structure built in a fuzzy behaviour, a behaviour link, has been outlined for this purpose. Detailed description of the method for combining fuzzy behaviours will be presented in Chapter 4. Here, the combination mechanism is briefly introduced.

Fuzzy behaviours in MARCO's low level control layer are combined through a behaviour selection network based on behaviour links. A behaviour link represents the relationship of connected behaviours, namely promotion and inhibition, as shown in Fig.3-5. A behaviour selection network is composed of all links between behaviours.

The selection network functions as an energy redistributor, continuously changes the energy flow among the behaviours. The process results in the energy being accumulated in the most suitable behaviour for the control of a robot, with respect to the current state of environment and motivations of the control system. The behaviour is then be selected and the control output is produced by composing the behaviour's activation energy and the accumulated one. Unlike hard wired suppress channel mechanism [Brooks89][Gat91a], a behaviour link provides a more flexible way to both

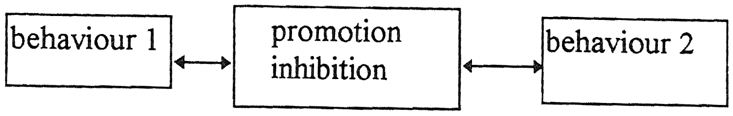


Fig. 3-5 A Behaviour Link

combine and select behaviour with smoother transition of control. With the selection network, layered structure for organising fuzzy behaviours are not needed and reactive and task-oriented behaviours can both reside in a single low level control structure.

3.2 Higher Level Control Layer

In the above sections, the low level control layer of MARCO architecture has been described. This section introduces the higher control layer. The higher control layer of the MARCO architecture is responsible for arranging the sequences of controlling activities which are usually carried out at different time and places under various conditions. This higher layer is called the sequencing layer[Hasemann95].

To complete a navigation task, a robot is provided with a plan which is produced either by a high level planning system or a programmer. The execution of the plan is finalised in the low level control layer. In MARCO, the low level control involves the concurrent execution of active fuzzy behaviours and dynamic selection of behaviours to control the robot. If no plan is provided, the low level control consists of only the survival controlling activities which are realised through reactive behaviours. When a plan is fed into the robot control system, the low level control activities comprise of reactive behaviours as well as task-oriented behaviours. However, the low level control alone is inadequate to complete a complex navigation task. It does not have abilities to schedule the right sequences of tasks given in a plan and reorganise the current execution when it fails.

Suppose that the robot is given a navigation task indicated in Fig. 3-6. The navigation plan is as follows:

1. follow corridor A;
2. enter door B;
3. go to position C.

The preferable execution procedures would be like this:

1. move along and find corridor A, then follow it;
2. find door B, when close to door B, enter it;
3. go to position C.

It should be no problem for the low level control layer to execute these tasks when they are submitted individually. The problem occurs when the robot follows corridor A and moves close to door B, where it registers the door and starts to enter it. If there is no intervention, two active

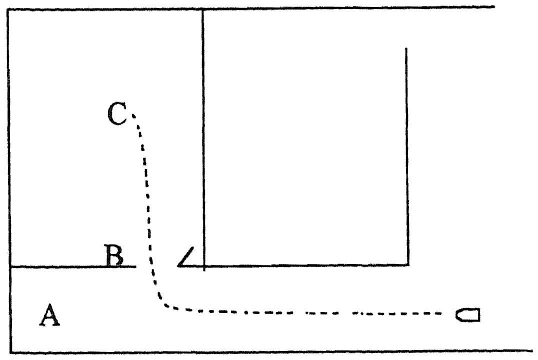


Fig. 3-6 An Example Navigation Task

purposeful controlling activities exist at the same time. One is "following corridor A" because the robot is still in the corridor and the other is "entering door B". The first activities have forward moving control output while the second try to turn the robot to the right. These two control actions are contradictory and the result may be that the robot oscillates endlessly, stops or crash into wall. To prevent such conflicting controlling behaviours, an intervention or mediation mechanism is needed to organise the correct order of controlling activities. For the above example, "following corridor A" activities need to be stopped before "entering door B" starts. Such an intervention mechanism should also be able to monitor the controlling activities and take appropriate actions when things go wrong. For instance, the robot may be unable to go through the door due to unforeseen reasons. The intervention mechanism should then initiate retrying activities or start other failure recovery process, either readjust control parameters, or activate other controlling activities, or even abandon the task step in the worst case. Like other low level control structures [Brooks89] [Mataric90] [Gat91a][Bonasso91][Konolige92], MARCO's low level control layer is not capable of taking care of such mediation jobs. The computational structure for such a mechanism turns out to be very different from the low level control. In three layer architectures [Bonasso94,95] [Hexmoor93,95] [Chatila92] [Ingrand95][Payton90][Gat92], this computational structure is the middle sequencing layer. To organise the sequencing layer for MARCO's higher level control, an existing method, RAP is exploited. The control method of the sequencing layer is heavily borrowed from the RAP. In fact, MARCO's sequencing layer is a modified RAP system and implemented with commonly used multiprocess operating system scheduling technology. A brief introduction to the RAP has been given in Chapter 2. For more detailed information, please refer to [Firby89]. In the following sections, the organisation of MARCO's sequencing layer is described. The differences between a pure RAP system and the sequencing layer are also discussed.

3.2.1 Interfacing to Low Level Control Layer

This section begins by examining the possible interface between the sequencing layer and the low level control layer in MARCO. A RAP system is mainly used to directly control the discrete actions of the robot movement. There is no low level control layer in a RAP system. This is because RAPs are mainly developed for the robot applications which involve many objects handling or manipulations. In MARCO, the sequencing layer is responsible for organising the sequences of the task execution which is finalised in the low level control layer. In a RAP system, control actions can be passed on directly to the hardware through a method which only consists of primitive action commands. In MARCO, no primitive actions can directly be sent to the hardware from the sequencing layer. Instead, the sequencing layer issues commands to initiate the control actions in the low level control layer. The sequencing layer intervenes with the low level control layer in three ways. First, it can initiate or terminate a behaviour in the low level control layer. Second, it can monitor and change a behaviour's states in the low level through a behaviour's soft channel. Third, it can adjust a behaviour's action by altering the behaviour's control parameters. For every task-oriented behaviour in the low level control layer, there are 3 types of routines involved in the sequencing layer: initiation, monitoring and termination.

For example, we can invoke two different task-oriented behaviours at the same time. We can also initiate or stop a task-oriented behaviour in the middle of another behaviour execution. However, two task-oriented behaviours, which will interfere with each other, must not be allowed to be active at the same time. This can be done by the sequencing layer using the proposed interface. Take the example in the beginning of Section 3.2 again. To resolve the contradiction that the robot is trying to enter door B while still following corridor A, the sequencing layer first monitors the current environment condition. When the robot moves close and its perceptual subsystem discerns door B, the sequencing layer first terminates "following corridor A" activities. It then generates the behaviour "enter door" and also starts a task which monitors the progress of the behaviour. When the robot has entered the door, the sequencing layer terminates the monitoring task and also removes the behaviour from the low level control layer. Other possible control with the interface is the adjustment of a behaviour's parameters through the soft channel. If the speed setting is too fast for "enter door" behaviour, the sequencing layer can directly access the behaviour to reset a slow speed so that "entering door B" activities can be more sustainable.

3.2.2 Task Template, Tasks and Task Queue

The next step is to develop the computational structure needed to manage the activation and deactivation of controlling activities. We need a convenient way to specify situation-driven or planned controlling procedures that will achieve the robot's goal. In particular, we need to be able to tell the robot what to do when conflicting situations rise.

As mentioned earlier, the solutions have been found using Firby's RAP with some modifications. The heart of the sequencing layer is a data structure called a task template. A task template is similar to a RAP. The term is somewhat more direct to describe the procedures of controlling activities instead of discrete actions. A task template is a collection of methods(task nets) for accomplishing something, together with annotations describing under what circumstances each method is applicable. A task is the computing process of executing a task template code. For example, a task template for entering a door might contain two methods, one for when the initial position of the robot is close to the door, and the other for when it is not. The first method might contain three steps: initiate a door-entering activities in the low level control layer, monitor that controlling activity until it completes or fails, and then finally terminate the activity. The second method would begin with another task template for corridor following and then start the same controlling activities as with method one, when the task prescribed by corridor following task template is finished. Fig. 3-7 gives the possible structure of this example task template based on a RAP syntax. A simple annotation starting with "#" is given to the first method of the task template. Please refer to [Firby89] for a complete description of the RAP structure.

```
(Define-Task Template              # start to define a task template;
  (Name (enter-room ?room))        # specify the name of a task template and its argument;
  (Succeed (state success))        # success clause of the task template;
  (Method                          # define the first method to select;
(Context (and (known ?room)                    # context to use method: room known
              (near ?room)))                   # and close to robot;
(Task-Net                                                   # define task-net for the method;
    (t0 (start-behaviour room-entering ?behaviour ) for t1) # first step of the task net, a
                                                            # created behaviour as
                                                            # precondition for step t1;
    (t1 (monitor-behaviour ?behaviour ?state)               # second step, monitor
                                                            # beh. state, achieved, as
        (achieved ?state) for t2)                           # precondition for step t2;
    (t2 (remove-behaviour ?behaviour ?state)                # remove behaviour at step
                                                            # t2; set task state to success,
        (success ?state))))                                 # the method succeeds.
(Method
 (Context (and (unknown ?room)
               (known ?current-corridor)
               (in-corridor ?room)))
 (Task-Net
     (t0 (start-task corridor-following ?current-corridor)
               (found ?room) for t1)
     (t1 (remove-task corridor-following) for t2)
     (t2 (start-behaviour room-entering ?behaviour) for t3)
     (t3 (monitor-behaviour ?behaviour ?state)
         (achieved ?state) for t4)
     (t4 (remove-behaviour ?behaviour ?state)
         (success ?state)))))
```

Fig. 3-7. A Simple Example Task Template

The context of using the second method is that the location of the room is unknown and the robot is currently in a corridor where the room resides. The robot must first find the room and then enter it. This method is realised by a task net with five steps. In the first method, the task net only involves the initiation, monitoring and termination of behaviours at the low level control layer. In the second method, the first step invokes another task template for management of corridor-following activity. After the room is found by the perceptual subsystem as the result of following the corridor, this activity is stopped by the removal of the task. The rest of the control activity is the same as the one prescribed in the first method. It should be noted that the termination of the corridor-following task involves both the higher level managing task and its initiated behaviours at the low level control layer.

Methods may do the following things in MARCO's sequencing layer. They may initiate, terminate, and monitor controlling activities at the low level control layer. They can also instantiate other task templates. Ordinary computation can also be carried out in steps within a method's task net. In this aspect, a method is similar to a TCA's task net[Simmons90] and different from pure RAP's definition. The advantages of such structure will be discussed in the next section, together with other modifications. Once a task template is instantiated, it becomes a task and is inserted into a task queue. A task queue is the same as RAP task agenda. The terminology was changed because the name, task queue is more descriptive.

3.2.3 Task Scheduler

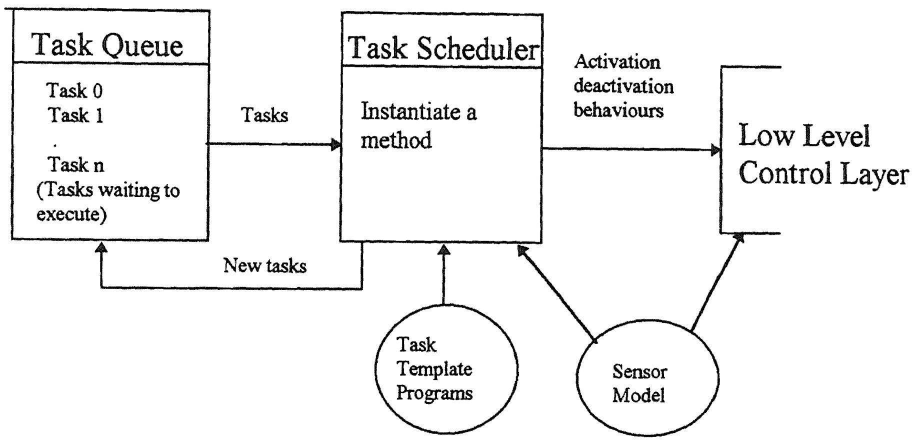


Fig. 3-8 Task Execution Diagram

MARCO's sequencing layer mainly consists of three parts: a set of task template programs, a task queue and a task scheduler. Unlike a RAP system, the sequencing layer does not exclusively possess a sensor memory but shares a sensor model maintained by a perceptual subsystem with the low level control layer. The task scheduler is a simplified RAP interpreter, which mainly involves the creation, execution and termination of tasks in the task queue. It does not interpret an instantiated task template because a task template is directly implemented as a program in MARCO. This is also the reason we use the name task scheduler. However, many issues in the RAP interpreter addressed extensively by Firby are still very well suited to the task scheduler in MARCO.

A task execution requires that a task template description and environment information contained in a sensor model be available. The task is executed by the task scheduler according to the algorithm shown in Fig.3-8 which has the following steps:

1. Choose a task to run from task queue;
2. Check the task state to see if it is finished;
3. If finished, remove the task, subtasks from the task queue and invoked behaviour from low level control layer;
   If not finished, choose a method and execute the task net step;
4. If current step contains another task template, instantiate a subtask and put it on the task queue;
   If current step contains normal computation, execute it;
   return to task queue;
5. Go to step 1 and repeat.

Deciding what task should be selected for execution and what method should be used are fairly complex problems which Firby addresses extensively. The selection criteria and heuristics developed in RAPs are mainly for the application which involves many object recognition and manipulation processes. In MARCO's sequencing layer, such complex criteria are not necessary because MARCO is mainly intended for navigation tasks which involve moving from place to place. The procedural ordering of the task execution is less complex and more straightforward. Another difference is caused by the way the world model is constructed. The construction of the RAP memory or sensor model is done by RAPs explicitly designed for world modelling. It may be the part of a RAP task execution and result in a very complex RAP. In MARCO, an independent perceptual subsystem is employed to perform world modelling which is mainly the extraction of environment features. A task template only needs to access the sensor model for the information. This again makes the task execution procedures simpler. These differences mean that the structure of a task template and its invocation process by the task scheduler can be further simplified.

In the RAP's execution, the RAP interpreter chooses the task from task agenda because many tasks can be waiting on the agenda and it is ineffective to execute them in turn. The interpreter also has to choose a method because more than one method may be included. These selection processes employ various criteria and heuristics, need to trace the history of task execution and demand extra time overhead. To avoid the complicated selection processes, a simpler structure is proposed for a task template , together with a simple task execution strategy. Each task template only contains one method consisting of a task net. The ability of describing task execution is maintained through the task net and introduction of more task templates. A step in a task net contains either another task template or a normal function. The difference between a task template and a function is that a task template can be instantiated and inserted into the task queue and a function can only belong to a task template and be executed as a part of task net but not as a task. However, a function can include any process related to a subtask, such as instantiating another task template, as well as normal computation.

In this aspect, MARCO's task template is similar to TCA's task net. There are two obvious advantages of this structure. First, the selection of methods within a task template is no longer needed as only one method exists. Second, since many processing steps of a task net can be implemented as a function, the number of the tasks in the task queue can be greatly reduced, allowing all tasks to be executed in turn and eliminating task selection process.

In a RAP system, once a method is chosen, all subtasks in its task net are created and put on a task agenda. An ordering relationship imposed by annotations between steps is set-up for these subtasks. A subtask keeps waiting on the task agenda until its preceding subtasks are all finished. In MARCO's sequencing layer, the removal of the selection processes requires that a different task management strategy be used by the task scheduler. Instead of installing all the subtasks, only the subtask created by the current step is inserted into the task queue. To manage the tasks on the task queue, a simple task scheduling technology similar to those used in multiple-process operating systems is employed[Tanenbaum92]. Each task is assigned a state by the task scheduler. A task state represents either the current step of execution in a task net or one of pre-defined states, such as INIT, TIMEOUT, WAIT, SUCCESS, REMOVE. Although all tasks are executed by the task scheduler, the actual execution depends on the current state of the task. Task states can be divided into two categories: executable state or unexecutable state. Executable states include the actual steps of a task net and some of pre-defined states, such as INIT. A task is eligible to run at these states by executing the current task step. Unexecutable state, such as WAIT, means that the task scheduler puts the task back to the task queue to wait without further execution. A task is executed when its state is executable and is removed by the task scheduler when it finishes and enters REMOVE state. At every cycle of execution, the task scheduler executes the tasks in turn, checks their states, executes their current steps or puts them back to the task queue and reset their states according to their execution constraints. A task state is changed when the task enters another step or its running conditions have changed.

With the above scheduling technology, each task is actually a finite-state machine. Subtasks in a task net no longer need to be put on to the task queue at the same time and their ordering relationship can be better preserved. Because the number of tasks in task queue is small, the tasks can be executed very fast. More importantly, it allows the concurrent processing of multiple tasks. Because of the similarity of a task template and a RAP, Firby's task execution constraints can be used unchanged to specify the ordering information for a task and related task net. These constraints include: explicit ordering constraint, temporal constraints and internal state constraints. More details can be found in [Firby89].

Fig. 3-9 presents the task scheduling algorithm used by the task scheduler.

```
For(i=0; i<TASK_NUMBER; i++)
{
    task_state = task[i].state;
    switch(task_state) {
    case REMOVE:
          remove_task_from_queue(i);
            break;
    case unexecutables:
          if( time_to_change)
             task_state = change_task_state(i);
            break;
    case executables:
           execute_task(task_state, i);
           if(time_to_change)
             task_state = change_task_state(i);
            break;
      }
      rearrange_task_queue();
      task[i].state = task_state;
  }
```

Fig. 3-9 Task Scheduling Algorithm

From the algorithm, we can see when a task state is REMOVE, it is removed from the task queue by the task scheduler. This state arises from the success or failure of a task. When a task has successfully finished and method task net wants it to be removed, the state is set. In unexecutable states, a task state may or may not need to be changed. If a task is suspended forever, its state will never be changed and the task will never be actually executed again. But if a task is suspended for 1 second, its current SUSPEND state will become some other state when 1 second passes. The task can then be executed next time as long as its state is executable. The current task state can also be changed during execution. The changed state will become its new state when it is put back to the task queue. The task scheduler continuously executes tasks in the task queue, examines and changes their states correspondingly.

3.3 Summary

This Chapter described a two layer architecture, MARCO for the control of a mobile robot consisting a low level control layer and a sequencing layer and the basic building blocks for the construction of the layers.

The low level control layer is based on fuzzy behaviours, a control structure consisting of a fuzzy logic controller, a soft channel and a behaviour link. The fuzzy logic controller employs a triangular and half-triangular form of fuzzy membership functions and a singleton representation of fuzzy output for the simple representation and fast computation. The method of deriving fuzzy control rules uses the combination of Sugeno's fuzzy modelling of the operator's action and heuristic control knowledge. With fuzzy If-Then rules, human control knowledge can be easily translated to compose the functions of a fuzzy behaviour. A fuzzy behaviour can also accommodate sensor noise, uncertainty and imprecision. Fuzzy behaviours are combined to produce a set of control outputs for a robot through a behaviour selection network. This network can be built on behaviour links and used to redistribute behaviours' activation energy, resulting in the selection of the best suitable behaviour for the robot control.

The sequencing layer organises temporal sequences of controlling activities in the low level control layer. The sequencer is essentially based on Firby's Reactive Action Package(RAP). However, some changes are required to support fuzzy behaviour-based low level control. The sequencing layer consists of a task queue which contains a number of tasks instantiated from task templates. A task template is a simplified RAP which contains only one task net. Steps in a task net contain either a function or another task template. When a task is run, it can do the following things: it can initiate, monitor or terminate a fuzzy behaviour through the behaviour's soft channel, or it can carry out normal computation and insert a new task into the task queue, it also can change the control parameters of a fuzzy behaviour. A task scheduler replaces a RAP interpreter to manage the task execution.

Central to the functioning of the task scheduler is the use of a simple multiple-process scheduling technology. The execution of a task is traced by the task state, either the current step of the task net or a pre-defined state. After every cycle of execution, a executable task progresses one step and is put back to the task queue with a new state while an unexecutable task is put back to the task queue without execution but with possible changed state. The task scheduler executes all the tasks in turn on the task queue without the need of task or method selection. The structure of a task net in a task template and task scheduling method lead to the significant reduction of tasks on the task queue and allows the concurrent execution of tasks.

Chapter 4 Fuzzy Behaviour Organisation and Fusion

This Chapter will describe the organisation and fusion or selection of fuzzy behaviours for the implementation of a low level control layer for mobile robot indoor navigation tasks. This is characterised in the organisation of fuzzy behaviours, based on the sphere of influence of environment features typically presented in an indoor environment, such as corridor, door, etc. The Chapter starts with a brief description of a sensor model and perceptual subsystem which maintains an environment representation. Then it proceeds to discuss how to organise fuzzy behaviours based on the sphere of influence of environment. This will be extensively described through the implementation of several example fuzzy behaviours. These behaviours are combined through the behaviour selection network, which will be discussed in detail. The behaviour fusion and selection algorithm will also be presented.

4.1 A Sensor Model

This section briefly describes a sensor model and a related perceptual subsystem used in MARCO. The sensor model provides the state information of the world which is necessary for the MARCO control system. Because the main focus of the thesis is in the robot control architecture, this discussion will be limited to mainly addressing the needs of feature-based fuzzy behaviours. This is the reason that the sensor model is described before the organisation of fuzzy behaviours, though the sensor model should be shared by other layers of a control system.

4.1.1 Overview

An efficient sensor model should contain different types of information to meet different requirements. Although all control systems require sensor input information, sensor data can be handled very differently, from traditional world modelling, to directly "wiring" sensors to actuators [Brooks89][Connell89]] and the minimum use of state data[Gat91a]. The principle behind the elimination of a centralised world model[Brooks89] is because the centralised world model is prone to error. However, by doing so, the robot has a lack of overall estimation of environments. Some complex and high level tasks especially need such information to complete longer term planning goals. In a behaviour-based architecture, such a world model can also be useful to co-ordinate multiple behaviours, especially when these behaviours are for the purposes of recognition[Saffiotti95]. Without an overall estimation of the world state, the robot can only perform limited tasks[Leonard89].

Traditional sense-model-plan-act architecture has been very much criticised on the ground that planning and acting are explicitly based on an uniform analogical representation of the world[Brooks86]. Sensor data are processed only for constructing a world model upon which the other processing follows. This type of sensor processing has been augmented by little processing and even direct use of sensor data in many mainstream architectures, which results in much faster low level interaction with the environment. Such an approach is employed to implement the MARCO sensor model.

4.1.2 Structure of Sensor Model

In MARCO, an independent perceptual subsystem is responsible for sensor interpretation and world modelling. The perceived sensor data are maintained in a sensor model and shared by the whole system. In the low level control layer, fuzzy behaviours can access these sensor data during their control processes but with different requirements. Some behaviours need more abstracted types of information, such as an analogical representation of environment features; others need only little interpreted data, such as range values. MARCO's sensor model allows such different types of sensor information to be maintained for the different processing needs. The structure of MARCO's sensor model is mainly based on Firby's RAP memory, incorporating the above intuitions to support the control architecture.

Firby uses a RAP memory including a local sensor model and long-term memory [Firby89]. Explicit sensing strategies are provided for the task execution which involves complicated object recognition and manipulation tasks. Similarly, MARCO's sensor model consists of a local sensor model and a long term model which is the same as the long term memory in RAPs. The main difference between the RAP memory and MARCO's sensor model is that MARCO's local model only contains little assimilated sensor data and the RAP's one holds abstract environment descriptions. MARCO's local sensor model supports fast control processes which need only raw sensor data. World modelling is performed by an independent perceptual subsystem which extracts abstract environment information from the local sensor model and registers the information in the long term model. In RAP, the long term memory is maintained by a perception system in a different way. The perception system does not perform world modelling based on a local sensor model but migrates the necessary content from a local sensor model to the long term memory, which involves many migration strategies. Firby does not address how to support tasks which require little interpreted data for fast control process. However, many issues addressed by Firby are equally applicable to MARCO's sensor model.

In MARCO, the long term model contains information provided by a map and also environment features acquired by the perceptual subsystem during navigation. The perceptual subsystem also performs the registration of sensed information to the map data and at the same time localises the robot position. Information in the long term model is the analogical representation of the environment, mainly based on feature descriptions, such as wall, door, corridor, which is similar to Firby's object description. In this thesis, the same strategy used in subsumption-like architectures [Brooks89] [Kaelbling90] is employed to only extract just enough information for navigation purposes, but in the analogical form. A richer analogical representation of the world is left as a future research topic.

4.1.3 Perceptual Subsystem

The functions of the independent perceptual subsystem are sensing, sensor interpretation, world modelling, and localisation. In this thesis, these tasks are treated as standard routines, not controlled by MARCO. This subsystem provides three types of sensor information: raw range data, robot position data and abstract environment features. Raw range data are directly stored in MARCO's local sensor model and then maintained by the subsystem through a local data pool. To keep the pool size constant, old data are abandoned to make space for newly arrived data. Abstract environment features are either extracted from the local pool or retrieved from a map by the perceptual subsystem and then maintained in long term model. In this thesis, a laser range scanner is used as the primary example sensor, which is modelled on a real scanner, a high performance AccuRange3000 laser scanner. The sensing system can produce the maximum 720 points of range data with 0.5° angle resolution in every 40ms. In the example implementation, the perceptual subsystem samples 40 different positions in every scan and these samples constitute the main sources of sensor data for a MARCO sensor model. All sensor data and environment features are based on a robot-centered co-ordinate system which makes it easier to manipulate and maintain sensor information.

4.2 Fuzzy Behaviour Organisation

Having examined the fuzzy behaviour structure, its computational processing and MARCO's sensor model, this section further discusses how to partition the function of a low level control layer into different fuzzy behaviours for mainly indoor navigation tasks. Here, an indoor environment means a relatively "structured" environment where the main features of the environment, such as wall, door, corridor can be easily identified using necessary sensing technologies.

To complete a navigation task, a robot is engaged in two types of activities. First, the robot must interact with its surrounding. Although, the robot's environment can be known prior to task execution, this knowledge is mostly approximate and the environment can also be modified. The robot cannot predict that a person will stand in its route or a small object lies in front of it. The robot may also run into a nearby wall before its perceptual subsystem extracts this feature and initiates an appropriate action. This type of activity is not predetermined by the robot's navigation plan. However, these activities are the by-products of executing a navigation plan and the robot's abilities to deal with these situations are the preconditions of successful navigation. They constitute the robot's basic survival capabilities. In behaviour-based architectures, such capabilities are called reactive behaviours [Gat94] [Saffiotti et al 93b] [Hasemann95]. Their main characteristics are fast response and robustness in dealing with environment contingencies. Sensor input to these behaviours is mostly crude sensor data with little interpretation, indicating the immediate surroundings. The second type of activity involves the robot's purposeful actions with respect to its navigation task. A typical task is to go to a specified position. These capabilities of completing purposeful actions are called task-oriented behaviours[Saffiotti et al 93b][Hasemann95]. The sensor input to these behaviours includes more meaningful description of the parts of the environment and takes a longer processing time to acquire.

In MARCO, reactive and task-oriented behaviours of the low level control layer are organised in the following way. Reactive behaviours constitute basic survival abilities for the robot and provide the ground for the success of task-oriented behaviours. Task-oriented behaviours form individual task performing bodies required for a navigation task. Reactive behaviours can run concurrently and task-oriented behaviours must be chained to complete a task. Each behaviour has its own sphere of influence in relation to the environment. Environment features or task goals are the activation stimulus of behaviours.

4.2.1 Reactive Behaviours

For a navigation task, a robot control system is constantly required to provide the two control outputs, speed and heading, whatever control method is used. Reactive behaviours should be able to control the robot even when no task-oriented behaviours are present. To do so, moving and avoiding hitting objects are the two most important functions. In addition, some purposeful control actions can be implemented without the provision of abstract information by a plan or perceptual system, such as edge following. These actions can also be treated as reactive because of the simple and fast sensor input. Four example reactive behaviours were implemented from the above observations and are presented in this section for discussion. They are Keep Moving, Avoid Obstacle, Follow Edge and Recover Stall. Similar behaviours have been reported in the literature. However, the sphere of influence of environment is employed as the basis of the organisation of different behaviours, as well as the provision of the support for task-oriented behaviours. The sphere of influence of environment is defined for a behaviour with respect to its function. Each behaviour is associated with a sphere of influence of environment. A behaviour can be active when within its sphere of influence and inactive otherwise. Fig. 4-1 shows the sphere of influence of environment for the three example reactive behaviours. Avoid Obstacle behaviour becomes active when the robot is close to any object. Its sphere of influence is the close surrounding of objects. Keep Moving behaviour is active when the robot is in its sphere of influence, an open space. Follow Edge is a quite special behaviour which can be treated as both a reactive and task-oriented behaviour. In the reactive

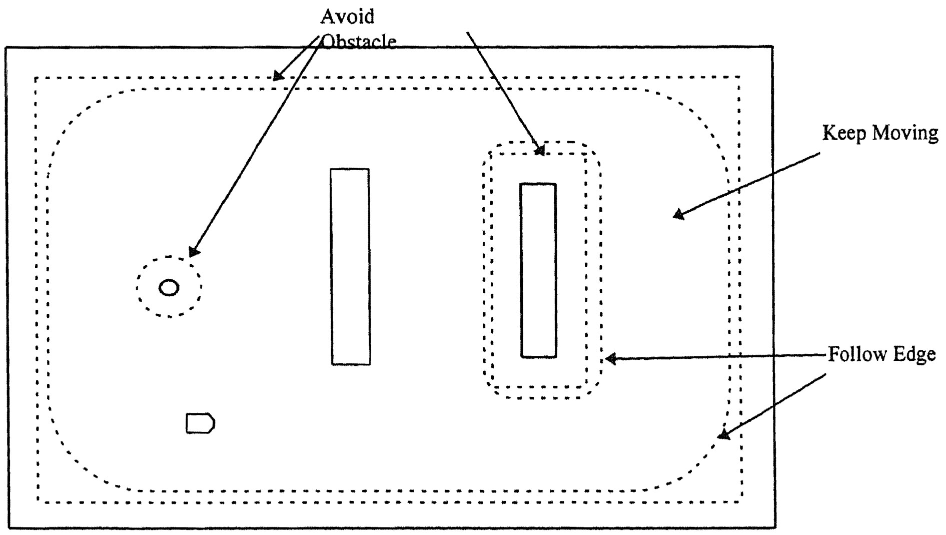


Fig. 4-1 Sphere of Influence of Environment

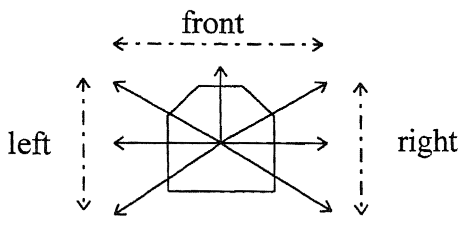


Fig. 4-2 Sample Point Coverage for Avoid Obstacle Behaviour

sense, the behaviour can be activated automatically when the robot moves close to a wall or a relatively linear object sensed but not extracted by the perceptual subsystem. The behaviour acts in a way similar to a rat following an edge of wall. Its sphere of influence is around walls or similar objects. In the task-oriented sense, its sensor input does take a little longer to be produced even though it is not a description of features. It also needs to avoid conflict with other task-oriented behaviours. This dual purpose behaviour can be very useful for some types of indoor tasks. In the following sections, the implementation of these example behaviours is described.

4.2.1.1 Avoid Obstacle

As described in Chapter 2, a fuzzy behaviour consists of three parts: a fuzzy logic controller, a soft channel and a behaviour link. For Avoid Obstacle behaviour, its sensor inputs to the fuzzy logic controller are the range data from the local sensor model, gathered by the robot sensor, in this example implementation, a laser scanner. Local sensor model contains data sampled from positions covering the front, right and left sides of a robot as shown in Fig. 4-2. Newly acquired data are cached into the local sensor model and old data are removed to keep the amount of data constant. Minimum distances to obstacles in the three directions are taken and fed into the fuzzy behaviour. A fuzzy logic controller for this behaviour is implemented using the method introduced in Section 3.1.2.3, the combination of fuzzy modelling of operator's control action and heuristic control knowledge, as well as a trial and error approach. Four fuzzy control rules are used in the behaviour. They are:

If obs_right is CLOSE and obs_left is not CLOSE Then left_heading

If obs_left is CLOSE and obs_right is not CLOSE Then right_heading

If obs_front is CLOSE Then decrease_speed

If obs_front is CLOSE and obs_left is as CLOSE as obs_right Then left_heading

Here, the first two rules prevent the robot from colliding with obstacles at the right and left by generating a heading change truncated by a rule antecedent weight. The third rule slows the robot's speed down when it approaches an obstacle at the front. The final rule forces the robot to turn left and escape when the robot runs into a dead end. The output, *left_heading* can also be replaced by *right_heading* when preferred.

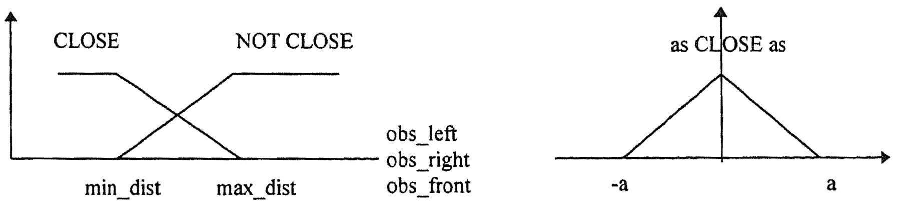


Fig. 4-3 Membership Functions of Fuzzy Variables of Avoid Obstacle Behaviour

The membership functions of fuzzy variables *obs_right, obs_left* and *obs_front* are shown in Fig. 4-3. The membership function CLOSE is determined by two distances, a minimum distance and a maximum distance which marks the beginning of the sphere of influence of environment for the behaviour. Note that the min. and max. distances for *obs_left* and *obs_right* are different from those for *obs_front*, though they have the same structure of the membership function. The membership function, *as CLOSE as*, is used to measure the degree of equality between two fuzzy variables in order to prevent a non-action caused by conflicting control rules. When the robot approaches symmetrically a corner or a straight wall, *obs_left* and *obs_right* usually have a very close measurement value which causes a very close control output for turning left and turning right. The two opposite control outputs result in a non-action. The membership function as shown in Fig. 4-3, measures the situation and the related control rule takes action when it occurs. The above four rules produce synthesised control actions through defuzzification.

In the example implementation, the following state information is defined for each fuzzy behaviour: *running, activity, frustration, achievement.* A behaviour can be enabled or disabled by setting or resetting the state, *running*, only by a higher layer. The state *activity* represents the current activation level of the behaviour, calculated by taking the maximum of all weights of the fuzzy rule antecedents. The state *frustration* indicates the frustration level of the behaviour execution, currently referring to motionless and very slow of the robot movement. Activation and frustration information are accessed by behaviour links in behaviour selection process, which will be described later. The state *achievement* indicates the progress of a behaviour to achieve a goal and is accessible only by a higher layer. For Avoid Obstacle behaviour, the state *achievement* is meaningless.

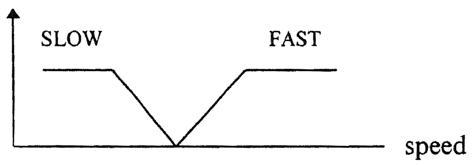


Fig. 4-4 Membership Functions of Speed

4.2.1.2 Keep Moving

This behaviour provides the robot with a constant speed when no task-oriented behaviours are available to activate the speed control. Its main sphere of influence of the environment is an open space. The behaviour is not involved in any heading control. The design of this example fuzzy behaviour is relatively simple. It takes a required normal speed and the current robot speed as sensor input and creates a new speed for output. The fuzzy control rules are as followed:

If speed is FAST Then decrease_speed
If speed is SLOW Then increase_speed.

The first rule decreases the robot speed by the amount with decrease_speed truncated at the antecedent weight when the robot moves too fast. The second rule does the opposite. The membership function of the fuzzy variable, speed, is shown in Fig. 4-4.

Like Avoid Obstacle behaviour, the state *achievement* bears no meaning to Keep Moving behaviour.

4.2.1.2 Follow Edge

This behaviour becomes active when a wall edge is sensed but not extracted by the perceptual subsystem. The purpose of this example behaviour is to help perform some tasks which require the robot to move along the edge of wall or barrier closely. This behaviour can be employed as either a reactive behaviour or a task-oriented behaviour. In some implementations[Saffiotti93], the wall following behaviour is considered as a task-oriented behaviour in which walls are specified as goals for the behaviour to act on. In ATLANTIS's reactive layer, a wall following behaviour is designed as a reactive behaviour using an ALFA circuit. However, its ability is limited. Follow Edge behaviour is quite different from the others[Saffiotti93][Gat91b] [Cheng97]. First, the behaviour can be used as both reactive and task-oriented behaviour. Second, only simple information of a wall is needed instead of an abstract description of a wall. Third, the abilities of the behaviour are enhanced. It can allow the robot to follow a straight edge as well as concave and convex corners without subgoal positions being planned beforehand. The capabilities of the behaviour are illustrated in Fig. 4-5. The behaviour can make the robot control simpler for some tasks along wall edges. The behaviour can also provide support for some task-oriented behaviours, such as goal reaching behaviours. The behaviour can first guide the robot to an easy position around a barrier without high level planning and then be taken over by a task-oriented behaviour. The time needed to plan subgoals can therefore be reduced.

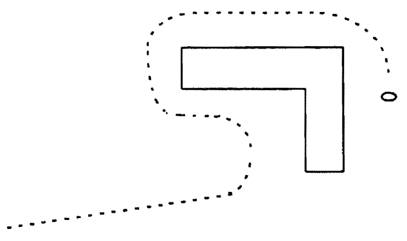


Fig. 4-5 Edge Following Actions

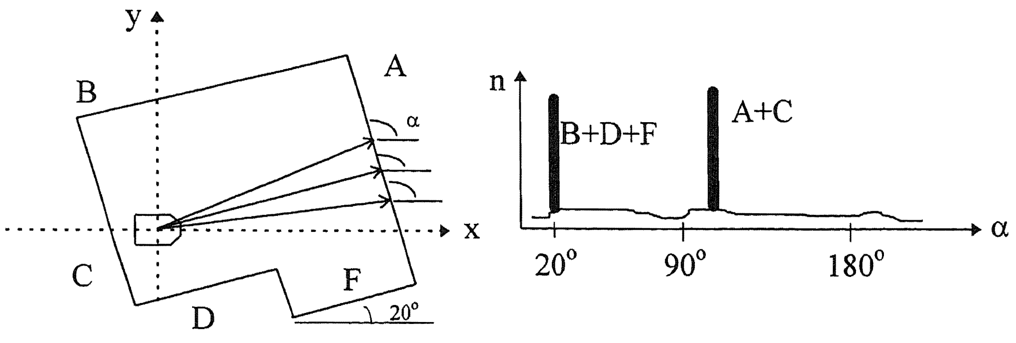


Fig. 4-6 Angle Histogram Calculation

Follow Edge behaviour takes a sensed wall angle and an imaginary track along the wall as sensor input. Locally sensed angle is calculated using an angle-histogram method[Hinkel88]. With the range data provided by a high quality laser scanner, it is possible to have an accurate estimate of a wall angle, though the accuracy of a single measurement is not essential as the robot constantly senses the nearby wall. The angle-histogram has been originally developed for world modelling and localisation [Hinkel89][Hoppen89][Weiβ94]. Angles between two adjacent reflected data point, with respect to the robot current position, are calculated over an entire scan. The main features of the robot environment, usually walls, are reflected through the biggest counts of the same angles. By normalising the angle, two other histograms in X, Y axles can be calculated respectively. Three data sets can be obtained from the calculations, the robot's direction to the wall and its distances to the main walls. The angle-histogram is depicted in Fig. 4-6.

To provide simple and fast information for Follow Edge behaviour, the angle-histogram is used differently here, only to calculate an angle over a small local section instead of an entire scan. The calculation algorithm is given in Fig. 4-7.

```
get_laser_data();
find_min_distance_position();
while(not discontinuous)
    extend_to_certain_local_scope();
calculate_local_angle_histogram();
```

Fig. 4-7 Local Angle-histogram Calculation Algorithm

After the calculation, two angles are selected indicating possible wall segments with maximum reflected points. The structure of a local wall section can be inferred from the data, either a single straight edge, two edges with a concave corner or with a convex corner. A wall angle is selected and an imaginary track is constructed based on the above data. The information gives the robot a little sense about its vicinity without more abstract processing. This method is not suited to sonar based sensing because of its wide beam angle. The calculation takes little time over a small section and is only carried out after a certain distance or an angle change. A wall angle and imaginary track, as well as sensor data from a local sensor model, are provided to Follow Edge behaviour.

The behaviour has six fuzzy control rules as followed:

If speed is FAST Then decrease_speed
If speed is SLOW Then increase_speed
If wdist is NEAR and
(obs_right is not CLOSE and angle is NEGATIVE BIG) Then turn_right
If wdist is NEAR and
(obs_left is not CLOSE and angle is POSITIVE BIG) Then turn_left
If wdist is NEAR and
(obs_right is not CLOSE and angle is not POSITIVE BIG) and
wdist is POSITIVE BIG Then turn_right
If wdist is NEAR and
(obs_left is not CLOSE and angle is not NEGATIVE BIG) and
wdist is NEGATIVE BIG Then turn_left

The first and second rules control the speed of the robot. The rest rules control the robot heading during edge following and only have effects when the robot is near a wall. The third and fourth rules steer the robot to the direction parallel to a wall while the fifth and sixth rules guide the robot towards an imaginary track along a wall. The membership functions of fuzzy variables are provided in Fig. 4-8. Follow Edge behaviour does not need an achievement indicator.

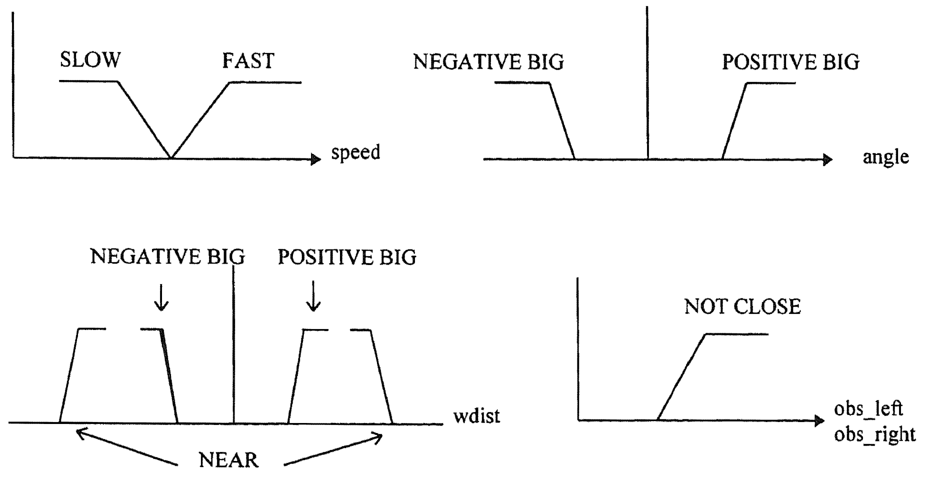


Fig. 4-8 Membership Functions of Follow Edge Behaviour

4.2.1.3 Recover Stall

This reactive behaviour is active when the robot is trapped in a local minima and all other behaviours fail to pull the robot out of a motionless state. The behaviour uses the robot movement states as inputs and exercises an escaping strategy with one control rule as followed:

If move is SMALL or stop_time is LONG Then increase_speed.

The control output is the escaping speed which stimulates the robot to move again. The behaviour is automatically activated when other fuzzy behaviours contribute their activation energy to it through their behaviour links.

4.2.2 Task-Oriented Behaviours

Task-oriented behaviours are responsible for completing the robot tasks. In this thesis, these tasks are mainly navigation from place to place in indoor environments, such as office building, construction sites, etc. A typical task is to go to a specified position. Another such task could be to find and then enter a designated room. Two types of task-oriented activity can be involved in the navigation. One is the direct control action for completing a given task. For example, Reach Position task causes the robot to steer directly towards the direction leading to the position. The other is indirect control activity which may or may not directly result in the completion of the task goal but provides the support for the robot to complete its final goal. Following a corridor is not the activity that can direct the robot to enter a room. However, without this purposeful corridor following activity to guide the robot near the door, the robot cannot find and then enter the room. Of course, the robot can wander around and try to find the room itself. But this is not an efficient or even feasible way in a complex environment. These two types of activity should be supported by task-oriented behaviours.

To support indoor navigation tasks, some of task-oriented behaviours are specifically organised based on the typical indoor environment features, such as corridor, door. These behaviours can be activated by a higher level when their related environment features are present in the long term model. The sphere of influence of environment for such a behaviour is from the related feature to the current robot position. In the example implementation, four task-oriented behaviours were developed. They are Follow Corridor, Track Path, Cross Door, Reach Position. These behaviours can be used for both the direct or the indirect control purpose of completing the robot task. Their input data are provided in the long term model, though corridor and door features are extracted by the perceptual subsystem or provided through a map while path and position

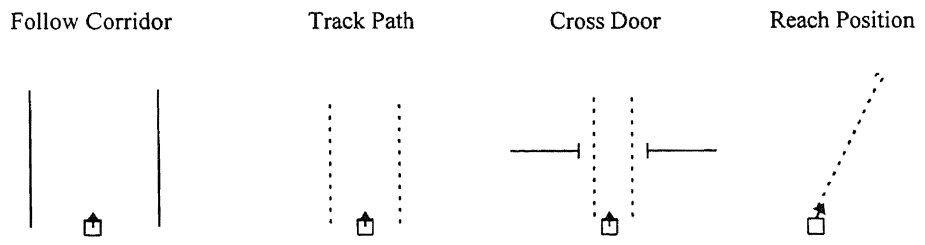


Fig. 4-9 Acting Pattern of Four Task-oriented Behaviours

data are given by a high level planning system or human. Their control actions are illustrated in Fig. 4-9. Note that Follow Corridor, Track Path and Cross Door behaviours have the same pattern of control actions, but use different input data. Follow Corridor behaviour controls the robot to follow an actual corridor comprising of two parallel walls while Track Path behaviour is for following a lane which can be seen as an imaginary corridor. Cross Door behaviour guides the robot in or out of a door way which can also be abstracted as a short corridor. The same set of control rules is employed to implement the three example fuzzy behaviours. Their different perceptual features can be converted to a similar data structure representing a passage way which can then be used by the same fuzzy logic controller. Here, Follow Corridor and Reach Position behaviours are presented.

4.2.2.1 Follow Corridor

Follow Corridor behaviour uses a corridor feature from the long term model as well as range values from the local sensor model as sensor input and controls the robot to move along the centre of the corridor. A corridor feature is extracted by the perceptual subsystem or provided through a map. It consists of two parallel wall segments and has a certain width and length constraints. The perceptual subsystem extracts a corridor feature according to these constraints and stores it in the long term model. Corridor following is a common behaviour for indoor navigation. The behaviour is implemented with the following fuzzy control rules:

If speed is FAST Then decrease_speed
If speed is SLOW Then increase_speed
If lane_dist is NEAR and
  (obs_right is not CLOSE and angle is NEGATIVE BIG) Then turn_right
If lane_dist is NEAR and
  (obs_left is not CLOSE and angle is POSITIVE BIG) Then turn_left
If lane_dist is NEAR and
  (obs_right is not CLOSE and angle is not POSITIVE BIG) and

```
  (lane_dist is POSITIVE BIG) Then turn_right
If lane_dist is NEAR and
  (obs_left is not CLOSE and angle is not NEGATIVE BIG) and
  (lane_dist is NEGATIVE BIG) Then turn_left
If lane_dist is not NEAR and
  (obs_right is not CLOSE and lane_dist is POSITIVE BIG) Then turn_right
If lane_dist is not NEAR and
  (obs_left is not CLOSE and lane_dist is NEGATIVE BIG) Then turn_left.
```

Fuzzy Variable *lane_dist* refers to the perpendicular distance from the robot to the centre line of a corridor. The distance has positive value when the robot is at the left and negative at the right. The fuzzy variable *angle* is the corridor angle in a robot centred co-ordinate system. If negative, the robot points to the left of a corridor central lane, otherwise to the right. The first two rules control the robot following speed. The 3rd and 4th rules steer the robot to the corridor direction when the robot is in the lane. The 5th and 6th rules guide the robot back to the track when the robot drifts to the two sides of the central lane but remains close. The 7th and 8th rules forces the robot towards the track when the robot is very much out of the position. Follow Corridor behaviour has four state indicators: *running, activation, frustration* and *achievement.* The state *achievement* indicates the progress of following to the end position of a corridor. Once the robot arrives at the position, the behaviour is removed. Note that the first six rules are very similar to those of Follow Edge behaviour. They also have similar membership function structures.

4.2.2.2 Reach Position

Navigation means that a robot moves from a starting position, negotiates with its environment and finds its way and moves to a goal position. Getting to a specified position is a common task of a mobile robot navigation system. In the above sections, other example behaviours are described which control the robot survival in its' environment and finding its way. This section describes one of the most important task-oriented behaviours, Reach Position. With the other behaviours taking care of survival and path following activities, the design of this behaviour becomes relatively simple. The behaviour is only responsible for steering the robot towards a goal position. The control strategies of the behaviour is similar to a potential field goal reaching behaviour[Arkin90][Payton90][Slack93]. A goal position presents an attractive force to the robot, expressed as a speed vector. The robot approaches the goal with the same speed as the magnitude of the vector and heading of the vector. The sphere of influence of the goal reaches as far as the robot position. Therefore, the farther the robot is away from the goal, the stronger the attractive force. The robot moves towards the goal position under the constraint of maximum velocity.

Reach Position behaviour needs an X/Y goal position in robot co-ordinates, the robot speed and range data from the local sensor model are used as sensor input. The goal position is provided and placed in the long term model by a high level planning system or human. Because of the use of the local robot co-ordinates, the goal position is updated towards the origin of the co-ordinates, making it easier to check the progress. The fuzzy behaviour employs six rules as followed:

If speed is FAST Then decrease_speed
If speed is SLOW Then increase_speed
If angle is POSITIVE MEDIUM and dist is not SMALL and
(angle is BIG or obs_left is not CLOSE) Then turn_left
If angle is NEGATIVE MEDIUM and dist is not SMALL and
(angle is BIG or obs_right is not CLOSE) Then turn_right
If dist is very SMALL Then decrease_speed_stop
If angle is BIG Then decrease_speed_stop

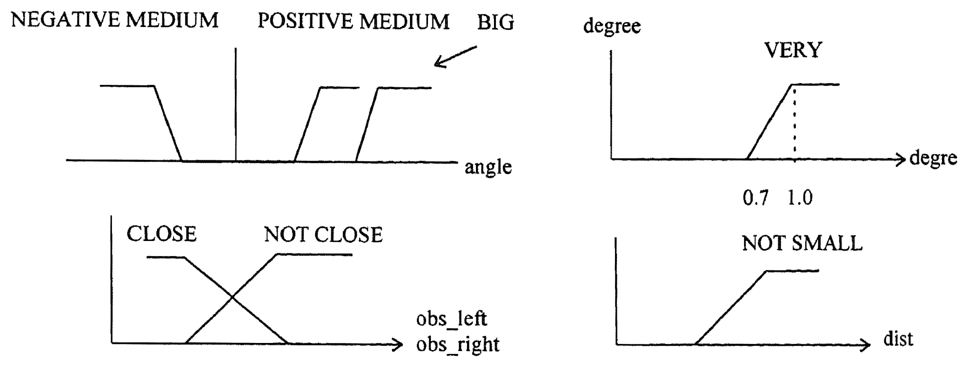


Fig. 4-10 Some Membership Functions of Fuzzy Variables of Reach Position

The first two rules are for the control of the speed. The 3rd and 4th rules steer the robot towards the correct direction when the heading deviation is not big. The 5th rule decreases the robot speed sharply when the robot arrives at the goal. The 6th rule also slows down the robot sharply when the heading deviation is too big, allowing the robot to turn to the correct direction first. In the 5th rule, a fuzzy hedge *very* is used to reduce the degree of the truth of SMALL to intensify the sense of the closeness to a goal. Its membership function is depicted in Fig. 4-10, together with some other variables. The state *achievement* is indicated by the arrival at the goal position.

4.3 Fuzzy Behaviour Fusion

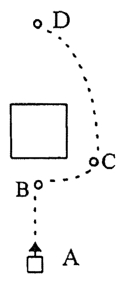


Fig. 4-11 An Example of Behaviour Coordination

A central problem for an autonomous mobile robot operating in uncertain and dynamic environment is how to combine task-oriented activities with reactivity. For instance, a mobile robot should reliably avoid unforeseen or moving obstacles during task-oriented navigation. The previous sections have described the organisation and possible implementations of behaviours individually responsible for reactive and task-oriented control activities. To examine how we should co-ordinate these activities, let us see one example presented in Fig. 4-11. The robot is asked to reach position D from its current position A. The robot is first controlled by Reach Position behaviour which assumes there are no obstacles ahead of the robot. When the robot moves close to the obstacle at the position B, the obstacle causes Avoid Obstacle behaviour to become active. Now there are two active behaviours that have conflict over the control output for the robot movement. Reach Position behaviour tries to steer the robot towards D while Avoid Obstacle behaviour produces the opposite escaping heading output. Because they have the same activation strength, this contradiction leads to a non-action or oscillation, called a local minima, in the robot movement. The robot totally freezes or oscillates endlessly at the vicinity of the obstacle.

To resolve such contradiction, one method is the use of global path planning [Arkin87] [Payton90] [Garcia-Alegre93] [Vandorpe94], especially when the obstacles are complex and sensing can give an accurate picture of this complexity. As global path-planning is the task at the highest level of a robot control system and not the focus of this thesis, another method, local combination or co-ordination of behaviours is considered. This method, in one form or another, is widely used in the robot literature for mixing goal directness and reactivity. In Brook's subsumption architecture[Brooks89], behaviours are organised hierarchically as layers, with reactive behaviours at lower layers and goal directed behaviours at higher layers. The control actions of the robot are produced by co-ordinating multiple layers by means of a suppression mechanism. Higher layers subsume the roles of lower layers when they wish to take control. This suppression mechanism is realised through hard wiring between layers. The switching of behaviours is not smooth. The similar scheme is used in ALFA[Gat91a] in which behaviour switching is realised through "circuit" channels. Arkin[Arkin90] uses a weighted averaging scheme to combine reactive and goal-directed behaviours. Each behaviour is a motor schema functioning in a potential field. A fixed weight is assigned to each behaviour. The vector forces produced by behaviours are then weighted and combined using a potential field summation to produce the final control. While it is possible to create a smooth control action, to select and adjust weight can be difficult. This method can also be problematic when a decisive control action is required because of its summation nature. Saffiotti proposed a method which is somehow a combination of Brook's hierarchical switching and

Arkin's weighted averaging. This method is called context dependent blending and is used to fuse fuzzy behaviours[Saffiotti et al 93a]. Each behaviour is prioritised according to its importance in the navigation activity and is associated with a desirability function which determines the applicability of the behaviour given the current context of the environment. The desirability function is created by a T-norm operation based on the behaviours' activation level and its priority. Higher priority behaviours can suppress lower priority behaviours by means of desirability functions. The control action is produced using weighted summation in which the weight is actually the desirability measure of the behaviour. This approach is very effective in producing smooth goal-directed control output in the face of a dynamic and uncertain environment. However, local minima still exist because of the lack of a dominant control action which can lead the robot out of such local equilibrium point[Saffiotti95]. The success of the scheme depends on detailed task planning. Another approach reported is fuzzy multiplexing [Goodridge94], which uses an additional fuzzy controller to perform the weight assignment to fuzzy behaviours. Qualitative rules are used to determine the gains for each behaviour using sensor input and behaviour state information. The difficulties lie in the derivation of the selection rules under various circumstances. This method is only useful if behaviours are not mutually incompatible and can be safely blended by a weighted summation[Goodridge94].

In the following sections, a different behaviour fusion scheme is described to combine fuzzy behaviours in MARCO's low level control layer. Before proceeding to the detailed discussion, the organisation of MARCO's fuzzy behaviours is revisited briefly. Fuzzy behaviours are organised into reactive and task-oriented behaviours according to the sphere of influence of environment features. These behaviours use sensor information from the local sensor model and the long term model and perform control processes with their fuzzy logic controllers. The output of a fuzzy behaviour includes the control output for a robot actuator and its state information which can be accessed by the sequencing layer and behaviour links. In particular, two state variables are provided to indicate a behaviour's activation level and execution frustration level, and can be used by behaviour links. The robot survival ability is strengthened by a failure-recovery behaviour. This organisation needs a behaviour fusion scheme which can facilitate the individual robustness of fuzzy behaviours and also maintain an effective, smooth transition of behaviour control towards task achievement. Such a scheme has been developed to fuse MARCO's fuzzy behaviours. The scheme is called a behaviour selection network and is inspired by Maes's work[Maes90] in artificial life research.

Maes used a bottom-up mechanism to select behaviours for artificial low level animals, such as bug, hen, etc. In her method, the selection of a behaviour is based on the internal motivational states of a creature as well as external circumstances. The different behaviours of a creature are linked in a network with "predecessor", "successor" and "conflicter" links. Through these links, behaviours activate and inhibit each other, respectively increasing and decreasing each other's activation level. At the same time, the activation energy accumulates in a behaviour that represents the "best" choice, given the current situation and motivational state of the creature. Once the activation level of a behaviour reaches a certain threshold, it may be selected, and its processes start operating. In designing MARCO's behaviour selection network, the concepts of situational and motivational activation/inhibition and a bottom-up, not centrally controlled, selection dynamics are adopted. The behaviour selection scheme is however, realised in a very different way. Maes's method is used in artificial low level creatures that have no specific goal during their activity. MARCO's selection scheme is mainly to create task-oriented navigation. Maes assigns a set of integral numbers as situational and motivational activation levels. MARCO utilises smoothly changing fuzzy predicates produced during fuzzy control processes instead. In Maes's method, the activation energy is distributed through predecessor, successor and conflicter links. MARCO uses behaviour promotion/inhibition links. Another difference is that no threshold is needed to select a behaviour. The behaviour with the highest activation level is always selected. As a result, the activation level of a behaviour changes more naturally and smoothly as the environment and the robot state change, enabling an effective and smooth change of behaviour control.

4.3.1 Behaviour Promotion/Inhibition Links

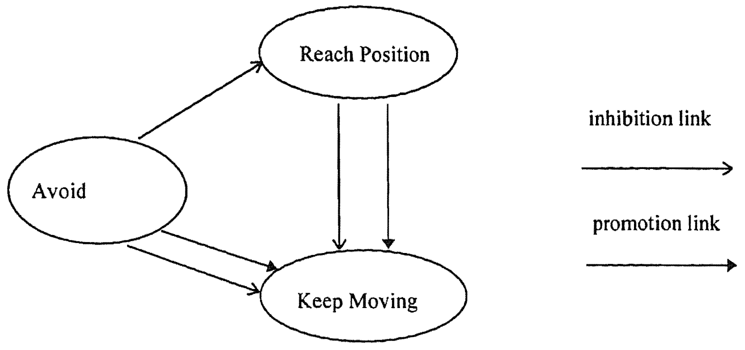


Fig. 4-12 An Example Behaviour Selection Network

Fuzzy behaviours in MARCO low layer are linked into a network through behaviour links. A behaviour link is a "soft" data structure which distinguishes itself from the suppression "circuit" used in subsumption and other similar architectures [Brooks89] [Gat91a][Kaelbling88]. As described in the fuzzy behaviour structure in Section 3.1.3, each fuzzy behaviour has such an entity to contain its relationships with other fuzzy behaviours. There are two types of relationships between fuzzy behaviours: promotion and inhibition, separately represented by promotion and inhibition links. An example of a network connected with behaviour links is shown in Fig. 4-12. Through a promotion link, a fuzzy behaviour increases the activation level of linked fuzzy behaviours and at the same time decreases its own activation level. The purpose of a promotion link is to distribute a behaviour's activation energy to other behaviours in order to satisfy the motivation of the overall robot control system. A behaviour can also use an inhibition link to decrease the activation level of the linked behaviours in order to have more chance to control the robot.

4.3.2 Behaviour Activation

The activation level of a fuzzy behaviour in the MARCO control layer, which determines whether or not it can be selected, consists of two parts: situational activation and motivational activation/inhibition. The situational activation level depends on a behaviour's current environmental conditions, or the sphere of influence of environment. Motivational activation/inhibition is the method used to distribute activation energy through behaviour links.

4.3.2.1 Situational Activation

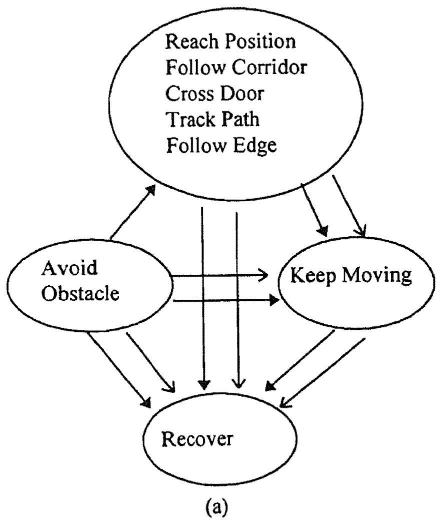
Each behaviour is associated with a situational activation level which is an actual behaviour state, *activation.* This level is continuously changed as the robot's environment and moving state change. In the example implementation, the value of the activation level is calculated by taking the maximum fuzzy predicate of the antecedent parts of all the fuzzy rules of a behaviour. This maximum fuzzy predicate reflects the highest degree of the influence of the environment to the behaviour. For example, suppose that an obstacle is very close to the left of a robot and the fuzzy input variable *obs_left* has the value of 1.0, the full strength of closeness and *obs_right* and *obs_front* has 0.0 and 0.5 respectively. The situational activation level of Avoid Obstacle behaviour is therefore 1.0. Under normal situations when motivation activation/inhibition has little influence, the activation energy of a behaviour is mainly determined by the situational activation level.

4.3.2.2 Motivation Activation/Inhibition

Motivations are defined for the overall robot control system. Fuzzy behaviours distribute their activation energy through behaviour links to reflect the current motivation of the robot control system. Three types of motivations have been defined in the example implementation. They are task completion, safety and aliveness. Task completion is the motivation for achieving a goal and is used to support task-oriented behaviours. Safety is the motivation for survival and not crashing into the environment and is employed to strengthen the robot survival ability. Aliveness is the motivation to keep the robot alive; the robot should not stop, stall or move very slow. These motivations are supported through the organisation of different fuzzy behaviours and behaviour selection network. Motivational activation causes the distribution of one behaviour's activational energy to other linked behaviours when its execution is frustrated by the current situations in order to safeguard the overall interests of the current robot control, i.e., motivation. On the other hand, motivational inhibition helps a behaviour to subdue other behaviours by decreasing their activation energy. This inhibition also serves the current need of the robot control. At any time, a behaviour's overall activation energy is determined by three types of activation energy: situational activation, motivational activation and inhibition. There is a continual flow of activation energy among behaviours in matching the current situation and motivation of the robot control. After energy redistribution, the behaviour with the highest activation energy represents the best one matching the current robot control requirement. This behaviour is then selected to control the robot.

4.3.3 Behaviour Selection Network and Algorithm

The behaviour selection network consists of all the links among behaviours. In a MARCO control system, only one task-oriented behaviour is allowed to exist in the low level control layer with several reactive behaviours. However, all of the task-oriented behaviours have similar links to the reactive behaviours. Fig. 4-13a shows a complete behaviour selection network for the example fuzzy behaviours in MARCO's low level control layer. Fig. 4-13b shows some of possible subnetworks of



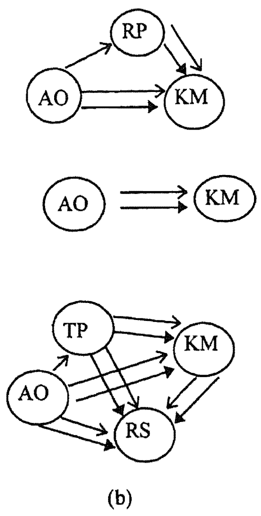


Fig. 4-13 Behaviour Selection Network
(a) a complete network; (b) some possible subnetworks:
AO- Avoid Obstacle, KM-Keep Moving, RP-Reach Position, TP- Track Path, RS - Recover Stall.

behaviour selection.

Note that Avoid Obstacle behaviour and Keep Moving behaviour are always needed in any combination of behaviours. The two behaviours are bonded together to provide basic speed and heading control.

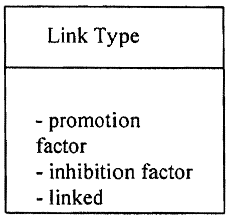


Fig. 4-14 Behaviour Link Structure

The behaviour link which connects the behaviour has a data structure shown in Fig. 4-14. Link type defines the type of the link, promotion, inhibition or promotion/inhibition. The last one means that both links exist between the linked behaviours. Promotion factor determines the level of promotion with respect to promoting the behaviour's current activation level. Inhibition factor determines the opposite in decreasing the recipient behaviour's activation energy. Linked behaviour is the name of a linked behaviour. To select a behaviour, the following information is needed:

a_level — activation level of a behaviour;

a_level_S — situational activation level;

f_level — frustration level of execution;

i_value_i — inward inhibition energy, to be decreased from the recipient behaviour's;

i_value_0 — outward inhibition energy, to decrease the recipient behaviour's;

p_value_i — inward promotion energy, to be increased in the recipient behaviour's;

p_value_0 — outward promotion energy, to increase the recipient behaviour's;

i_factor — inhibition factor;

p_factor — promotion factor.

Fig. 4-15 presents the behaviour selection algorithm. Here a more detailed description of the selection process is given. For every currently active behaviour in a network, its initial activation level is the same as the behaviour's state value, *activation*. Its final activation level is determined by the redistribution of activation energy through the

```
for(i=0; i< CURRENT_BEHAVIOUR_NUMBER; i++)
{
        initialise();
        a_level[i] = a_level$_s$[i];
        for(j=0; j<LINKED_BEHAVIOUR_NUMBER; j++)
        {
           if(LINK_TYPE == PROMOTION)
           {
              p_value$_o$[j] = f_level[j] * p_factor[j];
              p_value$_i$ [i] += p_value$_o$[j];
              a_level[j] -= p_value$_o$[j];
            }
           if(LINK_TYPE == INHIBITION)
           {
              i_value$_o$[j] = a_level$_s$[j] * i_factor[j];
              i_value$_i$ [i] = MAX(i_value$_i$ [i], i_value$_o$[j]);
            }
         }
        a_level[i] += p_value$_i$ [i] - i_value$_i$ [i];
        a_level[i] = MIN(a_level[i], 1.0);
 }
  selected_behaviour = get_behaviour_with_largest_a_level();
```

Fig. 4-15 Behaviour Selection Algorithm

network. The distributed activation energy may consist of two parts: an increased portion and a decreased portion. The increased portion, called inward promotion energy, comes from all promotion-linked behaviours which also reduce their activation energy by the same amount, outward promotion energy. The decreased portion, called inward inhibition energy, is produced by selecting the maximum outward inhibition energy from all inhibition-linked behaviours. The final activation energy is the sum of situational activation energy and the distributed portions of energy. This final activation level is clipped at the full strength 1.0. The behaviour with the highest activation level is then selected. The final control output is produced by fusing the activation level and fuzzy control output through multiplication. Because the number of active behaviours is small(maximum of 4) and the behaviour selection process is also not complex, the algorithm takes very little time to select a behaviour.

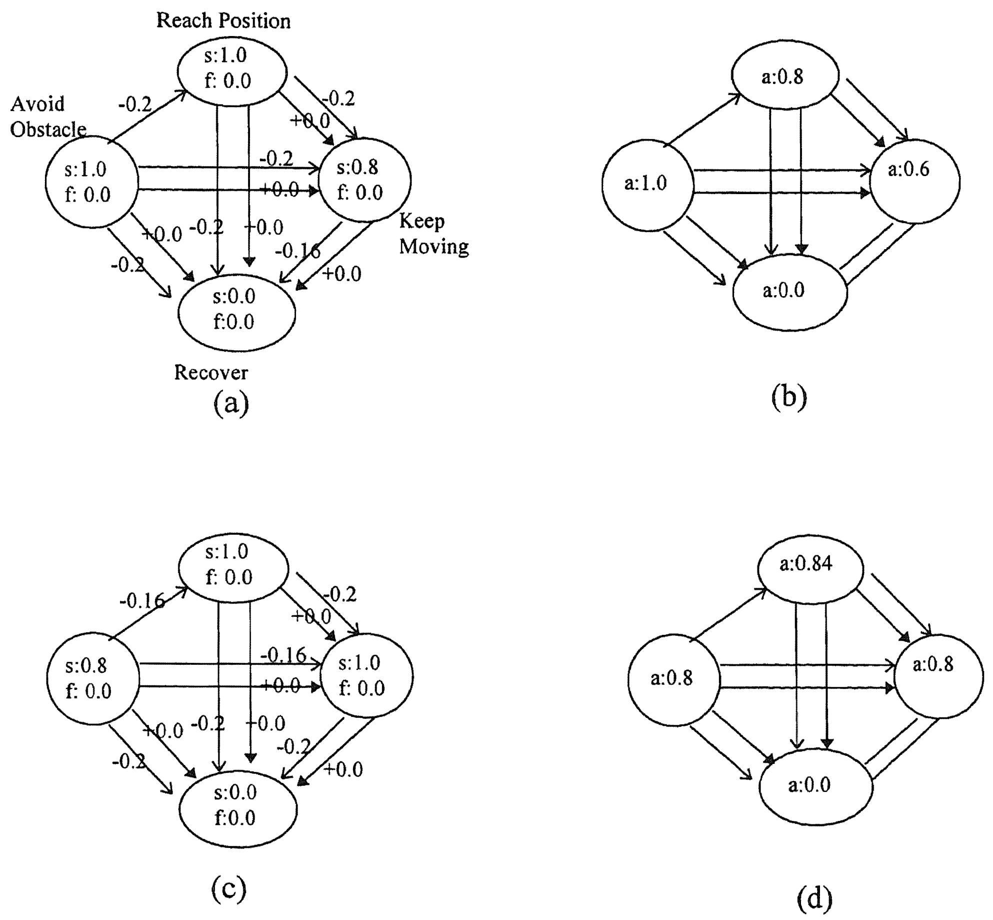


Fig. 4-16 Energy Redistribution Process for Robot Control in Fig. 4-11.

(a)(b): behaviour selection at position B;

(c)(d): behaviour selection at position C;

s: situational activation level, f: frustration level, a: activation level.

+: promotion, -: inhibition.

For example, in Fig. 4-11, the robot is controlled by three reactive behaviours and one task-oriented behaviour. Suppose the promotion and inhibition levels are set as 15% and 20%, respectively. At position B, the situational activation levels are 1.0, 1.0, 0.8, and 0.0 for Avoid Obstacle, Reach Position, Keep Moving and Recover Stall respectively. At position C, they becomes 0.8, 1.0, 1.0 and 0.0. Fig. 4-16 gives the energy redistribution process in the behaviour selection network. Fig. 4-16(a) shows the situational activation level and their distributed energy at position B. The final activation energy is indicated in Fig. 4-16(b). As a result, Avoid Obstacle behaviour is selected to control the robot at position B, while Reach Position behaviour is selected at position C, which is shown in Fig. 4-16(c) (d). At position B, the robot is close to the obstacle. It must move away to avoid collision and temporarily abandon the reaching position task. At position C, the robot moves out of the danger of collision and the main task is resumed. The behaviour selection network ensures that the correct behaviours take control of the robot at all times in order to complete a task.

4.4 Summary

In completing a navigation task, a mobile robot is involved in two types of activities: basic surviving control activity in face of dynamic and uncertain environments and purposeful control activity leading to the task accomplishment. In the MARCO control layer, the capabilities of carrying out these control activities are constructed into fuzzy reactive and task-oriented behaviours with the following intuitions. Reactive behaviours constitute the basic survival abilities for the robot and provide the grounding for the success of task-oriented behaviours. Task-oriented behaviours form individual task performing bodies and can be sequenced to complete an ultimate task goal. These fuzzy behaviours are then organised, based on the sphere of influence of environment features. Every behaviour is associated with one type of environment feature and becomes executable when such a feature is available in MARCO's local sensor model or long term model. Such an organisation supports the robot's direct interaction with the environments. The efficiency of the organisation can be further improved by allowing behaviours to access different abstract information from different sensor space, either for fast or more abstracted processing. With feature-based organisation, indoor navigation control can also be easily implemented by the introduction of fuzzy behaviours associated with rich types of indoor environment features. Several such example fuzzy behaviours are implemented and described to give an indication of how fuzzy logic-based behaviours can be organised for possible indoor tasks using the above mentioned approach.

While allowing multiple behaviours to be active at the same time, conflicting control actions from behaviours must be resolved through behaviour fusion or selection. A behaviour selection network has been developed for this purpose. Behaviour promotion/inhibition links are designed to introduce situational activation and motivational promotion/inhibition among behaviours to redistribute their activation energy. This selection dynamics operates from the bottom-up and is not centrally controlled. At any time, the most favourable behaviour is selected for the robot control with respect to the current environment conditions and the motivations of the control system. The final control output is then produced by fusing the accumulated energy with the behaviour output through multiplication. Because of the use of fuzzy predicates created during fuzzy control processes, the activation energy flows continuously and smoothly among the behaviours. This results in the effective and smooth transition of the robot control among the behaviours in order to accomplish tasks.

Chapter 5 Learning of Optimal Mobile Robot Control Behaviours

5.1 Introduction

Navigation through a dynamic and uncertain environment to a specified destination without hitting objects is a complex task. The robot control system must be robust enough to cope with various possible environmental conditions. In developing such a robust system, a reactive control approach has proved to be superior to a traditional approach. A fuzzy behaviour based reactive control system is more capable than the systems based on the other methods[Saffiotti et al 93a][Garcia-Alegre93] [Goodridge94]. The main advantages of such systems is that expert knowledge and human experiences can be easily translated into fuzzy control rules of fuzzy behaviours. A fuzzy logic controller is also capable of accommodating approximate, imperfect and noisy information presented in real world environments and producing a smooth control output [Saffiotti et al 93a] [Vandorpe94][Garcia-Alegre93]. Developing a non-fuzzy logic based reactive control system requires the selection and structuring of the control parameters that underlie the behaviours of the robot [Pearce92][Arkin87]. Similarly, a fuzzy behaviour-based reactive system needs the selection and tuning parameters which characterise the control rules for a fuzzy behaviour, and also the weights which affect fuzzy behaviour selection in the behaviour selection network. A fuzzy logic controller of a fuzzy behaviour consists of several fuzzy control rules, each of which may contain several fuzzy variables. The performance of the fuzzy logic controller depends on the appropriate design of all the membership functions of all the fuzzy variables. One set of membership functions may be effective or optimal for some control rules but may have adverse effects on the other control rules. The selection or adjustment of these membership functions have to be carried out to improve the overall performance of all the control rules. Generally, the selection and tuning of membership functions has been based on knowledge derived from imprecise heuristic knowledge of operators or control process[Sugeno85][Lee90]. Because this is mostly a manual process, it is difficult to obtain an optimal set of fuzzy membership functions for a fuzzy logic controller [Cooper93]. A lot of effort is needed to configure them, usually by trial and error methods, and often the results are still far from optimum. The robustness of individual fuzzy behaviours developed in such approach is limited. Apart from this, obtaining an optimal behaviour selection strategy is also a difficult task. A mobile robot can face various environment conditions during navigation. The behaviour selection mechanism should be able to produce effective and smooth control transition between fuzzy behaviours under all these conditions. Manually chosen behaviour selection parameters may work well under some environments. However, it cannot guarantee the effectiveness for a whole range of environment types. This will, therefore, affect the robustness of the robot control system. Facing the above difficulties, it is necessary to find a systematic approach for the design of optimal fuzzy behaviours and the behaviour selection mechanism. Chapter 5 describes such an approach for building robust mobile robot control behaviours. In particular, the approach, based on genetic algorithms, is used to develop a robust low level control layer of MARCO architecture.

In the remainder of this Chapter, the genetic algorithm learning technology is reviewed. The learning methodology based on genetic algorithms is then described and simulation experiments in learning fuzzy behaviours and behaviour selection network are discussed.

5.2 Genetic Algorithms

A genetic algorithm is a search technique modelled after natural evolution, where survival of the fittest is the principle. Genetic algorithms were first presented by Holland as a component of a larger framework called a classifier system[Holland75]. The genetic algorithm was used as a mechanism to evolve new elements which contribute most to improve the survival of the system in a non-stationary environment. In 1975, DeJong separated genetic algorithms from the classifier system and treated it as a function optimisation technique[Goldberg89]. From this study, it appeared that genetic algorithms were better alternatives to conventional optimisation methods. This was demonstrated by Goldberg[Goldberg89] that genetic algorithms could be used as a standalone multi-dimensional optimisation technique.

5.2.1 Basic Process

A genetic algorithm(GA) is a population-based search and test method. Multiple solutions are generated and then evaluated in parallel. Solutions to be evaluated in the next generation are constructed by taking the good solutions in the current population and mixing them. The basic process is outlined as follows:

(1) Generate initial population of solutions. Initially, all members of the population are randomly initialised;

(2) Evaluate members of population and assign each a fitness value. The fitness value will be used to guide reproduction process;

(3) Generate the next generation. Use genetic operators to select and construct new solutions from the existing population of solutions;

(4) Go to step 2 until some stopping criteria is met. The stopping criteria could be one when the best solution reaches a given performance measurement or the process has passed a given number of generations.

Unlike some conventional search techniques, a GA considers a space of search points for an optimal solution. Therefore, the chance of converging to local optima is reduced[Goldberg89]. Furthermore, a GA simply requires that a solution can be represented as a string of element, not a complicated function. This makes GAs attractive for various applications. A simple and interesting example has been presented in Dougal(Demonstration Of Using Genetic Algorithm Learning) [Parker93], in which a genetic algorithm was used to search for an optimal round trip route for students planning inter-railing holidays to a dozen European cities.

5.2.2 Genetic Representation and Operators

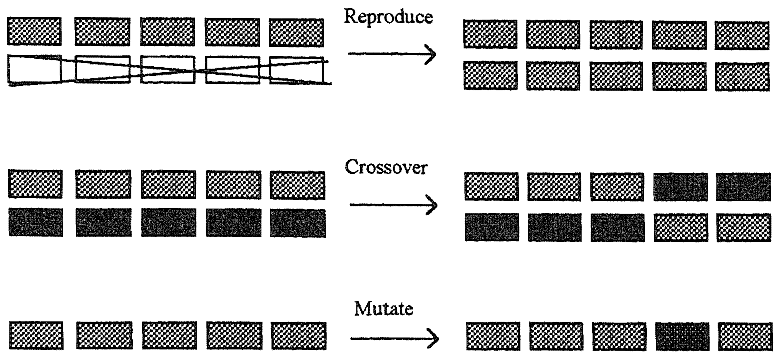


Figure 5-1 Genetic Operators

Genetic algorithms apply their operators to a representation of the search space points. In a traditional GA, the representation is a position-dependent bit-string, where each bit is a "gene" in the string "chromosome"[Goldberg89]. The choice of bit strings allows chromosomes to be conveniently cut into substrings, enabling the exchange of information between individuals. Typically, each generation of the GA begins by decoding the bit-string into search space points and using the search function to evaluate the fitness of the individual. Once the population has been evaluated, a set of genetic operators is applied. The three most commonly used are reproduction, crossover and mutation. These operators are expressed graphically in Fig. 5-1. Note that each of the rectangles in the figure represents a single bit of string. In practice, most representation use much longer strings.

The reproduction operator selects the fittest individuals and copies them exactly, replacing less-fit individuals so the population size remains constant. This increases the ratio of good individuals to the number of poorly-performing ones. The selection process uses a weighted roulette wheel, or biased selection; the best individuals are preferred, but not guaranteed, to be reproduced.

The crossover operator allows two individuals to exchange information by swapping some part of their representation. This creates a pair of new individuals that may or may not perform better than the parents. For example, if the string [0000000] was crossed with string [1111111], the result might be [0001111] and [1110000]. The choice of which individuals to cross and where to cut the chromosome is random. This random search component gives GAs much of their power[Goldberg89].

The mutation operator is mainly used to prevent the loss of information that occurs when a population cannot improve because all of the individuals in the population have the same value for a given gene. Since no amount of selection or exchange of the same value will change it, mutation allows lost information to be recovered, and further, maintains variety during convergence.

A GA can be thought as a search method that exploits points in the search space that have already been reached and explores other points that are yet to be tried. The reproduction operator exploits the knowledge present in the population by increasing the numbers of fitter individuals. The crossover operator explores the search space by producing new points to evaluate. This simultaneous exploration and exploitation moves the algorithm toward populations containing the fittest substrings in the fittest combinations. The GA eventually settles on a set of optimal or multiple sets of near-optimal individuals. The convergence time and solution quality depend on the nature of the problem and the parameters that control the GA.

5.3 Robot Learning

There are several factors to be considered in designing a robot navigation system that learns. To ensure adequate generalisation of a given environment, many trial runs are required during training. Due to time and costs for both robot and instructor, it is impractical to have a human instruct the robot during the learning. This problem can be more complicated when training the robot for multiple environments. Therefore, unsupervised learning is required. Further, because a goal is reached or an obstacle hit through the combination of many simple actions, it is impossible or very difficult to design a mathematical model to evaluate the robot's performance and therefore difficult to assign credit and blame in navigation. The learning system must evaluate the robot's navigation based on easily measurable characteristics of the system. For example, the travel time of a robot from start to goal can be easily and objectively measured and used by the learning system.

Although learning is an important feature of intelligent and autonomous robot systems, work beyond the conceptual stage is limited, especially for fuzzy behaviour-based reactive control system. Fikes, Hart, and Nilsson extended the STRIPS robot navigation system to allow it to learn from its failures[Fikes, et al 72]. Barto, Anderson, and Sutton attempted to solve non-linear robot navigation tasks using a two-layer neural network[Barto et al 82]. This simulation allowed the robot to learn an association between a landmark and the direction of travel that would lead it to the goal, which would provide positive reinforcement. Previous researchers have also applied genetic algorithms to robot navigation. Dorigo and Schnepf used this method to train simulated robots to avoid obstacles and follow moving targets[Dorigo91]. The genetic algorithm was used to determine when the robot should switch from one behaviour to another, as only one behaviour is active at a time. Thus the learning is at a fairly high and coarse level. The robot could not learn how to optimise their individual behaviours. Grefenstette, Ramsey, and Schultz's SAMUEL system takes a different approach; rather than optimise individual behaviours which are constructed using "decision rules", a genetic algorithm is used at the level of tactical plans comprising an entire set of decision rules for a given task[Grefenstette et al 90]. A GA has also been used in the optimisation of a mobile robot reactive control system by [Pearce92]. In the schema-based reactive control system, a set of parameters controlling motor-schemas are optimised to produce the different types of the robots for the purpose of safety, speed and directness. Training happens at the level of the combination of schemas, not for individual schemas. This is determined by the architecture of the reactive control system. The control output is produced by synthesising the results from all of the schemas at any time. Intended for learning a fuzzy behaviour-based reactive control system, a learning methodology different from the above methods is developed.

The learning methodology includes two parts of learning processes. It first focuses on the optimisation of individual components in the robot control system and then the overall control system. The learning methodology consists of several principles.

Individual behaviours are learned for its own functionality. Learning processes are generalised to obtain real useful results. Learning follows a simple-to-complex multistage course to enhance better exploration of solutions. Genetic algorithms are designed to facilitate efficient exploitation and exploration in the simple-to-complex multistage learning processes. Such a learning method can be more effective in building a real world mobile robot because it allows a good foundation to be built first and then the higher level of the control system through a general learning process. The methodology has been used to learn membership functions of the fuzzy behaviours and also the behaviour selection network for the MARCO's low level control layer in simulation. As a result, near-optimal fuzzy behaviours and a behaviour selection network have been automatically learnt. The results show it is possible to systematically learn a fuzzy behaviour-based reactive control system using the above learning methodology, therefore, greatly reduce the difficulties and efforts involved in the development of such systems.

Although GAs have been used in learning robot control systems, the use of GAs to automatically learn fuzzy behaviours and a behaviour selection network has not been reported in the literature. The developed methodology is mainly intended for the learning of MARCO's low level control layer. However, it is believed that such learning principles can also be applied to other systems because of the nature of their generalisation capabilities. The following sections introduce the learning of fuzzy behaviours and the behaviour selection network of MARCO's low level control layer using the learning methodology. The characteristics of the learning methodology will be exposed through the in-depth description of the learning processes.

5.4 Learning of Fuzzy Behaviours

5.4.1 Structure of fuzzy behaviour to be learnt

Let us first briefly review the structure of a fuzzy behaviour. A fuzzy behaviour in MARCO's low level control layer contains a fuzzy logic controller which is implemented with a set of fuzzy control rules. A triangular or half triangular forms of fuzzy membership functions are used in the antecedent part of a fuzzy rule. Fuzzy singleton representation is used in the output part of a fuzzy control rule. For example, Avoid Obstacle behaviour can have four rules to build up its function as follows:

if obs_left is CLOSE and
obs_right is not CLOSE then right_heading;
if obs_right is CLOSE and
obs_left is not CLOSE then left_heading;
if obs_front is CLOSE then speed_decrease;
if obs_front is CLOSE and
obs_right is as CLOSE as obs_left then left_heading.

Fig. 5-2 presents the membership functions of the fuzzy variables used in the rules. Note that, fuzzy set CLOSE for *obs_left* and *obs_right* is different from CLOSE for *obs_front*. Using a triangular and half triangular forms, every membership function of a fuzzy variable can be represented by the base values of its two extremes. For the rule,

if obs_right is CLOSE and
obs_left is not CLOSE then left_heading,

the low and high end values of the membership function of fuzzy variables, *obs_left* and *obs_right* are *min_dist* and *max_dist*. To select or tune fuzzy rules means the manual adjustment of the base values for all the fuzzy variables. It is a difficult task to tune fuzzy control rules one by one. Particularly, the range of sensor data and possible outcomes of actions taken by the robot can be unpredictable. Tuning one rule may

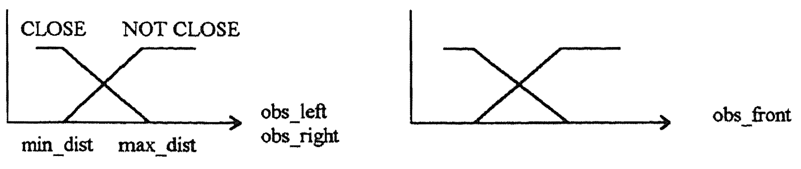


Fig. 5-2 Membership Functions of Fuzzy Variables for Avoid Obstacle

affect other rules. A fuzzy behaviour tuned in one environment may not work properly when the robot is placed in a different new environment. A set of fuzzy rules have to be designed, tested and redesigned many times. The use of GAs as an unsupervised learning method can greatly reduce the difficulties and efforts involved. To translate a fuzzy behaviour into genetic code, all the base values which characterise the membership functions of all fuzzy variables in the behaviour can be used. For example, fuzzy variable *obs_left* can be represented by the two ends of its membership functions, called *side_low, side_high*, into a pair of "gene"s in a genetic "chromosome". By

side_low	side_high	front_low	front_high	turn	speed

Fig. 5-3 Genetic Chromosome of Avoid Obstacle Behaviour

encoding all the fuzzy variables of a fuzzy behaviour into genes, the result is a complete genetic chromosome for Avoid Obstacle behaviour as shown in Fig. 5-3.

In this chromosome, genes **side_low** and **side_high** stand for *obs_left* and *obs_right*, genes **front_low** and **front_high** are for *obs_front*, **turn** represents the singleton values *left_heading* and *right_heading*, and **speed** controls the singleton value *decrease_speed*. The tuning of membership functions of a fuzzy behaviour, therefore, can be replaced by the search for an optimal set of genes through genetic algorithms.

5.4.2 Fuzzy Behaviour Learning Method

Since the performance of a fuzzy behaviour is determined by the values of membership functions of its fuzzy variables, genetic algorithms can be used to optimise these parameters using the navigational performance of the robot as a fitness metric. Fuzzy behaviour learning processes are designed using the developed learning methodology. In order to provide a systematic way of designing membership functions, the learning requires that fuzzy behaviours are all learnt from scratch. The learning does not rely on premeditated data. The only constraint is that every element, or gene has low and high limit values which cannot be exceeded. The range of the two limit values is wide enough to ensure that the learning is from almost zero knowledge. Individual behaviour is also learnt for its own functionality. Every fuzzy behaviour performs a different role in the low level controlling activities. To learn the behaviour is actually to build up its functionality for the role. To do this, the robot is placed in different types of environments for learning different roles. This is because MARCO's fuzzy behaviours are implemented based on the sphere of influence of environment features. Fuzzy behaviours are activated when their associated environment features are provided by the robot control system. To learn these feature-invoked functionalities, different and specific environment configurations must be presented for learning different behaviours. For example, Avoid Obstacle behaviour should be learnt in various types of environments scattered with obstacles. Follow Edge behaviour should be given different shapes of wall edges. Various goal configurations should be available for the learning of Reach Position behaviour. During the learning of one behaviour, other behaviours should be disabled and the robot should be solely controlled by the learning behaviour[†].

It is impractical to learn the fuzzy behaviours of the robot in real environments from scratch because the robot can easily be damaged. The cost of the resources is also much too great for such learning to be realised. Simulation is, instead, a very efficient approach to do the learning. Simulation allows the learning of the control system to occur by moving the robot thousands or millions of times without the presence of persons and the risk of damage. By appropriate design of the simulator and the learning system, the final learnt results can then be used as a base for further learning in the real robot. A simulated learning system has been developed for experiments in the learning of fuzzy behaviours and the behaviour selection network.

5.4.3 A Multistage Learning Course

For each behaviour, the learning follows a simple-to-complex multistage course for a complete learning process. Behaviours are learnt consecutively through three types of environments from simple to difficult. The learning progresses from a simple stage, to

[†] There is an exceptional case in this experiment. The learning of Avoid Obstacle behaviour needs Keep Moving to provide speed support because obstacle avoidance can only reduce and not increase speed. This is discussed later.

an intermediate and a final stage. These stages are defined by the degrees of difficulties of the environment configurations with respect to the behaviour's functionality. For some behaviours, they are determined by the varying clutter degrees of learning environments, where the clutter is defined by the percentage of the learning environment occupied by obstacles. For other behaviours, they are defined by the complexity of the environment features. Furthermore, the learning environments are randomised to provide a variety of configurations for a behaviour to interact with. Noise factors are also introduced into sensor data and the robot movement. The purpose of this randomised simple-to-complex learning process is to provide a gradual and general learning method to search for optimal solutions which can be applied in general circumstances. Three stages are currently selected in this learning process. They represent three typical types of simple, intermediate and complex environments with regard to a behaviour's function. At the simple stage, the learning environments provide sparse spaces or simple features for an initial population to begin with. After initial suitable solutions emerges, the population enters more demanding learning processes for better solutions. The learning in simple environments has many chances to find parameters that give a fast and safe behaviour. These good solutions in sparse or simple world are also more likely to be effective in denser or more difficult environments. At every stage, learning environments vary constantly but with the same degree of complexity. This varying property is especially important to produce general results because it presents a large number of situations to the learning process and helps to reduce the chances of local optimas. In general, a complex, denser and changing environment often requires more times for a converged result[Ram94]. The simple-to-complex multistage course provides a more efficient way to search better solutions[Qiu97a][Qiu97b].

5.4.4 Fuzzy Behaviour Learning Environments

Different types of learning environments are provided for the learning of different fuzzy behaviours. In this experiment, four fuzzy behaviours are used to test the learning methodology. These behaviours are Avoid Obstacle, Reach Position, Follow Edge and

Track Path. In the learning of Avoid Obstacle behaviour, Keep Moving behaviour is needed to provide speed support because Avoid Obstacle behaviour can only reduce speed and cannot increase speed. Keep Moving behaviour is easy to implement and therefore learning this behaviour is not necessary. However, because the learning of

Avoid Obstacle involves the interaction of two behaviours, their behaviour links also need to go through the learning process. This is done through the encoding of

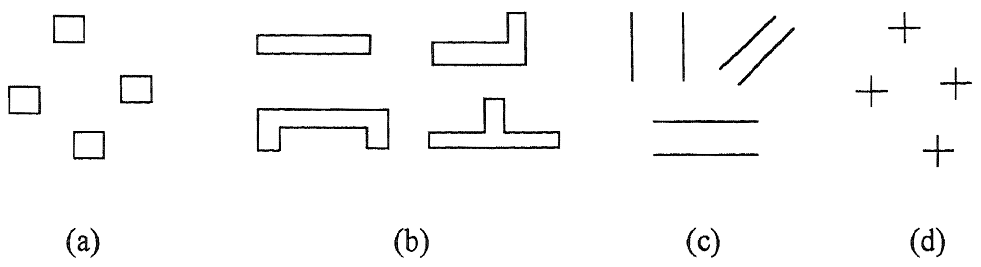


Fig. 5-4 Patterns of Learning Environments
(a) scattered obstacles in Avoid Obstacle learning environment;
(b) edge features for Follow Edge behaviours;
(c) paths for Track Path behaviour;
(d) goal configurations for Reach Position behaviours.

behaviour link factors into Avoid Obstacle chromosome. This is , in fact, a part of the behaviour selection network learning which will be described later. Fig. 5-4 presents example patterns of learning environments used in the learning of these fuzzy behaviours.

Fig. 5-5 shows some typical simulated worlds in three different stages, respectively, for the learning of Avoid Obstacle behaviour.

Note that in each row, 3 simulated worlds have same degree of complexity but different obstacle locations. In each column, while still randomly created, the complexity of the worlds is increased with 10% clutter differences. For the other behaviours, their example learning environments are shown in Fig. 5-6.

The generalisation of the learning environments is of great importance to the success of a learning process. The feature based configuration of the learning worlds provides the necessary environment conditions for the individual behaviour's functionality to be learnt.

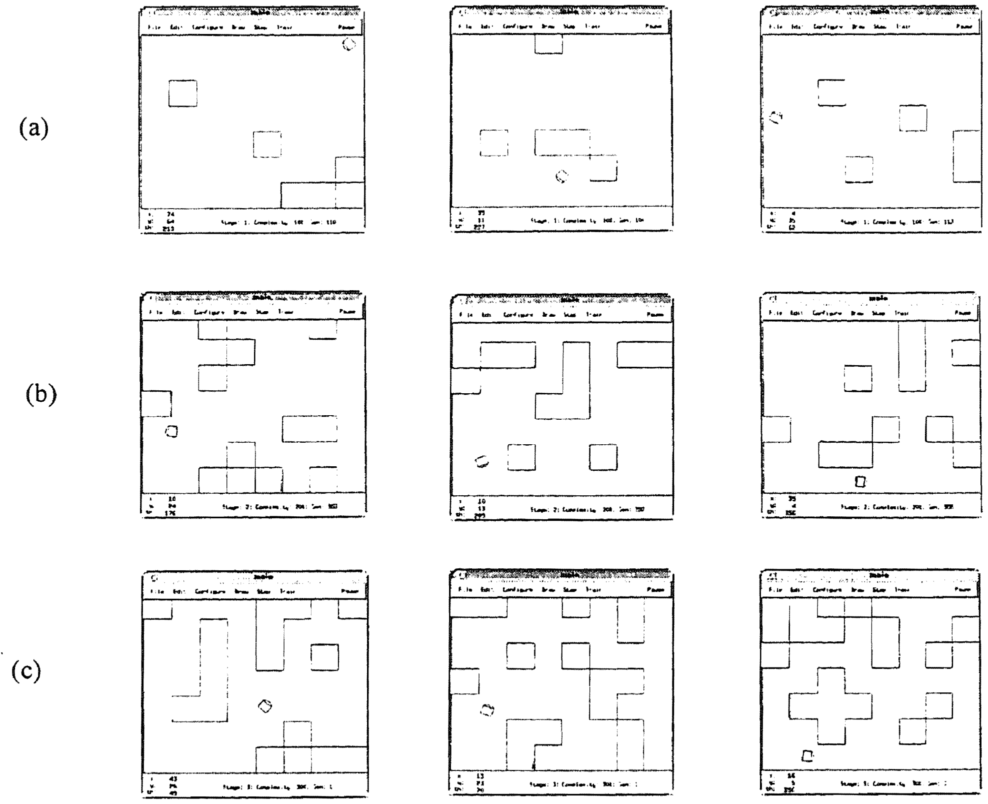


Fig. 5-5 Some Typical Simulated Worlds for Avoid Obstacle Behaviour
(a) 10% clutter worlds in stage 1;
(b) 20% clutter worlds in stage 2;
(c) 30% clutter worlds in stage 3.

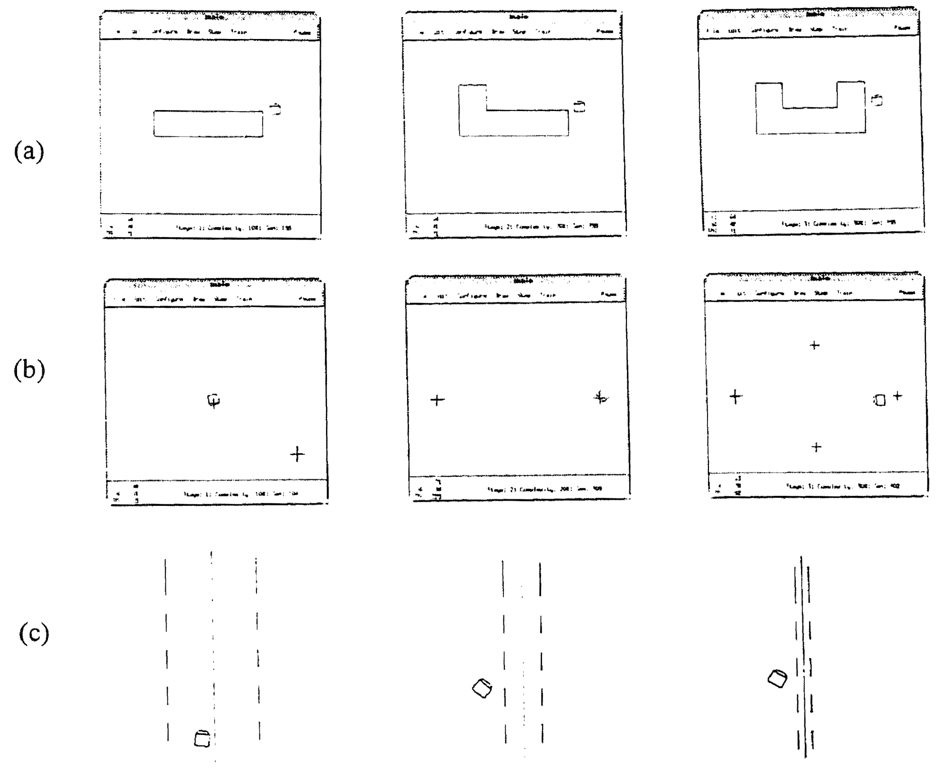


Fig. 5-6 Some example worlds for other behaviours in 3 stages:
(a) Follow Edge behaviour learning worlds;
(b) Reach Position behaviour learning worlds;
(c) Track Path behaviour learning worlds.

5.4.5 Behaviour Genetic Chromosomes

The search for optimal fuzzy behaviours is implemented via genetic algorithms. A population representing each type of fuzzy behaviour is maintained and manipulated through genetic operations. Individual members of the population are represented as strings of floating point values. This is different from traditional GA coding using a binary string or character string[Goldberg89]. Because the number of the fuzzy control rules and the number of fuzzy variables for a behaviour is relatively small compared to other GA applications, a floating point coding is more direct and efficient for the reproduction processing. For each behaviour, the low and high end values of all membership functions of all fuzzy variables are taken as position-dependent "gene"s and encoded into a "chromosome" representing a behaviour. Fig. 5-7 are the structures of four fuzzy behaviour chromosomes. Note that Follow Edge behaviour and

side_low	side_high	front_low	front_high	turn	speed

(a) Avoid Obstacle Behaviour

angle_low	angle_high	obs_low	obs_high	near_low	near_high	align_low	align_high	width_low	width_high

(b) Follow Edge behaviour

angle_low	angle_high	obs_low	obs_high	near_low	near_high	width_low	width_high

(c) Track Path behaviour

med_low	med_high	small_low	small_high	big_low	big_high	speed_low	speed_high

(d) Reach Position behaviour

Fig. 5-7 Structures of Fuzzy Behaviour Chromosomes

Track Path behaviour chromosome have very similar structures. The only difference is that Track Path behaviour is a task-oriented behaviour and needs to measure the completion of the "follow path" task, while Follow Edge behaviour just senses and follows the edge at its vicinity acting like a reactive behaviour. They have the same set of fuzzy control rules despite their different sensor inputs.

A member of the population can easily lend itself to the task of controlling the robot after its genes are extracted and used as parameters for the membership functions of a behaviour.

5.4.6 Design of Genetic Operators

A new population during the learning is produced by the combination of genetic reproduction operations. This learning algorithm uses five genetic operators. They are random initialisation, a crossover operator, a mutation operator, an average operator, and a reproduction operator. The design of the operators are mostly based on the existing technologies[Goldberg89][Davis91][Janikow91][Whitley89], and have been modified to meet the requirement of the learning methodology. The design of genetic algorithms are equally applicable to the learning of the behaviour selection network. The reason for the inclusion of the detailed design of the genetic algorithm in this section is that this learning methodology was first used for learning fuzzy behaviours. Three main operators, crossover, mutation and average are each assigned with a probability level which determines their chances of being selected for the current reproduction.

5.4.6.1 Random Initialisation

This operator produces an initial population from which the genetic evolution process starts. One of the principles of the learning algorithm is that the fuzzy behaviours are learnt from scratch. In the initialisation, the value of a gene of a chromosome is randomly created between its lowest and highest limit values. The limit values are chosen based on the role of the gene in the chromosome and possible meaningful extremes. For instance, the genes **angle_low** and **angle_high** for Follow Edge chromosome in Fig. 5-7 both have 0 as the lowest and 2π as the highest limit of the value. The genes **obs_low** and **obs_high** both have the range from 0 to 2m. These wide ranges ensure the learning commences from almost zero knowledge. The random initialisation operator creates the initial population by producing the genes within these wide ranges. The initial population is, therefore, a less constrained random result. This operator can also be used in local optimisation where a small range of the possible variation to provided data is set as the limits.

5.4.6.2 Crossover

A crossover operator is used to bring in new members of populations. In this learning algorithm, an uniform crossover operator is designed for the purpose. The genes of two parents may be exchanged at the positions where they differ, under the control of an exchange probability. In a bit string representation, a two-point crossover is often used. After two random positions in a chromosome are selected, the gene between two points are exchanged completely[Goldberg89][Janikow91]. In a floating point representation, such exchanges often have huge impacts on a chromosome. A variation of this form of the operation have been used to avoid this problem [Pearce92] [Davis91]. In this design, instead of the complete exchange between points, exchanges take place at positions where the genes differ and a probability test is satisfied. The exchange probability is selected at an appropriate level which would not have huge impacts on chromosomes while still allowing sufficient exchanges to enable wide exploration. For example, to crossover two chromosomes

0.500000, 0.300000, 0.700000, 0.400000 and 0.300000, 0.800000, 0.700000, 0.100000

using the method, 0.2 is selected as the exchange level. The genes at the three positions are exchangeable. Suppose that the randomly created exchange probabilities are 0.1, 0.3 and 0.8 respectively for the three positions. Because 0.1 is small than the level 0.2, the pair of the genes at the first position are exchanged while the genes at the two other positions remain unchanged. The results of the operation are two new chromosomes:

0.300000, 0.300000, 0.700000, 0.400000 and 0.500000, 0.800000, 0.700000, 0.100000 .

5.4.6.3 Mutation

The design of the mutation operator has two purposes in this learning algorithm. First, the mutation operator produces a new member for the next population by operating on a selected parent. Second, the operator is designed to help support the simple-to-complex multistage learning principle. Learning environments change from simple to difficult as the learning progresses. Initial wide and deep exploration can be more effective in simple environments and a local search can be more effective with difficult learning environments. Some good solutions can be learnt from the simple environments but cannot be obtained from difficult environments and vice-versa. This has been observed by other researchers[Pearce92] and also in this experiment. Because the learning is started from almost zero knowledge, there is not much benefit in finely tuning a gene of a chromosome at the early stage of populations, which will consist largely of rough forms of solutions. Instead, the whole range of search, from shallow to deep, should be applied to the genes of chromosomes to have a deep "scrambling" for better ones to emerge in simple environments. As the population grows and better solutions surface, this search is gradually localised by finely tuning the genes of a chromosome in difficult environments for even better solutions. To support such a search method, an adaptive uniform mutation operator has been designed. The mutation is applied to the genes of a chromosome only when a probabilistic rule permits. The probability control level is also appropriately selected. The effects of the mutation operators on a gene are determined by two parts. One is a random variation from the current gene value within the gene value limits. The other part controls the level of this variation applied to the gene and changes adaptively as the generations progress. The formula of the mutation is adopted from Michalewicz[Michalewicz92] and represented as follows:

$$\text{new\_gene} = \begin{cases} \text{old\_gene} + \Delta(t, \text{UB} - \text{old\_gene}); & \text{if a random boolean test is false} \\ \text{old\_gene} + \Delta(t, \text{old\_gene} - \text{LB}); & \text{if a random boolean test is true} \end{cases} \qquad (1)$$

where,

$\Delta(t, a) = a*(1 - p^{(1-t/T)^r})$, returns a value in the range [0, a];

p — random probability value [0, 1];
T — the maximum generation;
t — current generation;
r — exponent controlling the speed of probability distribution change.
UB, LB — gene's upper and lower boundary values.

5.4.6.4 Average Operator

An average operator is also used, operating on two parents in order to obtain more ways of exploration for better children, while a close link to parents is still maintained.

5.4.6.5 Reproduction Operator

In reproduction, parents are selected randomly but with a bias towards the fittest individuals. The best individuals are more likely, but not guaranteed, to be selected for reproduction. Newly produced members compete with the old population and the weakest individuals are removed by a razor cut method in order to keep the new population size constant.

5.4.7 Evaluation Functions

How a solution is evaluated often determines the success of a genetic algorithm. In learning mobile robot control behaviours, it is difficult to find a mathematical model to evaluate the performance of the mobile robot reactive control system because of the lack of the precise predictions in the robot movement, environment uncertainty and sensor noises. Extensive research for such an evalution function diminishes the advantage of using genetic algorithms. Instead, some directly measurable performance indexes can be used to evaluate the fitness of a solution. In this experiment, time, distance, range and collision measurement are chosen as the indexes because they can be readily retrieved. The learning of each behaviour is evaluated separately from the others because the functionalities to be learnt are different. For example, Avoid Obstacle behaviour is required to be able to control the robot to avoid collisions with the environments, to move the robot fast and to move the robot close to the objects in the various environments. With these requirements, the behaviour is checked for the time steps taken by the robot, the collision of the robot with the environment and the minimum range of the robot to the environment after the robot travels a certain distance. Time steps are also used in measuring the collision penalty. The longer the robot has survived before a collision happens, the less penalty it receives. A squared root time step function is used in order to limit the effect of the time steps. These observable data sets are used to form the following evaluation function:

$$\text{eval\_value} = \text{time\_weight} * \text{time\_steps} + \text{distance\_weight}*\text{distance} + \text{range\_weight}*\text{minimum\_range} + \text{collision\_weight} * \text{collision}/ \text{time\_steps}^{\frac{1}{2}}; \quad (2)$$

The raw evaluation value is directly used to evaluate the performance of a behaviour. The fittest individual has the minimum evaluation value. Similarly, the evaluation functions for Follow Edge, Track Path and Reach Position behaviours are given as (3) (4) (5):

eval_value = time_weight * time_steps + range_weight*sum_of_minimum_range + collision_weight * collision/ time_steps^{-2} ; (3)

eval_value = time_weight * time_steps + distance_weight*distance + angle_weight*sum_of_angle_change ; (4)

eval_value = time_weight * time_steps + distance_weight*distance + goal_weight*goal_left + achieve_weight * sum_of_minimum_distance_to_goals; (5)

With these evaluation criteria, Follow Edge behaviour is learnt in order to control the robot to follow a wall edge fast, smoothly, closely and without collision. Track Path behaviour is learnt in order to quickly move the robot into a designated path and follow it accurately, fast, smoothly to the end. Reach Position behaviour is learnt in order to move the robot fast, directly and precisely to reach goal positions.

There are three stages in a complete learning process. When the population progresses from one learning stage to the next more difficult stage, its members are first re-evaluated in the new environment and then start the new stage of learning. This process is necessary to ensure all members of a population are evaluated in the same new environment with an equal opportunity to start the new competition. Thus, a smooth stage transition is completed in the learning process.

5.4.8 Control Parameters and Learning Algorithm

The implementation of a genetic algorithm requires the specification of a number of parameters that govern the effectiveness of the algorithm, such as the probabilities of crossover, mutation and reproduction. The choice of these parameters is heuristic, and is based on guidelines[Goldberg89][Davis91][Janikow91] and has been empirically studied in this experiment. These guidelines include maintaining a diverse population to prevent premature convergence and a balance between exploration and exploitation. In addition, the simulation design factors are also considered, such as the speed of the simulated robot, the time needed to complete a learning process and the exploitation of the learning environments to support the exploration of the population.

For the uniform crossover operator, a too low exchange probability can prevent a wide exploration. A too high one can exert too much impacts on chromosomes. A value of 0.2 was found to be the appropriate level. There are two parameters to control the uniform mutation operator. One is the probability of mutation, the other is the exponent to determine the speed of the adaptation. For the probability, a too high value reduces the algorithm to a random walk, while a too low value defeats the purpose of the operator. A value of 0.2 was selected as an optimal level. The exponent determines the speed of the adaptation as the generation progresses. A value of 3.0 was chosen as the appropriate exponent after trials. The selection probability of crossover, mutation and average operators were 0.2, 0.2, and 0.1 respectively and were not changed in the learning process. After reproduction, a population of 50 members was maintained. The new population is then explored with the uniform mutation operator, average operator and exploited with the uniform crossover operator. These specially designed operators help to maintain a diverse and yet converged search.

A rank-based selection was used to select parents with a 1.5 bias towards the fittest individuals in the population. 5 new members are produced in every generation and their ranking is determined with 50 members of the current population. Afterwards, the weakest 5 members are removed to form the new population and keep the population constant. The number of the generation for the genetic algorithm was set to 1000 to allow adequate time for the multistage learning. The first stage ranges from 0 to the 299th generation. The second stage is from the 300th to the 599th and the final stage starts from the 600th and ends at the generation 1000.

The high level structure of the learning algorithm is presented in Fig. 5-8.

```
Randomize_initial_population();
for(i=0; i<GENERATION; I++)
{
  for(j=0; j< NUM_OFFSPRING; j++)
  {
     create_learning_environment((type_of_stage);
     select_parent_for_reproduction();
     select_operation:
       CROSSOVER, MUTATION, AVERAGE.

     while(! end_of_a_training_circle)
       move_robot();
     get_performance_indexes();
     evaluate_fitness();
     reproduction();
   }
 }
```

Fig. 5-8 High Level Structure of the Learning Algorithm

5.4.9 Simulation Results Analysis

The learning of four example fuzzy behaviours were carried out in simulation and the results were recorded for analysis. The effectiveness of the learning algorithm was checked from several aspects including the variations of fuzzy membership functions, genetic algorithm convergence and visualisation of physical movement.

5.4.9.1 Learnt Membership Functions

Table 5-1 The first 5 members of the initial population at stage 1 for Follow Edge Behaviour

angle_low	angle_high	obs_low	obs_high	near_low	near_high	align_low	align_high	width_low	width_high
0.111983	1.0959	430.917736	685.433138	144.868537	550.708239	0.22263	1.507288	252.2124	133.9624
0.742808	0.214144	216.451728	861.447839	167.501085	560.670737	0.192419	0.404823	434.4416	101.6808
0.153649	1.261797	301.651998	383.932265	152.174064	830.036857	0.099227	0.365512	157.4104	356.4492
0.413723	0.370342	250.862662	1183.979189	94.092752	595.493343	0.419816	1.125918	268.5932	136.092
0.784763	0.43788	369.724193	1642.348454	134.317972	499.257881	0.323119	1.247509	326.8936	2.8628

Table 5-2 The first 5 members of the initial population at stage 2 for Follow Edge Behaviour

angle_low	angle_high	obs_low	obs_high	near_low	near_high	align_low	align_high	width_low	width_high
0.164199	0.717262	203.269094	540.403546	188.526037	550.708239	0.22263	0.923196	161.03	255.8336
0.164199	0.906581	203.269094	685.433138	167.422071	550.708239	0.22263	0.624126	145.8328	175.244
0.164199	0.717262	203.269094	685.433138	188.526037	550.708239	0.249367	0.923196	161.03	255.8336
0.153649	1.150595	203.269094	685.433138	146.318104	550.708239	0.22263	0.325057	130.6356	133.9624
0.165274	0.933928	203.269094	609.912373	188.526037	501.800023	0.249367	0.723955	161.03	194.898

Table 5-3 The first 5 members of the initial population at stage 3 for Follow Edge Behaviour

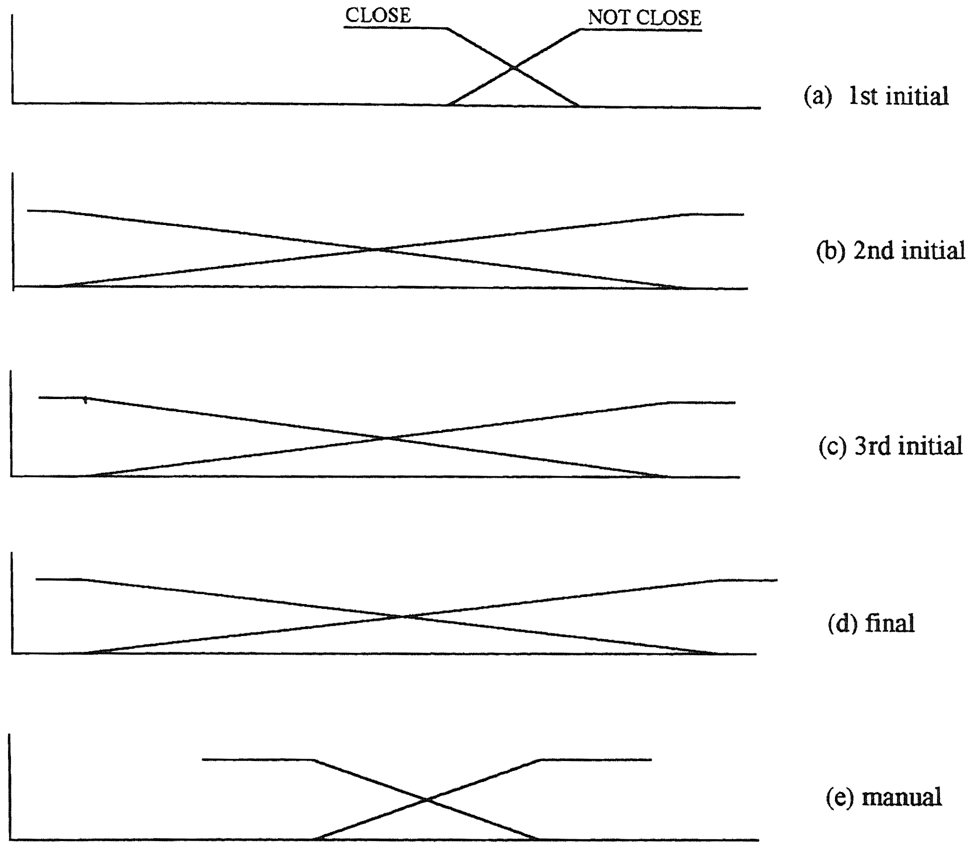
angle_low	angle_high	obs_low	obs_high	near_low	near_high	align_low	align_high	width_low	width_high
0.164199	0.813576	203.269094	685.433138	188.526037	550.708239	0.22263	0.923196	161.03	225.3656
0.162099	1.493876	209.815193	685.433138	188.526037	316.830616	0.22263	0.624127	161.03	225.366
0.164199	0.717262	203.269094	685.433138	188.526037	550.708239	0.22263	0.923196	161.03	255.8336
0.164199	0.717262	203.269094	685.433138	188.526037	550.708239	0.22263	0.923196	161.03	255.8336
0.162099	1.493876	209.815193	685.433138	188.526037	316.830616	0.22263	0.624127	158.3008	225.366

Table 5-4 The first 5 members of the final for Follow Edge Behaviour

angle_low	angle_high	obs_low	obs_high	near_low	near_high	align_low	align_high	width_low	width_high
0.16081	0.813576	203.269094	685.433138	188.526037	550.708239	0.219495	0.923196	161.03	225.3656
0.164199	0.813576	203.269094	685.433138	188.526037	550.708239	0.219495	0.773661	161.03	225.3656
0.164199	0.813576	203.269094	685.433138	188.526037	550.708239	0.219495	0.923196	161.03	225.3656
0.164199	0.717262	203.269094	685.433138	188.526037	593.868019	0.219495	0.773661	161.03	225.366
0.164199	0.813576	203.271036	685.433138	188.526037	593.869123	0.219495	0.923196	161.03	225.3656

The membership functions of the behaviours were learnt as intended. The chromosomes representing the membership functions were changed from initial random values to converged values within a small range after the learning process stops. Tables 5-1, 5-2 , 5-3 and 5-4 give the comparisons of the first 5 chromosomes of Follow Edge behaviour from the initial population at stage 1, 2, 3 and the final population respectively. In Table 1, the five chromosomes are very different because they were produced randomly within the limits. After first stage of learning the 5 members of Table 5-2 are much more similar than of Table 5-1. As the learning progresses, the similarity of the 5 members increases further as shown in Table 5-3 and Table 5-4. The level of the change on the membership functions of individual fuzzy variables can vary greatly because of the different initial values and different roles in a fuzzy control rule. The most significant changes occur during the first stage of the learning because wide and deep exploration by the crossover and mutation operations make the suitable solutions surface quickly. In the later stages, these changes become smaller because wide and deep explorations were gradually replaced by locally tuning of the genes of the chromosomes.

Fig. 5- 9 graphically compares the change of membership function for one fuzzy variable of Avoid Obstacle behaviour. It shows the membership functions of obs_left from the best individual of initial population at stage 1, 2 and 3, the final population and manually tuned behaviour. The similar trend of changes to the above 5 members of Follow Edge behaviour can be observed. Interestingly, it can also be seen that the differences between manually tuned values and learnt values are significant. In manually tuning of the fuzzy behaviour, some of the learnt values will be less likely to be considered as they seem to be unsuitable. In fact, they turn out to be optimal control values when combined with other ones. This is because manually tuning of fuzzy control rules, one by one, is difficult. There are many factors to influence the control output of a fuzzy controller. The learning allows the optimisation of the entire set of fuzzy control rules for a behaviour. Good overall performance of a fuzzy behaviour is learnt instead of finely tuned individual rules. The difficulties and efforts are greatly reduced.



	side_low	side_high
1st initial	577.311401	753.700301
2nd initial	38.611232	909.932132
3rd initial	105.507143	874.094843
final	96.397641	937.302441
manual	400	700

Fig. 5-9 Membership functions of fuzzy variable *obs_left* of Avoid Obstacle behaviour during the learning process:
(a) the initial best at stage 1; (b) the initial best at stage 2; (c) the initial best at stage 3; (d) the best of final population; (e) one of manually tuned behaviour.

5.4.9.2 Genetic Algorithm Convergence Evaluation

The performance of the learning algorithms is also analysed for their convergence at every stage of the learning. The best, the worst and average evaluation value of each generation have been recorded for analysis. For each example fuzzy behaviour, the fitness values of the population converges to a set of values within a maximum of 2%

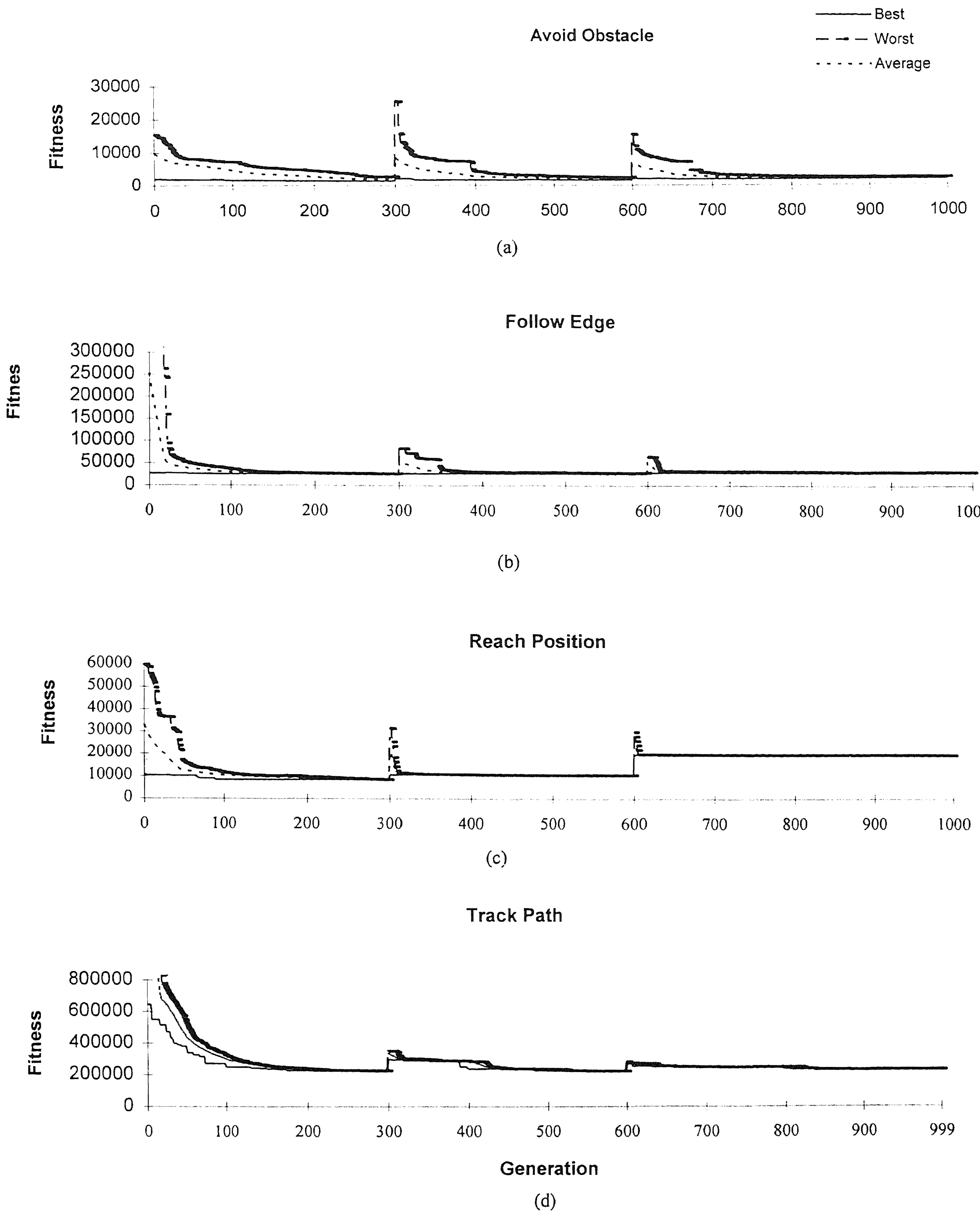


Fig. 5-10 Generation vs. Fitness charts of learning processes for fuzzy behaviours: (a) Avoid Obstacle; (b) Follow Edge; (c) Reach Position ; (d) Track Path.

variation after the learning process terminates. The small variation is mainly caused by the random noise introduced in the sensor and the robot movement and will exist even for the same set of membership functions. At the ends of earlier learning stages, the fitness values were also seen to have converged but with higher variation ranges. The transition between learning stages does have impacts on the populations. This is because the learning environment becomes more difficult and presents more chances for failure. This can be seen in evaluation chart Fig. 5-10. At the start of a new stage, there are big increases in the worst evaluation values. As the learning continues, this phenomenon gradually reduces towards the end of the stage and then reappears when the learning process again enters a more difficult stage. Although the worst evaluation value of a later stage may exceed that of a previous stage, the average value of the later stage is smaller than that of the previous one. This indicates that the overall performance of the population improves over the whole learning process.

5.4.9.3 Effects of Multistage Learning

A simple-to-complex multistage learning course is one of the main principles of this learning methodology. The effectiveness of the multistage learning is evaluated by comparing it to a non multistage learning process. In the non multistage learning, the most complex environments used in multistage learning are employed in the whole learning process. Other factors, such as the genetic algorithm control parameters, performance evaluation functions and randomisation of the learning process all remain unchanged. Some of the genetic algorithm control parameters are listed as follows:

genetic operators:
  uniform crossover: 0.2 exchange prob., 0.2 selection prob.;
  uniform mutation: 0.2 mutation prob., 0.2 selection prob., 3.0 speed exponent;
  average: 0.1 selection prob.
population size: 50
new population size: 5
generation: 1000

selection method: 1.5 biased rank-based selection.

The comparisons were made mainly on the convergence of the genes. The initial populations for the two learning processes are the same as given in Appendix B.

Starting from the same initial population, the two learning approaches produced very different learning results as indicated in Table 5-5 and Table 5-6. Table 5-5 shows the first 10 members of populations at the ends of stage 1, 2, and 3 as well as the genes' evolution measurement for the multistage learning approach. Table 5-6 gives the first 10 members of population at generation 299, 599 and 1000 and the measurements of the variations of their gene values for the non multistage learning. We can see that in the multistage learning, the gene values evolved faster and converged to within the maximum 5% variation range, much smaller than that of non multistage learning, 26%. At the end of the first learning stage, the population still had a more diverse combinations of genes than that of the single stage learning. This is because simple environments provide more chances for initial deep and wider exploration of population for possible better solutions. In contrary, the complex environment did not help to yield variety of initial solutions after the equivalent 300 generations for the non multistage learning. The population lacks diversity, which is needed for an efficient learning process. A complex environment prevents the initial suitable solutions from emerging quickly. With the multistage learning process, the population was able to continue to efficiently evolve, based on the diverse initial solutions produced in the earlier stages, and finally settled on a small variation range. For the non multistage learning, the evolution process was slow and the final genes, though converged, had much less stable structure than that of the multistage learning. Some of the gene values are obviously unsuitable for practical applications, such as the speed values, which are too high for reversing. The multistage learning process was also much faster than the non multistage learning process. For example, in learning Avoid Obstacle behaviour, the non-multistage learning took nearly 3 times as

Table 5-5 The first 10 members of the end population at 3 stages for multistage learning

Stage 1	side_low	side_high	front_low	front_high	turn	speed
10 members	38.611232	909.932132	808.111097	833.125097	1.906284	-51.136032
	108.978043	869.801743	812.622531	837.029931	1.940549	-45.594978
	121.558446	962.463246	808.111097	831.911897	0.960112	-40.284876
	105.507143	874.094843	812.778144	837.184644	1.906284	-41.348488
	121.558446	962.463246	808.111097	833.125097	1.906284	-51.136032
	48.681906	943.620606	792.46435	816.26515	1.903953	-43.845314
	55.971152	950.909852	792.46435	808.42945	1.834982	-41.845314
	121.558446	890.146146	808.111097	831.911897	1.906284	-40.284876
	38.611232	933.549932	792.46435	816.26515	1.903953	-43.845314
	22.502554	863.407354	808.111097	831.911897	0.960112	-51.136032
Average Deviation	48%	4%	1%	1%	18%	8%

Stage 2	side_low	side_high	front_low	front_high	turn	speed
10 members	105.507143	874.094843	812.778144	837.184644	1.906284	-41.348488
	108.978043	953.664943	808.111097	832.517597	2.146908	-43.747959
	105.507143	886.234643	812.778144	837.184644	1.906284	-41.348488
	88.764799	909.039199	808.111097	832.518497	1.906284	-45.710454
	96.397641	777.140241	817.133966	842.147966	2.146908	-47.441996
	55.810886	938.546486	808.111097	823.229297	1.906284	-43.747959
	88.764799	956.686399	800.287723	824.088523	1.905118	-42.065095
	82.474597	940.798597	817.133966	841.844066	1.906284	-47.441996
	96.397641	991.336341	792.46435	816.26515	1.903953	-43.845314
	72.059188	960.896188	802.621247	826.724747	1.905118	-42.596901
Average Deviation	14%	5%	1%	1%	4%	4%

Stage 3	side_low	side_high	front_low	front_high	turn	speed
10 members	96.397641	937.302441	808.111097	833.125097	1.906284	-51.136032
	96.397641	896.875952	808.111097	833.125097	1.906284	-43.747959
	96.397641	881.685741	808.111097	833.125097	1.906284	-51.136032
	96.397641	881.685741	808.111097	833.125097	1.906284	-51.136032
	96.397641	937.302441	808.111097	833.125097	1.906284	-51.136032
	96.397641	937.302441	808.111097	833.125097	1.906284	-51.136032
	96.397641	937.302441	808.111097	833.125097	1.906284	-43.747959
	96.397641	937.302441	808.111097	833.125097	1.906284	-51.136032
	96.397641	937.302441	808.111097	833.125097	1.906284	-51.136032
	96.397641	937.302441	808.111097	833.125097	1.906284	-51.136032
Average Deviation	0%	2%	0%	0%	0%	5%

long as the multistage learning did to finish 1000 generations. Much of time was spent in the early stage of learning in the complex environments and the many initial random solutions caused extremely slow movement of the robot and resulted in a very inefficient learning process. The multistage learning was able to avoid such problems

Table 5-6 The first 10 members of the population 299, 599, 1000 for non multistage learning

299th	side_low	side_high	front_low	front_high	turn	speed
10 members	500.90647	1393.7134	395.37882	467.99442	1.87938	-64.58798
	423.42118	574.75978	395.37882	461.03982	1.87938	-98.90892
	403.75825	764.50435	395.37882	458.97012	1.99936	-94.15747
	217.03148	571.67528	395.37882	467.99442	1.99936	-57.63856
	403.75825	731.83945	395.37882	467.99442	1.99936	-94.15747
	214.92408	677.78028	395.37882	458.97012	1.99936	-57.63856
	403.75825	1065.2021	395.37882	458.97012	1.99936	-97.05541
	217.03148	571.67528	395.37882	467.99442	1.99936	-57.63856
	403.75825	1065.2021	395.37882	458.97012	1.82697	-94.15747
	403.75825	555.09685	395.37882	458.97012	1.82697	-94.15747
Average Deviation	24%	28%	0%	1%	4%	21%

599th	side_low	side_high	front_low	front_high	turn	speed
10 members	500.90647	1393.7134	395.37882	467.99442	1.82697	-94.15747
	403.75825	1065.2021	366.39735	429.98865	1.87938	-95.53073
	403.75825	643.46815	395.37882	458.97012	1.87938	-94.15747
	500.90647	1393.7134	381.01173	453.62733	1.82697	-94.46457
	403.75825	555.09685	395.37882	446.36562	1.95165	-95.53073
	500.90647	1393.7134	395.37882	467.99442	1.85427	-94.15747
	403.75825	758.40205	395.37882	458.97012	1.82697	-90.00019
	403.75825	700.93525	395.37882	458.97012	1.85427	-92.07883
	403.75825	555.09685	395.37882	461.22612	1.87938	-94.47566
	403.75825	555.09685	395.37882	458.97012	1.99936	-94.15747
Average Deviation	9%	36%	2%	2%	2%	1%

1000th	side_low	side_high	front_low	front_high	turn	speed
10 members	500.90647	1393.7134	395.37882	467.99442	1.82697	-94.15747
	403.75825	643.46815	395.37882	458.97012	1.87938	-94.15747
	403.75825	758.40205	395.37882	458.97012	1.82697	-90.00019
	310.39487	563.38607	395.37882	463.48212	1.93937	-75.89801
	403.75825	1065.2021	395.37882	458.97012	1.82697	-75.14691
	403.75825	555.09685	395.37882	458.97012	1.87938	-94.15747
	403.75825	643.46815	395.37882	461.22612	1.91317	-94.15747
	403.75825	731.83945	366.39735	417.38415	1.87938	-95.53073
	403.75825	555.09685	395.37882	458.97012	1.87938	-94.15747
	403.75825	555.09685	395.37882	458.97012	1.92545	-94.15747
Average Deviation	5%	26%	1%	2%	2%	7%

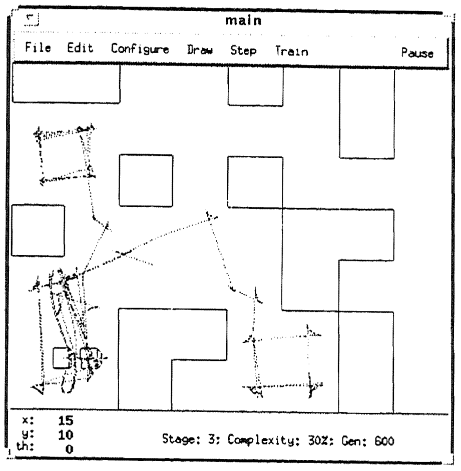
by exploiting the initial simple environments and this speeded up the whole learning processes.

5.4.9.4 Visualisation of Learning Results

In the learning of our example fuzzy behaviours, the visualisation provided a qualitative feel for the success of the learning algorithms. The simulation system is able to reproduce the traces of the robot with learnt chromosomes. This section discusses some of the typical traces of the robot in the learning of the example fuzzy behaviours as the last evaluation point.

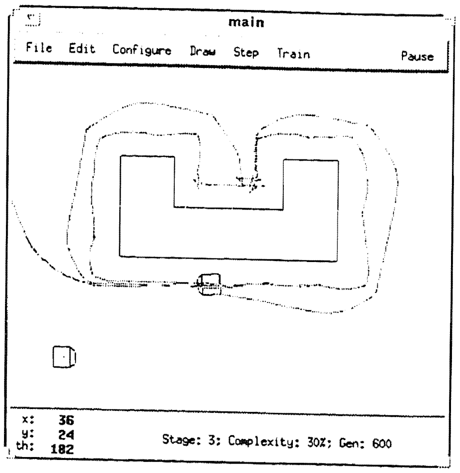
Fig. 5-11 and Fig. 5-12 shows the snapshots of the traces of the simulated robot in order to indicate the navigational abilities of different fuzzy behaviours in the learning processes. Fig.5-11(a) shows how the best Avoid Obstacle behaviour of the initial population, the best of the final population and the manually tuned behaviour controlled the robot movement in a 30% clutter environment. The courses of the robot were displayed as dot curves. The course leading to the low right part of the environment was produced by the best of the final population. The course leading to the upper left part of the environment was created by the manually tuned one. The control by the best of the initial population results in the course around the starting area. Table 5-7 shows their measured performance indexes. As seen in Fig. 5-11a, the robot controlled by the best of the final population travelled a longer distance, moved faster, straighter and moved closer to the environment than the others. Although the best of the initial population was able to control the robot to avoid collision with the environment, the robot moved very hesitantly and slowly. After the learning process, its final counterpart exhibited much more robust abilities and performed better than the manually tuned one.

Fig. 5-11b is the comparison of Follow Edge behaviours of the initial worst, the final worst and the manually tuned. The robot was asked to follow a concave-shaped wall edge, which is very difficult without navigation planning. In the development of this



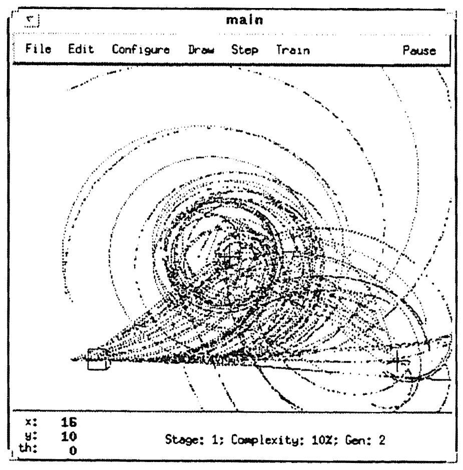
```
x: 15
y: 10
th: 0    Stage: 3; Complexity: 30%; Gen: 600
```

(a)



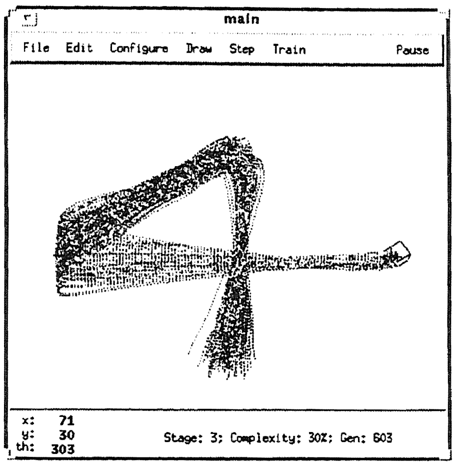
```
x: 36
y: 24
th: 182    Stage: 3; Complexity: 30%; Gen: 600
```

(b)



```
x: 16
y: 10
th: 0    Stage: 1; Complexity: 10%; Gen: 2
```

(c)



```
x: 71
y: 30
th: 303    Stage: 3; Complexity: 30%; Gen: 603
```

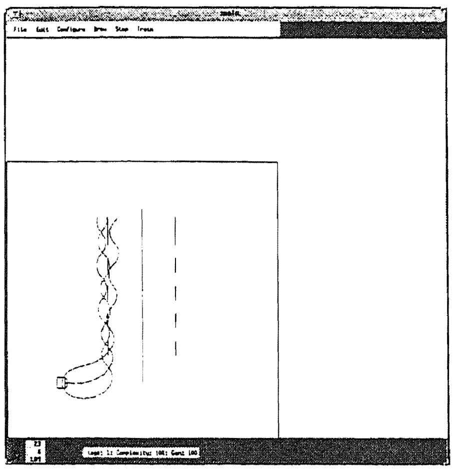
(d)

Fig. 5-11 Snapshots of the course of the robots controlled by four fuzzy behaviours:
(a) the initial best, the final best and the manually tuned Avoid Obstacle;
(b) the initial worst, the final worst and the manually tuned Follow;
(c) the initial 50 Reach Position behaviours for two goals;
(d) the final 50 Reach Position behaviours for four goals.

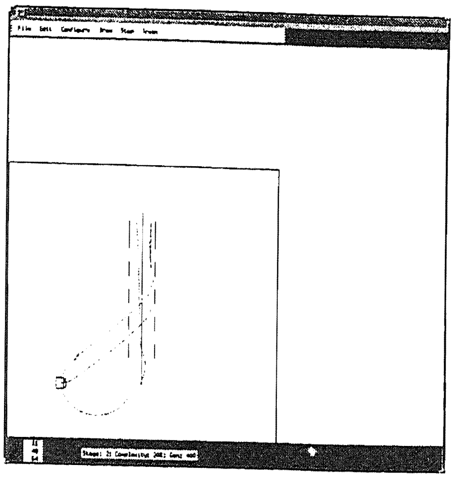
behaviour, many hours were spent in manually tuning the behaviour and the tuned behaviour still did not function satisfactory before the learning was introduced. A more satisfactory result was obtained for Follow Edge behaviour through the learning process. In Fig. 5-11b, three courses were produced by the

	time steps	min. dist.	collisions
final best	1163	99.4	0
initial best	1830	1207.26	0
manual	1401	489.81	0

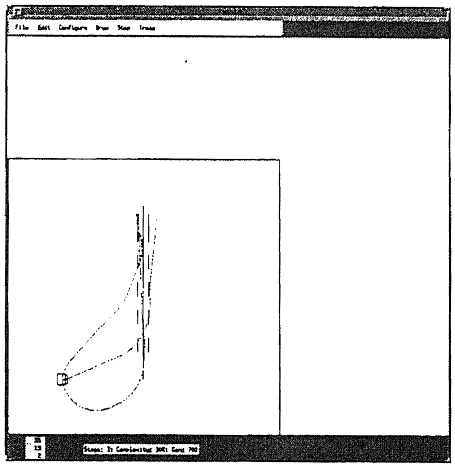
Table 5-7 Performance indexes of three Avoid Obstacle behaviours



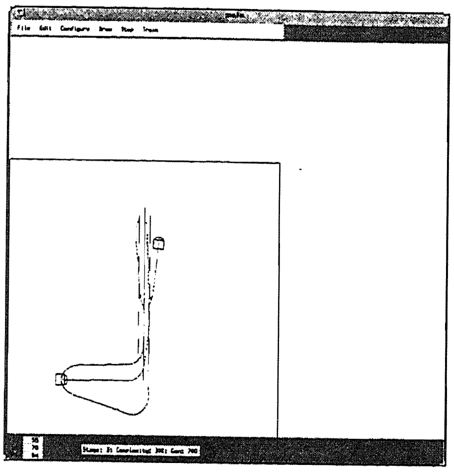
1



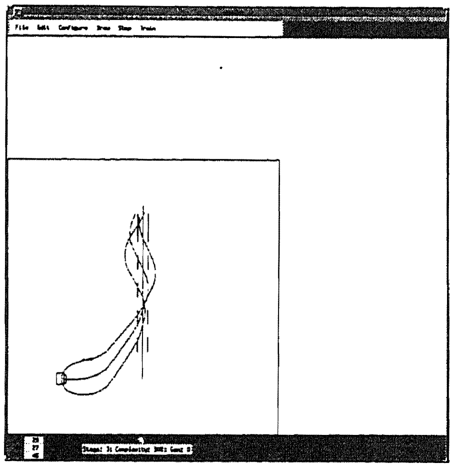
2



3



f



m

Fig. 5-12 Snapshots of the course of the robots controlled by Track Path
1. the initial average at stage 1;
2. the initial average at stage 2;
3. the initial average at stage 3;
f. the final average;
m. the manually tuned.

robot controlled by the three selected behaviours. The inner-most course was created under the control of the final worst. It shows that the robot followed the edge smoothly and closely, turned at the inside and outside corners accordingly and moved fast, though it was the weakest behaviour in the final population. In contrast, the course next to it was created under the control of the manually tuned behaviour.

Although the robot followed straight edges reasonably well, it did not exhibit robust abilities in turning at the corners and took more time to finish the task. The course leading out of the display area was the result of the initial worst which showed the intention of following the edge, but was not able to function properly.

Fig. 5-11c shows all of the courses of the robots controlled by the initial population of the Reach Position behaviours to reach two goal positions. Fig. 5-11d shows the courses of the robots under the control of the final population to reach four goal positions. In Fig. 5-11c, the robot started from the low left corner of the environment and was asked to reach the first goal on the robot's left and the second goal on its front. Most of the 50 robots failed to complete the task and created circular-like courses in attempting to steer towards the first goal. In Fig. 5-11d, the robot started from the near centre position of the environment facing to the right side of the environment. The first goal was at the right side of the environment and the second was at the left side. The third goal was located at the upper side of the environment and the fourth one was at the lower side. After 1000 generations of learning, the 50 members of the population were able to control the robots to reach the four goals in an ordered sequence and produced straightforward "4" shaped courses with only small variations. Fig. 5-12 displays the courses produced by the average member of the initial population at stage 1, 2, and 3, the average one of the final population, and the manually tuned Track Path behaviour. The difficulty level of the learning environment was increased by decreasing the width of the path as the learning progressed over stages. The robot was asked to follow the path three times with each chromosome from different initial heading positions, which, therefore, imitated the different path positions. These paths are imaginary so that only the "following path" functionality was learnt, without influences from any physical objects. From Fig. 5-12, it can be seen that the performance of the initial average members of three stages was improved significantly. At the end of the learning process, the behaviour was learnt as intended and was able to control the robot by quickly steering to the centre of the path, then following it straight away to the end of the path. Compared to the learnt behaviour, the manually tuned behaviour did not perform very well and its courses were not as straight as those of the average learnt behaviours.

5.5 Learning of Behaviour Selection Network

5.5.1 Components to be learnt

After individual fuzzy behaviours are learnt, the behaviour selection network, which is responsible to select the best behaviour for the control of the robot at any given time, must also be learned. Here, the structure and function of a behaviour link are first reviewed. According to the definition in Section 3.1.5, a behaviour link is a data structure used to represent the relationship of two linked behaviours. In MARCO, two types of the relationship have been defined: promotion and inhibition. Promotion is the way of distributing one behaviour's activation energy to other behaviours. This usually happens when some behaviours experience execution failures, or unfavourable robot control activities, such as motionless or very slow movement of the robot. Inhibition does the opposite. It is used by one behaviour to subdue other behaviours from taking the control of the robot. The promotion/inhibition links are set-up between the behaviours based on their importance with respect to the safety, goal, and other robot navigational motivations. A behaviour's activation energy is determined by its own situational activation energy and the synthesised energy from all the linked behaviours. Although it is quite obvious to determine the relationship of the behaviours, the selection of appropriate levels of promotion/inhibition is not so straightforward. According to the behaviour selection algorithm described in Section 4.3.3, the outward promotion and inhibition level of one behaviour is calculated as :

$$\text{p\_value}_o = \text{p\_factor} * \text{frus\_level};$$
$$\text{i\_value}_o = \text{i\_factor} * \text{a\_level}_s;$$

The frustration level and situational activation level of the behaviour are produced in the fuzzy control process of the behaviour. They do not need to be learnt. It is *p_factor* and *i_factor* that need to learnt in order to maintain optimal promotion/ inhibition levels for the behaviour selection.

5.5.2 An Incremental Learning Approach

The learning of behaviour links is very different from the learning of individual behaviours because more than one behaviour is involved. To design a practical learning process, several problems have to be considered. First, the behaviour links between some basic reactive behaviours should be learnt first. Once learnt, these behaviour links should not be changed in the learning of other behaviour links. In the robot's navigational activities, some basic reactive behaviours, such as Keep Moving and Avoid Obstacle behaviour must be always present in the low level control layer. They are the most fundamental part of a reactive control system. More purposeful control activities, brought up by other fuzzy behaviours, such as Follow Edge, Reach Position, depend on the support of these reactive behaviours for success. An optimal selection network of those reactive behaviours, once learnt, will form a solid foundation for a robust robot control system. Second, conflicting behaviours should not be learnt together. A behaviour can have promotion/inhibition links with several other behaviours. However, some of them may cause conflicting control activities when becoming active at the same time. For example, to learn behaviour links between Avoid Obstacle and other behaviours, we should not put Follow Edge and Track Path behaviour into a single learning process because they have contradictory control behaviours. In MARCO, such conflicting control activities are resolved by the higher level of the control system, the sequencing layer, which organises a collection of behaviours in harmony for the control of the robot. Therefore, the learning of the behaviour selection network should be carried out for the learning of the behaviour links of every such collection of behaviours, which will be activated by the sequencing layer in a navigation task. The above considerations lead to an incremental learning approach for the learning of the behaviour selection network. Behaviour links between fundamental reactive behaviours are learnt first. The behaviour selection network between those fundamental behaviours and the task-oriented behaviours are then learnt to obtain an optimal behaviour selection network.

5.5.3 Simulation Design and Results

Using an incremental learning approach, the behaviour selection network for our example fuzzy behaviours is learnt through the learning of several collections of behaviours as shown in Fig. 5-13.

To learn the behaviour links is to learn an optimal set of promotion/inhibition factors

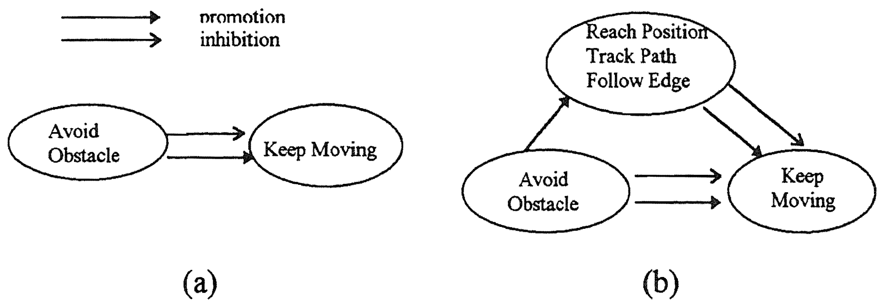


Fig. 5-13 Behaviour Selection Network to be Learned
(a) subnet between fundamental behaviours;
(b) subnet between task-oriented and fundamental behaviours.

which decide promotional/inhibitional level for a behaviour. These factors are taken as genes to form a behaviour link chromosome. Fig. 5-14 presents the chromosomes for the learning of the behaviour selection network.

(a)

p_factor	i_factor

(b)

i_factor$_1$	p_factor$_2$	i_factor$_2$

Fig. 5-14 Structure of behaviour link chromosomes:
(a) Avoid Obstacle - Keep Moving; (b) Task-oriented - AO and KM.

Fig. 5-14(a) is the chromosome representing behaviour links between Avoid Obstacle and Keep Moving behaviours. Fig.5-14(b) is the structure of all chromosomes used to learn the links between example task-oriented behaviours and two fundamental reactive behaviours. Note that the chromosome in Fig. 5-14(b) does not include genes representing the links between Avoid Obstacle and Keep Moving, which has to be learnt first and then remains unchanged.

As already mentioned in Section 5.4.4, the learning of Avoid Obstacle behaviour needs the support of Keep Moving behaviour. This means their behaviour links also have to be encoded into the Avoid Obstacle behaviour's chromosome and then becomes a part of the integrated learning for the Avoid Obstacle behaviour. In this experiment, their optimal behaviour link has already been learnt during the learning of the behaviour and the results are used in the learning of other behaviour links.

The same learning methodology and algorithm used in the learning of individual fuzzy behaviours was employed for the learning of behaviour selection network. It was my intention to verify the effectiveness of such a learning methodology in the robot learning, both for individual behaviours and overall robot control systems. In the simulation, the same set of control parameters was used in the genetic algorithms, such as genetic operators' probability, population size, generation number, etc. The learning of the selection network for the 3 clusters of the behaviours was evaluated in a similar way to the learning of fuzzy behaviours as described in Section 5.4.7. The evaluation functions are as follows:

Reach Position - Avoid Obstacle & Keep Moving:

eval_value = time_weight * steps + ach_weight * closest_distance +
collision_weight * collision_flag/steps^{-2}; (5)

Track Path - Avoid Obstacle & Keep Moving:

eval_value = time_weight * steps + disp_weight * sum_of_dist_to_center +
swing_weight * sum_of_drifted_angle + ach_weight * closest_distance +
collision_weight * collision_flag/steps^{-2}; (6)

Follow Edge - Avoid Obstacle & Keep Moving:

eval_value = time_weight * steps + range_weight * sum_of_min_range +
turn_weight * sum_of_turning_angle +
collision_weight * collision_flag/steps^{-2}; (7)

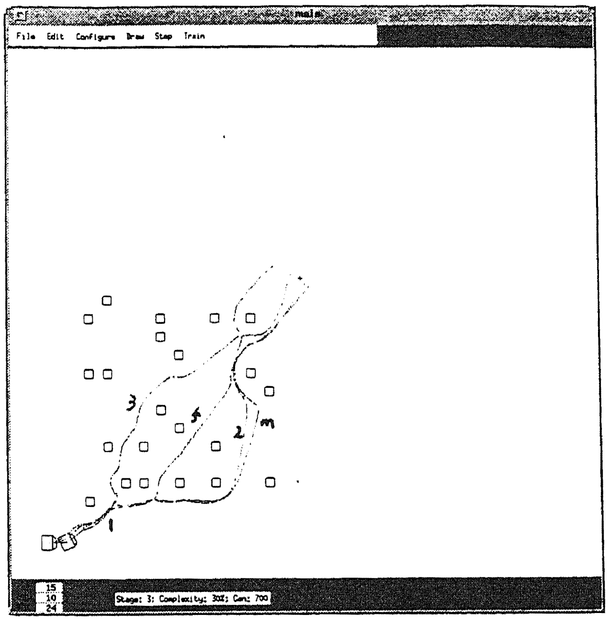

File Edit Configure Draw Step Train
3
f
2
m
1
15
10
24
Stage: 3; Complexity: 30%; Gen: 700

Fig. 5-15 Snapshots of the courses of the robot for the learning of the behaviour selection network, Reach Position-Avoid Obstacle+Keep Moving:
1. the initial best at stage 1;
2. the initial best at stage 2;
3. the initial best at stage 3;
f. the final best;
m. the manually tuned link.

Some of the simulation results are presented. Fig. 5-15 gives the visual displays of the courses produced by the best member of the initial population at the stage 1, 2, 3 and the final population, as well as manually tuned links of the behaviour selection network between Reach Postion, Avoid Obstacle and Keep Moving behaviours. These courses are identified with 1, 2, 3, f and m respectively.

Table 5-8 Behaviour Link Chromosome and Their Performance Indexes

(a) Reach Position - Avoid Obstacle & Keep Moving

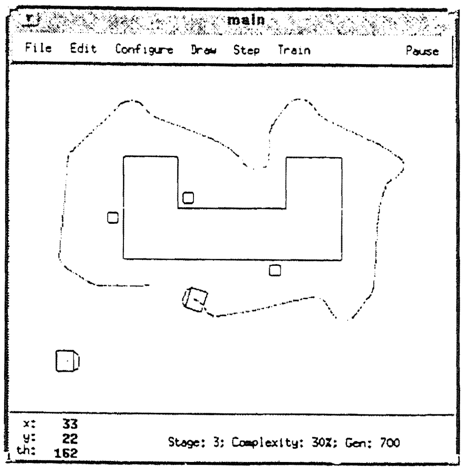
Stage	Chromosome			Performance Indexes		
				steps	achieve	bump
1st init. best	0.444158	0.347369	0.496497	94	6733.43457	1
2nd init. best	0.186135	0.347369	0.196298	931	78.521057	0
3rd init.best	0.171714	0.237879	0.336871	533	76.792549	0
final best	0.170712	0.261922	0.356444	433	43.017998	0
manual	0.15	0.2	0.15	942	43.009029	0

(b) Follow Edge - Avoid Obstacle & Keep Moving

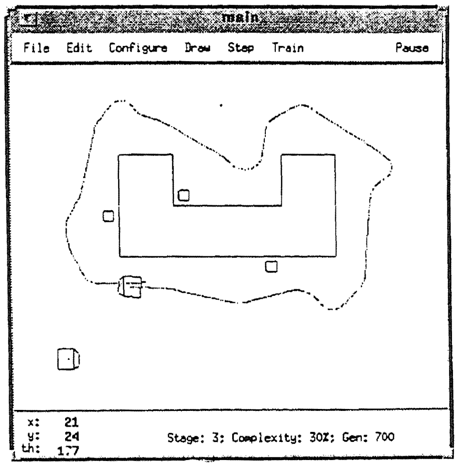
Stage	Chromosome			Performance Indexes			
				steps	range sum	turn sum	bump
1st init. best	0.180584	0.475453	0.034198	1115	61007.82422	145187.2344	0
2nd init. best	0.232373	0.390271	0.103924	1075	50201.22656	145786.625	0
3rd init.best	0.227366	0.393437	0.255269	943	48587.01953	138190.4844	0
final best	0.211281	0.302934	0.158032	877	47963.66016	112254.3516	0
manual	0.15	0.2	0.15	1059	58607.65625	127951.6641	0

(c) Track Path - Avoid Obstacle & Keep Moving

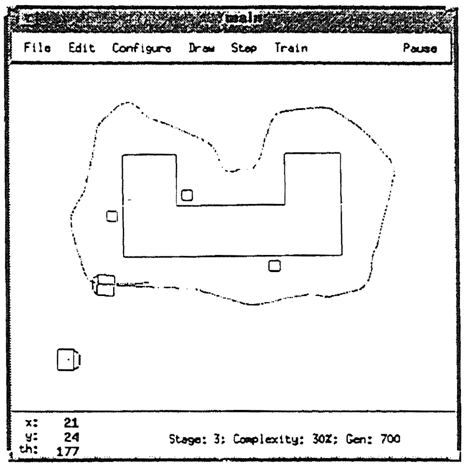
Stage	Chromosome			Performance Indexes				
				steps	disp sum	swing sum	ach	bump
1st init. best	0.329041	0.275558	0.065886	277	165275.375	20931.38281	280.888733	0
2nd init. best	0.123675	0.48875	0.148572	277	142484.1406	21601.01367	367.796265	0
3rd init.best	0.123675	0.48875	0.123436	264	138280.4219	20452.76172	229.148087	0
final best	0.103805	0.402376	0.185513	252	129932.9297	18063.29492	616.036926	0
manual	0.15	0.2	0.15	275	146009.2813	19343.53125	691.950623	0


main
File Edit Configure Draw Step Train Pause
x: 33
y: 22
th: 162
Stage: 3; Complexity: 30%; Gen: 700

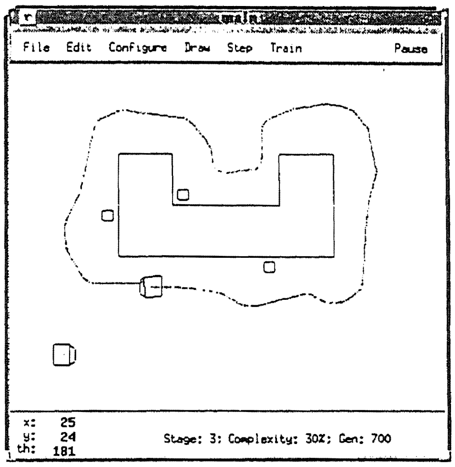
1


main
File Edit Configure Draw Step Train Pause
x: 21
y: 24
th: 177
Stage: 3; Complexity: 30%; Gen: 700

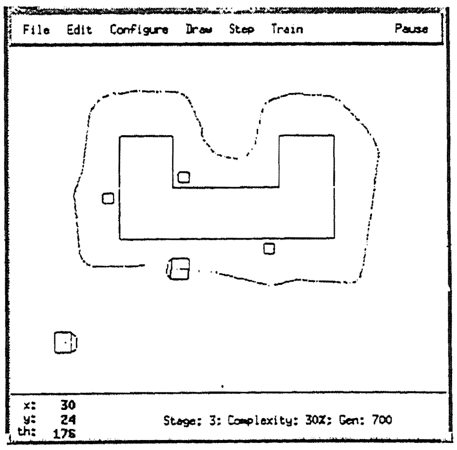
2


main
File Edit Configure Draw Step Train Pause
x: 21
y: 24
th: 177
Stage: 3; Complexity: 30%; Gen: 700

3

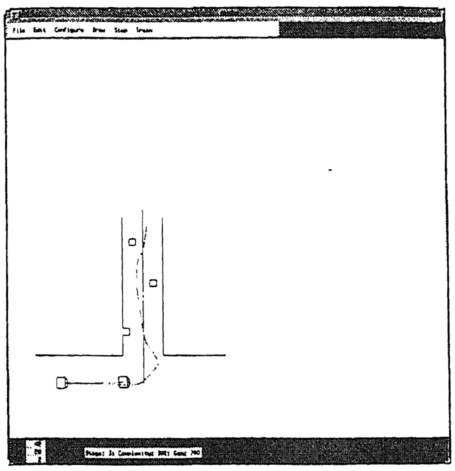

main
File Edit Configure Draw Step Train Pause
x: 25
y: 24
th: 181
Stage: 3; Complexity: 30%; Gen: 700

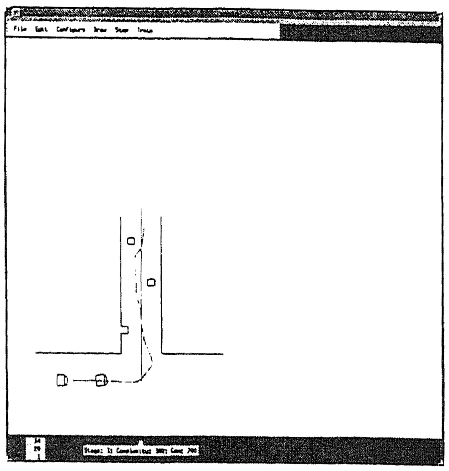
f

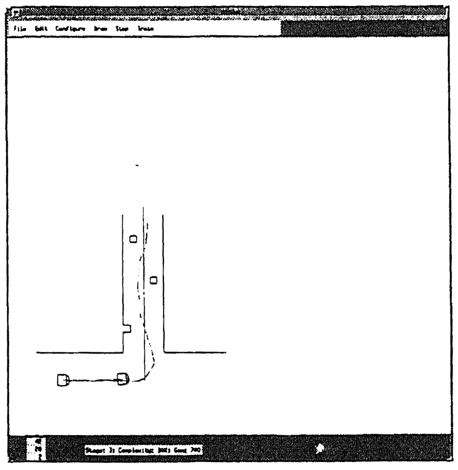

File Edit Configure Draw Step Train Pause
x: 30
y: 24
th: 176
Stage: 3; Complexity: 30%; Gen: 700

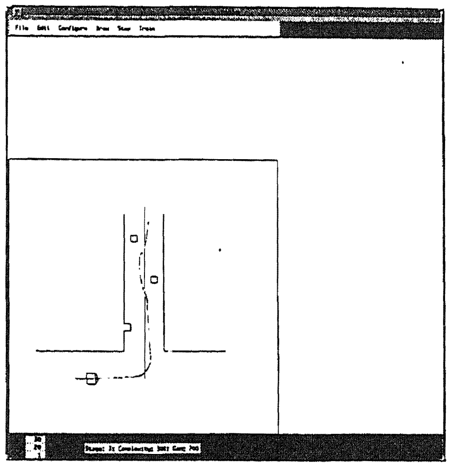
m

Fig. 5-16 Snapshots of the courses of the robot for the learning of the behaviour selection networks:
Follow Edge - Avoid Obstacle+Keep Moving:
1. the initial best at stage 1;
2. the initial best at stage 2;
3. the initial best at stage 3;
f. the final best;
m.and the manually tuned.


1


2


3


f

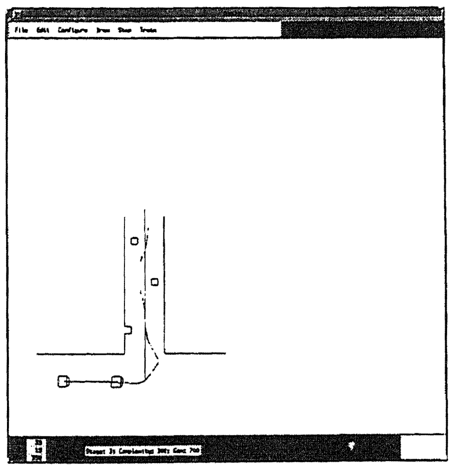

m

Fig. 5-17 Snapshots of the courses of the robot for the learning of the behaviour selection networks,
Track Path - Avoid Obstacle+Keep Moving:
1. the initial bests at stage 1;
2. the initial best at stage 2;
3. the initial best at stage 3;
f. the final best;
m. the manually tuned link.

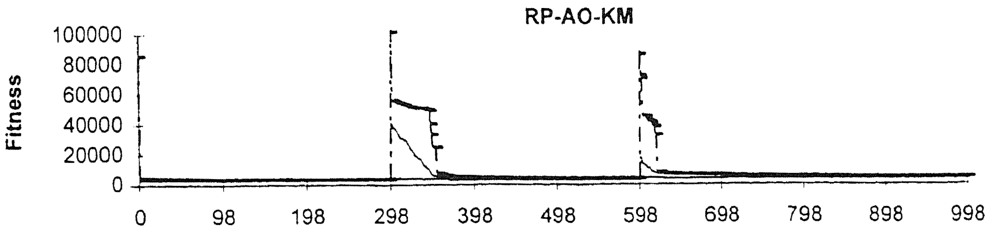
In Fig. 5-15, the robot was asked to reach the goal position at the upper right corner of the environment from the lower left corner. The robot was to manoeuvre through densely scattered obstacles and get to the destination fast, take as straight route as possible and arrive the position precisely. Because the robot had learnt near optimal individual behaviours, which will take their own responsibilities robustly, the overall control fitness of the robot for the task lay in the selection of the behaviours. From the display, we can see that f is the best of the 5 courses, in terms of fastness, straightness and precision, while 2 and 3 are reasonably good and 1 is the least favourite. The robot, controlled by the manually tuned behaviour links, performed not as well as the learned candidates, f and 3. Their behaviour link chromosomes and the robot performance indexes are listed in Table 5-8(a). The similar results for the other two behaviour clusters are presented in Fig. 5-16, Fig. 5-17 and Table 5-8(b)(c).

Fig. 5-18 gives the evaluation charts of the learning algorithms for the behaviour selection networks respectively. A similar convergence pattern to the learning of individual behaviours was observed in the learning processes.

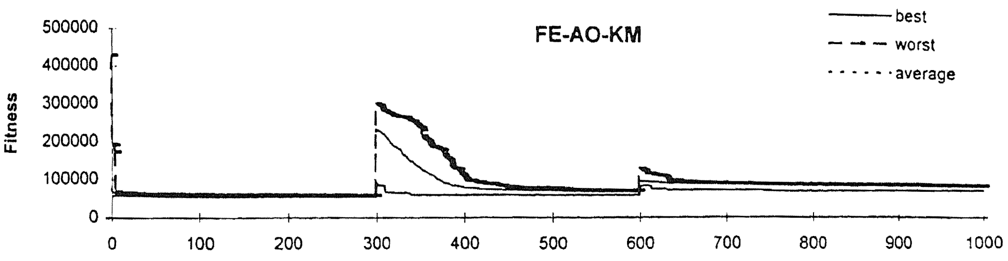
The simulation results show that with an incremental learning approach and the simple-to-complex multistage learning methodology, near optimal behaviour selection networks have been obtained. The genes representing promotion/inhibition factors all converged towards a set of value within small range of variations. The measured performance indexes show a gradual improvement of the controlling performance of the robot through the whole learning processes. The evaluation values also converged at the every stage of the learning and in the final population. These results indicate the effectiveness of the learning methodology in the learning of the behaviour selection network, though the improvement to the robot movement during the learning processes was not as significant as in learning individual behaviours.

5.5 Summary

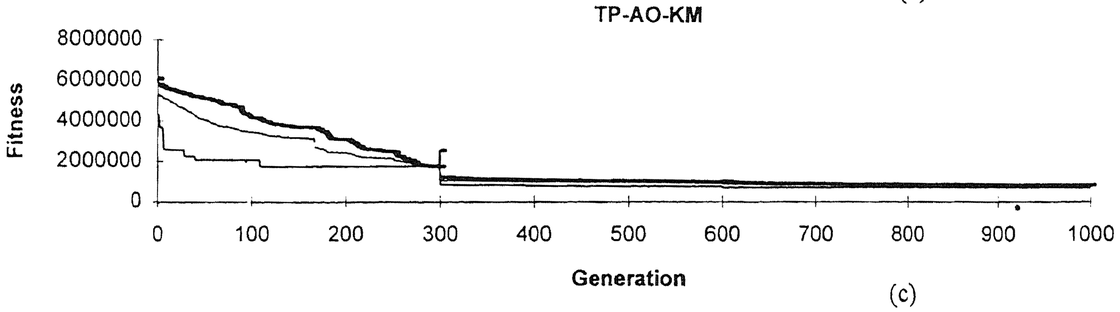
In this Chapter, a learning methodology has been developed to automatically learn membership functions of individual behaviours and the behaviour selection network. This methodology is based on genetic algorithms and contains several principles. To build a robust fuzzy behaviour-based reactive control system, individual fuzzy



(a)



(b)



(c)

Fig. 5-18 Generation vs. Fitness charts of the learning processes for the behaviour selection network:
(a) the cluster with Reach Position behaviour;
(b) the cluster with Follow Edge behaviour;
(c) the cluster with Track Path behaviour.

behaviours are first learnt to obtain robust individual functionality in the control system. Then, the behaviour selection network are learnt to obtain good overall control of a mobile robot control system. The learning methodology emphasises the following points. First, every component of the control system is learnt for its own functionality. Specific environment features and configurations are provided for the learning process of each component in order to ensure that each component is learning for its own role in the control system. Second, learning environments are generalised and a variety of randomised configurations are presented in the course of learning, in order that the learning results are useful for building real world mobile robots. Finally, the learning process follows a simple-to-complex multistage learning course for the better search for optimal solutions. The design of the genetic algorithms enables an initial wide and deep exploration and a gradually localised tuning of the population as the learning progresses.

The learning methodology has been used to learn both membership functions of individual fuzzy behaviours and the behaviour selection network for MARCO's experimental low level control layer. The effectiveness of the learning methodology has been demonstrated in the simulations. For all example fuzzy behaviours, near optimal membership functions have been automatically learnt. Their near optimal behaviour selection network has also been learnt. From the experiment, it can be seen that it is possible to automatically learn a low level control system using this learning methodology and therefore greatly reduce the difficulties and efforts in configuring a fuzzy behaviour-based reactive control system for a robot.

Chapter 6 Experiments

The preceding chapters described MARCO, a two layer control architecture for mobile robot navigation and the learning of low level control layer. This chapter will attempt to demonstrate that an actual implementation of a MARCO control system will indeed robustly control mobile robot navigation in real world indoor environments. Ideally one would like to prove that MARCO can solve the navigation problems in dynamic, uncertain and unpredictable real world in face of noisy and imprecise information. Unfortunately, it is very difficult to prove anything about MARCO because it is hard to give rigorous definitions for terms like "robust behaviour" and "unstructured environment". To quote Firby, "without a rigorous definition to prove things about, evaluation of the system must lie in actual performance"[Firby89].

Before studying MARCO system's performance on realistic mobile robot navigation problems, MARCO was implemented into a simulated mobile robot, SIMAR, to perform two simulated navigation tasks. One is a construction task, concrete floor slab finishing and the other is building security patrolling. As evidence that a MARCO control system can navigate and complete tasks effectively, this chapter produces the following types of information: traces of mediating activities in the sequencing layer which organises control activities in the low level control layer, traces of fuzzy behaviour activation levels in the low layer and traces of physical movement of the robot.

The simulation results presented shows that a MARCO system does indeed behave as suggested in the preceding chapters.

6.1 A Simulated Robot System

A simulated robotic system has been developed for the research of MARCO control architecture and possible applications. The system consists of two major parts: a controller and a simulator. The controller is, in fact, a MARCO control system, which performs sensor interpretation, world modelling and the two levels of control activities. The simulator contains simulated world construction, sensing, robot motion, servoing and other modules. The two subsystems communicate over a TCP/IP link. The robot controller solicits sensor and robot movement data from the simulator and sends control commands to the simulator after fuzzy behaviour-based control processes. The design of the simulated robot system is partly based on the configuration of a real experiment robot; Marco[†] is a 4-wheel driven mobile robot and is equipped with a laser scanner, a sonar radar, odemetry encoders and a bump ring. In simulation, sensor data is produced from a simulated laser scanner, odemetry encoders and a bump ring. Range data and encoder readings are perturbed with noise. The simulated worlds are 2D world models, constructed by the simulator. World models are analogical real worlds, with linear segments representing the

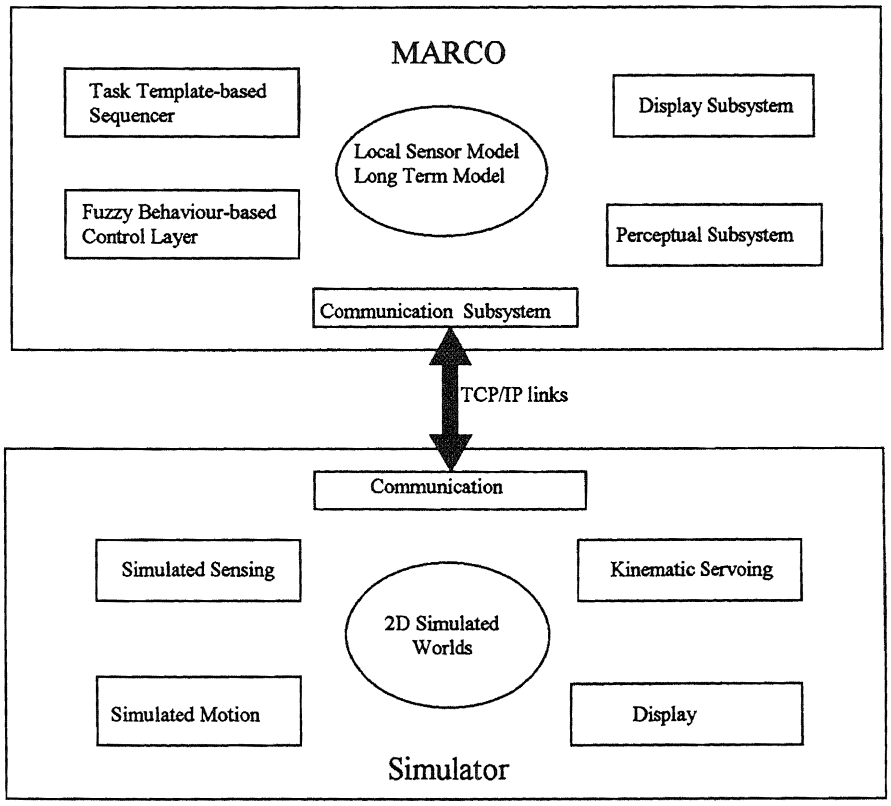


Fig. 6-1 Diagram of A Simulated Robot System, SIMAR

vertical surfaces of corridors, hallways, walls and the objects in an environment. The architecture of the simulated robot system is presented in Fig. 6-1. The parameters of

[†] Mobile a-utonomous r-obot for co-nstruction

SIMAR are selected mainly based on a commercial mobile robot, Pioneer, and given as follows:

MTV = 300mm/s — max. translational speed, in millimetre per second;

MRV = 1.0 rad/s — max. rotational speed, in radian per second;

$$\text{MTA} = \begin{cases} 100\ \text{mm/s}^2 & \text{— max. translational acceleration at slow speed(< 10 mm/s);} \\ 100\ \text{mm/s}^2 & \text{— max. translational accelaration at normal speed;} \\ 200\ \text{mm/s}^2 & \text{— max. translational acceleration at braking speed(< 0 mm/s);} \end{cases}$$

$$\text{MRA} = \begin{cases} 0.4\ \text{rad/s}^2 & \text{— max. rotational acceleration at slow speed;} \\ 0.4\ \text{rad/s}^2 & \text{— max. rotational accelaration at normal speed;} \\ 0.8\ \text{rad/s}^2 & \text{— max. rotational acceleration at braking.} \end{cases}$$

wheel encoder noise: 1% randomness in readings;

angle encoder noise: 2% randomness in readings;

angle drift over distance: 0.3% randomness on 0.5 degree every 100mm distance;

The robot motion is servoed through a simple kinematic model, a trapezoidal velocity function as shown in Fig. 6-2(a). When the simulated robot receives a speed command, it accelerates or decelerates at a constant rate set internally to the required speed. Rotational headings are achieved in the similar way by the rotational heading servoing as shown in Fig. 6-2(b).

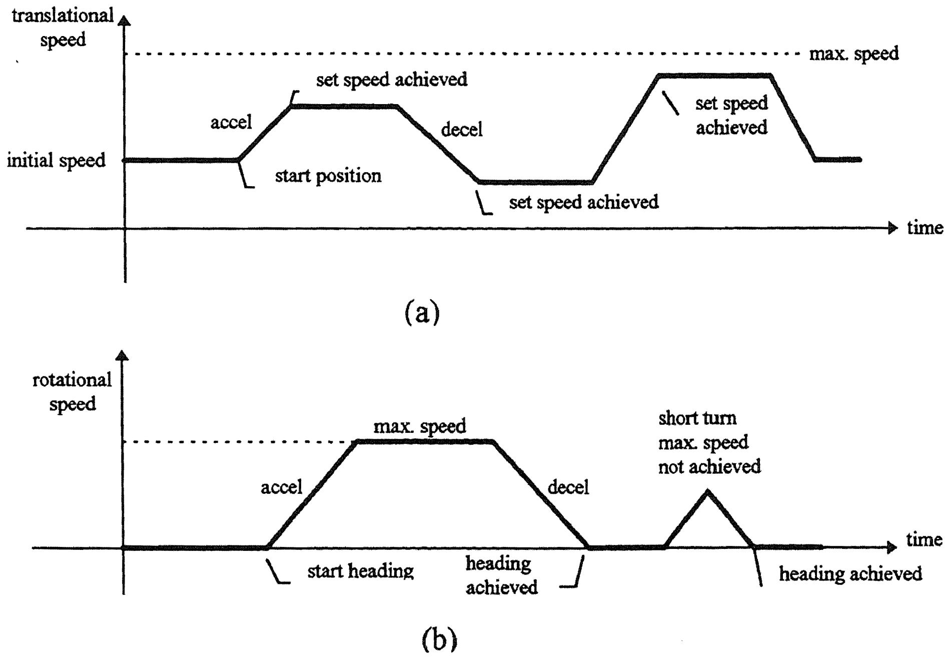


Fig. 6-2 A Simple Kinematic Servoing Model
(a) translational speed servoing;
(b) rotational heading servoing.

The simulated system has been developed in SPARC workstation 10(55MHz) with 32MB memory and 1GB hard disk. The development environment is given as follows:

1. Solaris 2.4 Operating System;
2. X11R5 X window library;
3. Motif 1.2.4 library;
4. SPARC work professional C, version 3.1.

This simulated robotic system will be used for both of the experimental tasks, concrete slab finishing and building security patrolling. In reality, the two tasks will require very different end effectors for the very different operations, which may have very big disparity in navigation performance using the same robotic system. It is assumed that the impacts of different end effectors and tasks to the navigational performance of MARCO control architecture can be reduced by means of more elaborate design of the mechanics and control systems, considering robot dynamics and employing more task templates and fuzzy behaviours with regard to various aspects of a task execution. The purpose of the experiments here, is to demonstrate the abilities of the two layer control architecture in the robot navigation.

6.2 A Concrete Slab Finishing Task

Concrete slab finishing in construction site is usually done by plasterers using a troweler[Arai89][Wing89]. After concrete is placed on floor slab, roughly levelled and allowed to harden, the rough surface of the concrete is then flattened and smoothed. Generally, this work requires a plasterer to operate and guide a trowel in a regular pattern over the whole surface of the setting concrete slab. This experiment will not consider the physical troweling actions but the navigation and control which guides a troweling machine over the entire area of the floor surface. SIMAR's objective is to find and enter a room, finish the concrete floor slab of the room and then exit the room.

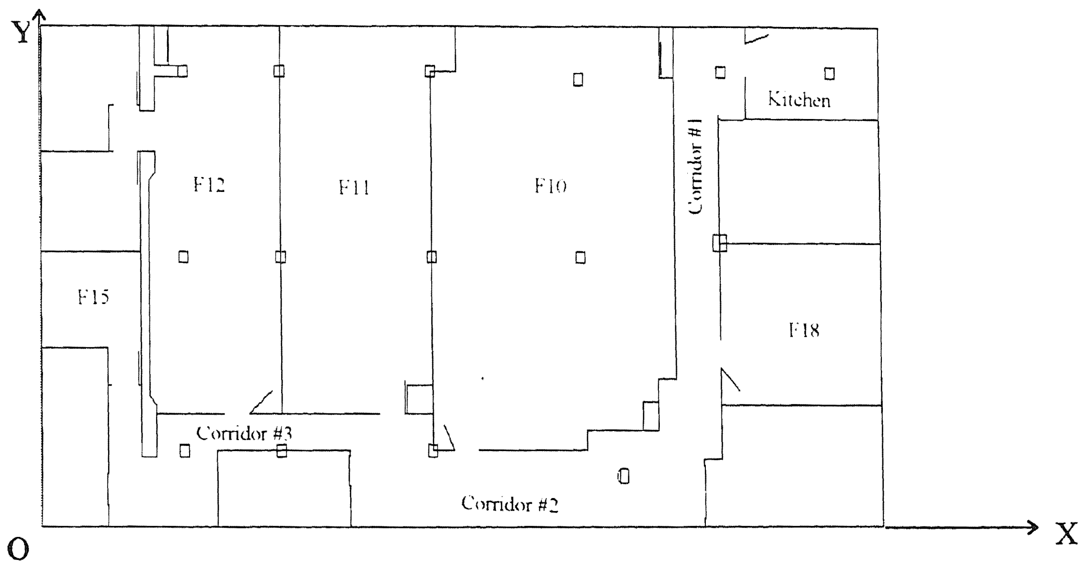


Fig. 6-3 The Layout of the First Floor of the Spire Research Centre

6.2.1 Experiment Set-up

Fig. 6-3 shows the layout of the experiment world, a simplified real environment, based on the 1st floor of the Spires Research Centre. In this simulated world, the walls are represented by line segments. Doors are the openings in line segments with a single connected line segment indicating *close* or *open* state. Rooms are represented by a set of enclosed line segments with openings. Corridors are identified by a pair of parallel line segments outside of rooms with certain width and length constraints. The robot's main experiment area consists of corridor #2 and room F10. This simulated world is constructed by the simulator using approximately measured data from the real floor map. From the simulator's point of view, the entire world consists of line segments. The simulator does not have any

```
;; Simplified feature-based map of the first floor of the Spire
;; Research Centre
;; corridor(id)  x, y, th, length, width
;; door (id)  x, y, th, width, name

CORRIDOR(1)  -3000,  -500, -90,  12000, 1600
CORRIDOR(2)  -3000,   350,    0,  12500, 2000
CORRIDOR(3)   7000, -1250,    0,   7500, 1000

DOOR (1) 5500, -750, -90, 900, F10
DOOR (2) 8000, -1780, -90, 890, F11
DOOR (3) 13500, -1780, -90, 890, F12
DOOR (4) .....
....
ROOM (1)
```

Fig. 6-4 Simplified Feature Map of the First Floor

perceptions of environment features, such as wall, door, corridor, etc. These perceptions are actually derived by the perceptual subsystem of the controller, which receives and interprets the simulated environment data from the simulator. Except for the sensor information interpreted through the perceptual subsystem, a simple feature based map is also provided to the robot controller in order to have an approximate world model in advance. This simplified floor map is presented in Fig. 6-4. The controller loads this map and constructs these features and stores them in the long term model. Note that this map is based on the robot co-ordinates. The simulator still uses geometrical co-ordinates for all of its processing. Fig. 6-5 gives a glimpse of the robot centred view of the environment, which includes the part of room F10, door and corridor #2.

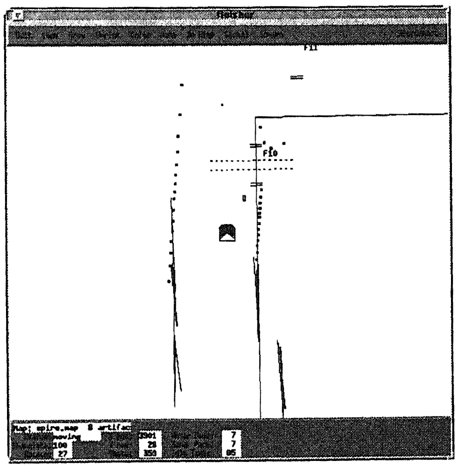


Fig. 6-5 Part of Environment in the Robot Centred Co-ordinate
..... : sensed location;
— : map location.
$\equiv$: door way

6.2.2 Navigation and Task Execution Plan

A navigation plan is usually produced and provided by the highest level of the robot control system, called the deliberative layer in a three layer architecture. Since planning

```
;;; Plan for slab finishing in room F10
;;; This plan consists of starting positions and paths.
;;; Two end points of each path, together with their out
;;; angle define each path or finishing lane. Lane order is
;;; from right to left.
;;; lane width:  400
;;; lane overlap: 0
;;; edge: 234
;;;
;;;  LANE (id) x1 y1 x2 y2 th1 th2 width
;;;
;;;  ANCHOR (id) x y th
;;;
;;;  lane(i) = 400 *i +134;
;;;

POS (1) 8234  2659   90
LANE (1)  8234  2659  8234  9977  90 180

POS (3)  7634  11435  -90
LANE (3)  7634  11435  7634  2011 -90 180

POS (5)  7034  1201   90
LANE (5)  7034  1201  7034  11435  90 180

POS (7)  6434  11435   -90
LANE (7)  6434  11435  6434  1201 -90 180

POS (9)  5834  1201   90
LANE (9)  5834  1201  5834  11435  90 180

POS (11)  5234  11435  -90
LANE (11)  5234  11435  5234  1201  -90 180

POS (13)  4634   634  90
LANE (13)  4634   634  4634  11435  90 180

POS (15)  4034  11435   -90
LANE (15)  4034   434  4034  634  -90 180

POS (17)  3434   634  90
LANE (17)  3434   634  3434  11435  90 180

POS (19)  2834  11435   -90
LANE (19)   2834  11435  2834  634  -90 180

POS (21)  2234   634  90
LANE (21)   2234   634  2234  11435  90 180

POS (23)  1634  11435   -90
LANE (23)   1634  11435  1634  634  -90 180

POS (25)  1034   634   90
LANE (25)   1034   634  1034  10139  90 180

POS (27)   434  10139   -90
LANE (27)   434  10139  434  1234  -90 0
ANCHOR (1) 5265  10011  0
ANCHOR (2) 5265  6070  0
POS   (3) 1634   434  0
FROM  (2) 2634   434  0
TO    (1) 434  1034  0
```

Fig.6-6 Task Plan for Concrete Slab Finishing

is not the topic of this thesis, it is assumed that a plan has already been given and reactive planning problems need not to be considered. In this experiment, the navigation plan is presumed to consist of the following parts. First, the robot finds and enters room F10. Second, the robot executes its main task, finishing the room. Finally, after the task has been completed, the robot moves out of the room. This navigation plan is sketched as follows:

1. find and follow corridor #2 until close to door of room F10;
2. find and enter F10;
3. execute concrete slab finishing task;
4. find and move out of room F10.

The execution of step 1,2, 4 involves the checking of the long term model for the availability of related environment features. This is different from step 3, which entirely depends on a task execution plan produced using extensive domain knowledge. Here, the experiment employs mostly a regular pattern [Kajima89] and also some special types of operation to form such a task plan. The regular pattern of operations consists of up and down straight movement actions of the robot to cover most of the rectangular area. Other areas, such as the vicinity of pillars and wall edges, which are difficult to operate using the regular pattern, are finished with special types of movement control. It is supposed that a detailed task execution plan as shown in Fig. 6-6 has been generated and given to the control system for the concrete slab finishing task. This plan is made up of the sequences of action goals for the task. The term, POS, represents a position point the robot must reach to. LANE means a straight path to be trawled. ANCHOR, FROM and TO represent points for the robot to position itself during its special types of operations. The control system activates the plan after the robot enters the room and then carries out the concrete slab finishing task by executing the sequence of these movement control actions prescribed by the plan.

6.2.3 An Implementation of Two Layer Control System

The main objective of this experiment was to demonstrate the capabilities of the MARCO control architecture and its possible applications in real world problems. The key points that make this experiment a good test of MARCO are the realistic and rich types of environment features presented for the evaluation of MARCO's feature-invoked fuzzy behaviours at the control layer and the chances of organising these low level control activities in the sequencing layer. At the low level of control, the following fuzzy behaviours were employed in the experiment: Avoid Obstacle, Keep Moving, Reach Position, Follow Edge, Follow Corridor, Track Path, Cross Door and Recover Stall. These behaviours can be organised into various behaviour clusters by the sequencing layer. The actual controlling activities were carried out by these behaviours. At the sequencing layer, the mediation of low level controlling activities was realised through task templates. The task net in a task template consists of either functions or other task templates. The function in a step of the task net can only be executed when a task template is instantiated. In this experiment, 3 main task templates, *sequence, follow, monitor* and 6 functions, *check, switch, terminate,* *fetch_plan, invoke, set*, were developed. The main task template, *sequence* is defined below.

```
Define-Task Template
   (Name (sequence))
   (Succeed (state SUCCESS))
   (Method
      (Task-Net
        (t1 (follow) for t2)
        (t2 (check ?room)
            (and (found ?room)
                  (near ?room) for t3)
        (t3 (switch follow_corridor enter_room ?beh ?state)
             (SUCCESS ?state) for t4)
        (t4 (terminate ?beh) for t5)
        (t5 (fetch_plan ?goal ?type)
             (POS     ?type) for t6
             (LANE  ?type) for t7
             (ANCHOR ?type) for t8
             (FROM ?type) for t9
             (TO ?type) for t10)
         (t6 (invoke reach_position ?beh ?goal ?state)
             (ACHIEVED ?state) for t11)
         (t7 (invoke track_path ?beh ?goal ?state)
             (ACHIEVED ?state) for t11)
         (t8 (invoke reach_position ?beh ?goal ?state)
             (ACHIEVED ?state) for t12)
         (t9 (invoke reach_positon ?beh ?goal ?state)
             (ACHIEVED ?state) for t13)
         (t10 (check ?goal ?state)
               (ACHIEVED ?state) for t14)
         (t11 (terminate ?beh ) for t5)
         (t12 (switch reach_position follow_edge ?beh ?state) for t15)
         (t13 (switch reach_position follow_edge ?beh ?state) for t5)
         (t14 (switch reach_position leave_room ?beh ?state)
               (ACHIEVED ?state) for t16)
          (t15 (check ?time)
                (= ?time WAIT_TIME) for t5)
          (t16 (terminate ?beh) for t17)
          (t17 (set SUCCESS ?state))
```

This task template was the primary task template which initiated the sequencing activities. It employed functions and another task template, *follow* to carry out the mediation task. The functions can bring up other task templates, in this case, *monitor*, and also perform normal processing. For example, the function *invoke* will initiate a task-oriented behaviour in the low level control layer and also create a task with *monitor* task template to monitor the progress of that behaviour.

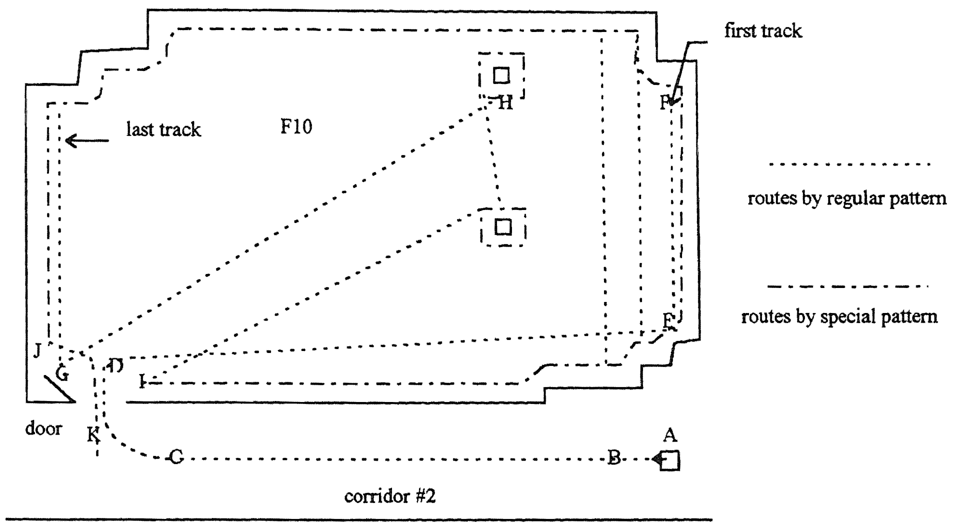


Fig. 6-7 Approximate Reference Trace Routes and Positions

6.2.4 Detailed Traces

One way to illustrate the effectiveness of the MARCO system is through the use of a detailed execution trace. This section includes two traces: a sequencing activity trace that shows the creation, execution and completion of all task templates required to carry out the concrete slab finishing task, and a fuzzy behaviour activation level trace that shows the way fuzzy behaviours competing and co-ordinating with each other through the fuzzy logic control processes and their selection network. The traces are intended to show the two layer control architecture adapting to the environment changes and uncertainty and carry out effective navigation control. Traces also give insight into the way the task scheduler functions. Fig. 6-7 gives some of expected robot routes and approximate positions for the trace references in the whole operation.

6.2.4.1 A Sequencing Trace

To make the detailed trace easier to follow, an overall tracing and controlling processes are first presented. According to the navigation plan in Fig. 6-6, the robot will be started from (A), controlled by several reactive behaviours. Following corridor activity will be initiated approximately at (B) when the corridor #2 is found. At position (C), the robot will stop following corridor activity and start door crossing when the door of room F10 is found. At position (D), the door crossing will be terminated and reaching position activity will be initiated. This activity will lead the robot to the starting position (E) of the first troweling path. The trace will show the sequencing activity for finishing the first path from E to F. The rest of the straight up and down troweling activities will be skipped. The trace will continue from (G), the end of the last straight troweling path, showing the sequencing activities which change the regular movement to a special troweling operation. From position (H), the robot will be guided to move around a pillar and finish its close surrounding area. This troweling operation cannot be efficiently finished using a regular pattern of operation. Follow Edge behaviour will be initiated for the special operation. Follow Edge behaviour is unique that it does not need a specific plan to act like other task-oriented behaviours. In this sense, it can also be seen as a reactive behaviour. However, more purposeful uses of the behaviour can only be realised under the control of the sequencing layer. The trace will demonstrate the uses of Follow Edge Behaviour to finish the first pillar with the help of an anchoring goal position, (H), from where the edge following activities can start. The trace will then skip the second pillar and proceed to show the sequencing actions for the initiation of edge following activities to finish the inside edge of the room by using Follow Edge behaviour. The trace will present the starting actions for the operation from position (I) and the termination process at (J). The final trace will follow the robot leaving the room, involving finding the door of room F10 at position (J) and completing the navigation task at the final position (K).

In the trace description that follows, the reference will be made at the current execution step of a task template and the robot position, (X, Y ,TH). The state (X, Y, TH) stands for the robot X/Y position and its heading within the first floor co-ordinate with the low left corner as origin, indicated in Fig. 6-3. This state information is mainly for trace purposes. A robot-centred co-ordinate system is used in the actual control system. In addition, the following uppercase characters, (A, B, C, D, E, F, G, H, I , J), will also be used to reference approximate positions as indicated in Fig. 6-7. Before the trace started, the robot perceptual task, communication task, low level control task were first initialised and started. The trace was recorded for the task templates which carried out the sequencing activities. The trace message is explained here. Line starting with "&&" indicate the main sequencing actions by the task templates. Lines starting with "--" show the current step of task net executed by the task scheduler, perceptual information or plan goal newly extracted. Lines with prefix "...." indicate the current active fuzzy behaviours in the low level control layer.

The first line of the trace is:

```
&& starting sequence it, top level.
```

This output tells us the invocation of top level sequencing task template. The task scheduler creates a top level task with the name "sequence it" and inserts the task into the task queue. At this point, the task scheduler has not executed the task but has been ready to start with the first step of the task template. After initialisation of the task, the task scheduler starts next cycle of executions of the tasks on the task queue. When it encounters the task "sequence it" again, the following trace continues:

```
-- Step: 10, state: #[Marco state X: 20.41m, Y: 1.46m, TH: 180.0]

&& Starting following, dad sequence it
-- Current behaviour cluster:
.....Avoid Obstacle
.....Keep Moving
.....Recover Stall
```

The trace reports the execution of the first step in the task net of *sequence* task template. The step tag is 10. A subtask, called *following* is generated and placed into the task queue in this step. This subtask's parent is task "sequence it". The three reactive fuzzy behaviours have been created by the low level control task. Currently, the robot is still in its initial state and is controlled only by the three reactive behaviours. Although "following" task has been in the task queue, its main function at this stage is monitoring the availability of a corridor feature. From its starting position (A), the robot moves along corridor #2. While the low level control layer takes care of the robot survival by the three reactive behaviours, the sequencing layer currently involves the activity brought about by two tasks, the top level "sequence it" and its subtask "following". After generating the subtask, the task "sequence it" enters the step which monitors a room feature, and at the same time, "following" subtask checks a corridor feature until it is found by the perceptual subsystem.

```
-- Step: 20, state: #[Marco state X: 20.40m, Y: 1.46m, TH: 180.0]
-- Found a corridor

&& Starting follow it, dad following
&& Starting behaviour Follow Corridor
-- Current behaviour cluster:
.....Avoid Obstacle
.....Follow Corridor
.....Keep Moving
.....Recover Stall
```

This trace is generated by the "following" subtask when it is executed by the task scheduler and a corridor feature has been found at position (B). The subtask "following" creates its own subtask, named "follow it" which is instantiated from *monitor-behaviour* task template. The subtask "follow it" simply monitors the progress of a behaviour and sets the success state when the behaviour achieves its goal. It has not been implemented with more functions, such as failure reporting, or recovery in this experiment. "following" also initiates Follow Corridor behaviour at the same step. The execution of this step results in three sequencing tasks in the task queue, "sequence it", "following", "follow it" and four active fuzzy behaviours at the low level control layer. From now on, the robot's activity is purposeful corridor following, not just reactive survival. The task scheduler is still executing the tasks in turn. The top level task "sequence it" keeps on monitoring for a room feature. The task "following" checks its own state of execution and the task "follow it" examines the achievement of Follow Corridor behaviour. The robot is guided by Follow Corridor behaviour to follow the central lane of corridor #2. After travelling along the corridor for about 4m, a door is detected and registered as the door of room, F10 at position (C). As required by our sketch plan, the sequencing layer should stop corridor following and initiates door entering activity. The trace below shows these actions:

```
-- Step: 15, state: #[Marco state X: 16.91m, Y: 1.43m, TH: 195.8]
-- Found a door

&& Deleting following
&& Deleting follow it
&& Deleting behaviour Follow Corridor
-- Current behaviour cluster:
.....Avoid Obstacle
.....Keep Moving
.....Recover Stall

&& Starting enter it, dad sequence it
&& Starting behaviour Follow Door
-- Current behaviour cluster:
.....Avoid Obstacle
.....Cross Door
.....Keep Moving
.....Recover Stall
```

Note that this trace is produced by the task "sequence it". The step tags of a task are independent of the ones of other tasks. These numbers only represent separate steps and are not necessarily in order. The step tag, 15 of the current trace has no relation with that of the last trace which is created by the "following" task. After the door of F10 has been found and the robot is quite near the door, the task "sequence it" forces the subtask "following" to stop by setting its task state to REMOVE. The task scheduler then removes the task from the task queue in the next cycle of task execution. The removal of a task also means the deletion of all its subtasks and initiated behaviours by the task scheduler. This step causes the removal of "following", "follow it" and behaviour Follow Corridor. Door entering activity is brought up at position (C) through the creation of the subtask "enter it" and the behaviour, Cross Door. The task "enter it" is also a subtask instantiated from the *monitor-behaviour* task template. After this step, the task queue contains two sequencing tasks, "sequence it" and "enter it". The low level control layer consists of four fuzzy behaviours, which co-operate to control the robot going through the door. The task monitors the state of the subtask "enter it", which, in turn, checks the progress of Cross Door behaviour. When the robot successfully goes through the door to position (D), the task "enter it" changes its state to SUCCESS and the task "sequence it" then sets the state to REMOVE, resulting in the termination of the "enter it" task and Cross Door behaviour. This is indicated by the following trace:

```
-- Step: 20, state: #[Marco state X: 14.94m, Y: 2.56m, TH: 90.6]
```

```
&& Deleting enter it
&& Deleting behaviour Cross Door
-- Current behaviour cluster:
.....Avoid Obstacle
.....Keep Moving
.....Recover Stall
```

The robot's main task is to trowel the floor of room F10. Once it gets into the room, the sequencing task should start the task plan execution sequences.

```
-- Step: 40, state: #[Marco state X: 14.94m, Y: 2.56m, TH: 90.6]
-- Had a position goal

&& Starting go to pos, dad sequence it
&& Starting behaviour Reach Position
-- Current behaviour cluster:
.....Avoid Obstacle
.....Reach Position
.....Keep Moving
.....Recover Stall
```

The above trace shows the current step, tag No. 40, executed by the task scheduler for the task "sequence it", the only remaining sequencing task in the task queue. Step 40 functions as a plan dispatcher, which fetches a plan goal and initiates a corresponding goal seeking activity. At this point, a position point, (E), is the first goal for the robot to complete after it enters the room. The task "sequence it" instantiates a *monitor-behaviour* subtask, "go to pos" and initiates Reach Position behaviour. The robot sets off from position (D) towards the specified position (E) as the first task execution step, controlled by four behaviours and monitored by two tasks.

```
-- Step: 85, state: #[Marco state X: 21.77m, Y: 4.78m, TH: 17.0]

&& Deleting go to pos
&& Deleting behaviour Reach Position
-- Current behaviour cluster:
.....Avoid Obstacle
.....Keep Moving
.....Recover Stall
```

The trace continues with the above information. As indicated at Step 85, the task scheduler deletes the monitoring subtask "go to pos" and behaviour Reach Position when the robot successfully arrives at the position (E). After that, the robot continues to execute next goal of the task plan. This is initiated by the "sequence it" task as shown in the following trace.

```
-- Step: 40, state: #[Marco state X: 21.77m, Y: 4.78m, TH: 17.0]
-- Had a track goal

&& Starting track it, dad sequence it
&& Starting behaviour Track Path
-- Current behaviour cluster:
.....Avoid Obstacle
.....Track Path
.....Keep Moving
.....Recover Stall
```

The second goal of the plan is a track, between position (E) and (F), which the robot is asked to trowel. The task "sequence it" generates the subtask "track it" and Track Path behaviour in the low level control layer. The troweling activities are finished when the robot moves to the end of the path, position (F), and the task scheduler removes the subtask "track it" and Track Path behaviour at Step 50 of the "sequence it" task. The robot has moved for about 8m as indicated by the robot's previous and current state in the trace.

```
-- Step: 50, state: #[Marco state X: 21.67m, Y: 12.01m, TH: 108.7]

&& Deleting track it
&& Deleting behaviour Track Path
-- Current behaviour cluster:
.....Avoid Obstacle
.....Keep Moving
.....Recover Stall
```

As discussed earlier, the robot employs mostly a regular pattern of troweling to finish the main rectangular area of the floor. This regular pattern of action is mainly controlled by two task-oriented behaviours, Reach Position and Track Path. They are invoked respectively by the sequencing task when a position, or a track is provided. The robot is first guided to a starting position and then trowels a specified track. The two types of control actions are combined to finish one piece of straight track. From the assumed plan given in Section 6.2.2. We know that there are 14 straight tracks. This means that the robot must move up and down 14 times to finish the main area of the floor. The trace of sequencing activities to finish the first track has been presented. The rest of the traces are skipped because they are all the same except for the different robot states. The complete sequencing trace for the slab finishing task is provided in Appendix C. From here, we proceed to the last trace of sequencing activities when the robot finishes the main area and starts to do some special type of finishing work. The trace restarts below.

```
-- Step: 50, state: #[Marco state X: 14.22m, Y: 3.58m, TH: 271.9]

&& Deleting track it
&& Deleting behaviour Track Path
-- Current behaviour cluster:
.....Avoid Obstacle
.....Keep Moving
.....Recover Stall
```

The robot moves to the end of the last troweling track, position (G), and the task scheduler executes Step 50 of the task "sequence it" which terminates the subtask "track it" and behaviour Track Path. On the task queue, there is only one sequencing task, "sequence it", while at the low level control layer, three reactive behaviours remain. The current step of the task "sequence it" is Step 40, which is to be executed in the next round of task execution by the task scheduler. The results of the new execution is shown as the trace continues.

```
-- Step: 40, state: #[Marco state X: 14.22m, Y: 3.55m, TH: 274.9]
-- Had a anchor goal

&& Starting go to pos, dad sequence it
&& Starting behaviour Reach Position
-- Current behaviour cluster:
.....Avoid Obstacle
.....Reach Position
.....Keep Moving
.....Recover Stall
```

This time, an anchor position, (H), is fetched from the plan and a goal reaching activity is launched by the initiation of Reach Position behaviour. When the robot reaches the anchor point, the sequencing task produces the following trace.

```
-- Step: 70, state: #[Marco state X: 18.85m, Y: 12.03m, TH: 71.4]
-- Anchored to a wall edge

&& Deleting go to pos
&& Deleting behaviour Reach Position
-- Current behaviour cluster:
.....Avoid Obstacle
.....Keep Moving
.....Recover Stall

&& Starting hug it, dad sequence it
&& Starting behaviour Follow Edge
-- Current behaviour cluster:
```

```
.....Avoid Obstacle
.....Follow Edge
.....Keep Moving
.....Recover Stall
```

The task "sequence it" removes the subtask "go to pos" and also Reach Position behaviour. It then starts another subtask "hug it" and behaviour Follow Edge at position (H). The purpose of this sequencing activity is to organise the low level control layer controlling the robot to move around a pillar and finish its close surrounding area. A goal position is used as an anchoring point where the edge following activities can start. The task "sequence it" and its subtask make sure that the right sequences of control actions are carried out. The completion of the surrounding of the first pillar is indicated in the trace below.

```
-- Step: 80, state: #[Marco state X: 16.14m, Y: 13.84m, TH: 173.8]

&& Deleting hug it, dad sequence it
&& Deleting behaviour Follow Edge
-- Current behaviour cluster:
.....Avoid Obstacle
.....Keep Moving
.....Recover Stall
```

The troweling of the second pillar area is finished in the same way. After finishing the pillars, the robot is asked to finish the inside edge of the room. We skip the second pillar finishing trace and the trace continues from anchoring the robot close to the position (I), where it starts to follow the wall edge of the room.

```
-- Step: 40, state: #[Marco state X: 15.45m, Y: 2.76m, TH: 240.0]
-- Had a from goal

&& Starting go to pos, dad sequence it
&& Starting behaviour Reach Position
-- Current behaviour cluster:
.....Avoid Obstacle
.....Reach Position
.....Keep Moving
.....Recover Stall
```

Although a *from* position goal causes the same activation of goal reaching activities, the sequences of actions that follow are different. The wall edge following activity needs to have an end point, indicated by a *to* goal position, in order to stop. The finishing of the anchoring activity and the initiation of wall following generates the following trace.

```
-- Step: 90, state: #[Marco state X: 16.12m, Y: 2.66m, TH:  7.0]
-- Anchored to a wall edge

&& Deleting go to pos
&& Deleting behaviour Reach Position
-- Current behaviour cluster:
.....Avoid Obstacle
.....Keep Moving
.....Recover Stall

&& Starting hug it, dad sequence it
&& Starting behaviour Follow Edge
-- Current behaviour cluster:
.....Avoid Obstacle
.....Follow Edge
.....Keep Moving
.....Recover Stall
```

At this stage, there are two sequencing tasks on the task queue. One is "sequence it" and the other is the subtask "hug it". At the low level control layer, four behaviours, Avoid Obstacle, Follow Edge, Keep Moving and Recover Stall engage in the room edge troweling operation. The robot's progress is monitored by the subtask "hug it". The robot trowels along the edge of the room from the starting *from* position, (I). This operation is finally stopped by the task "sequence it" when the task "hug it" monitors that the robot has arrived at the end *to* position (J) and sets its state to SUCCESS. The trace below gives the results.

```
-- Step: 100, state: #[Marco state X: 14.10m, Y: 3.47m, TH: 259.6]

&& Deleting hug it
&& Deleting behaviour Follow Edge
-- Current behaviour cluster:
.....Avoid Obstacle
.....Keep Moving
.....Recover Stall
```

The robot has finished its main task at this point. It has trowelled the main area of the room floor and also finished pillars and room edge areas. Now, it is the time to leave. The remaining traces cover the final sequencing activities by the sequencing tasks.

```
-- Step: 150, state: #[Marco state X: 14.10m, Y: 3.47m, TH: 259.6]
-- Found a door

&& Starting go out, dad sequence it
&& Starting behaviour Cross Door
-- Current behaviour cluster:
.....Avoid Obstacle
```

```
.....Cross Door
.....Keep Moving
.....Recover Stall
```

At Step 150, the task "sequence it" checks and finds the exiting door of room F10 and initiates the going out of the door activity at position (J). The subtask "go out" is created and inserted into the task queue and the behaviour Cross Door is invoked in the low level control layer. The robot starts to leave the room under the control of the one task-oriented behaviour and three reactive behaviours. Finally, the robot gets out of the room F10 at position (K). The trace shows.

```
-- Step: 180, state: #[Marco state X: 14.80m, Y: 1.83m, TH: 254.5]

&& Deleting go out
&& Deleting behaviour Cross Door
-- Current behaviour cluster:
.....Avoid Obstacle
.....Keep Moving
.....Recover Stall
```

At the higher level, there is now only one sequencing task "sequence it" left. the control system has finished its task and the sequencing activity is no longer required. The task "sequence it" enters its final step of task net. The trace shows the termination of the task.

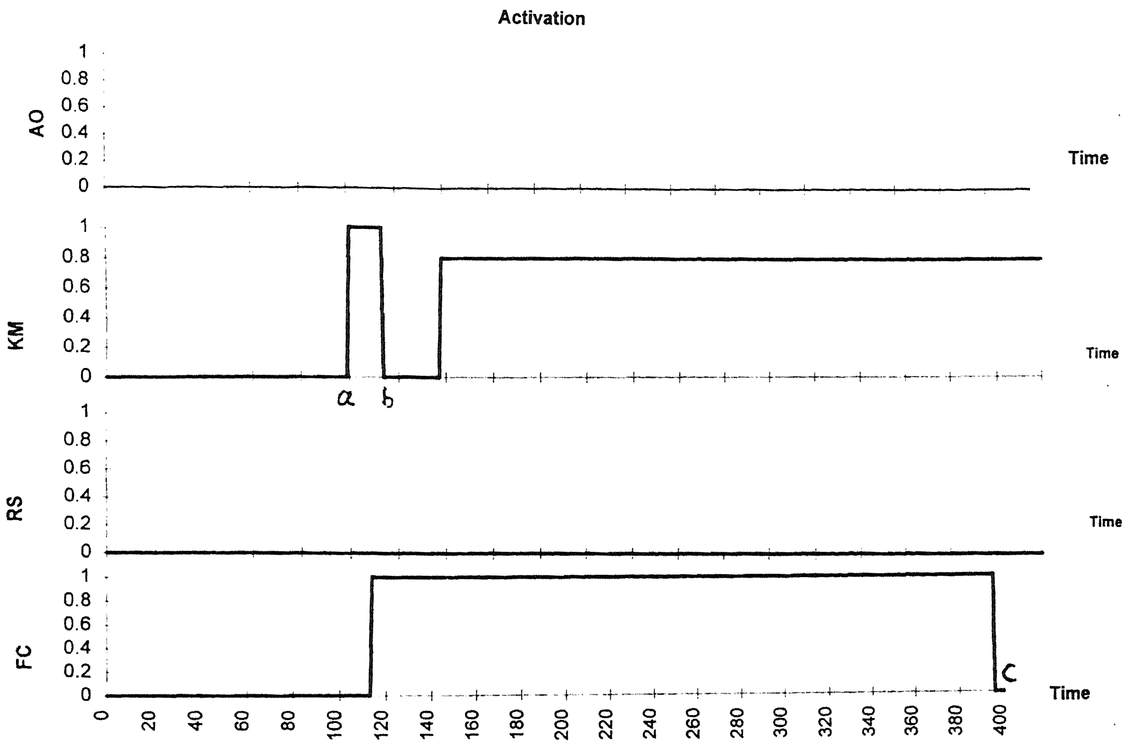
```
-- Step: 200, state: #[Marco state X: 14.80m, Y: 1.81m, TH: 256.0]

&& Task succeeded, sequence it!
&& Deleting sequence it
-- Current behaviour cluster:
.....Avoid Obstacle
.....Keep Moving
.....Recover Stall
```

The top level sequencing task is removed from the task queue. From position (K), the robot is only controlled by the low level control layer with three reactive behaviours. The robot wanders on.

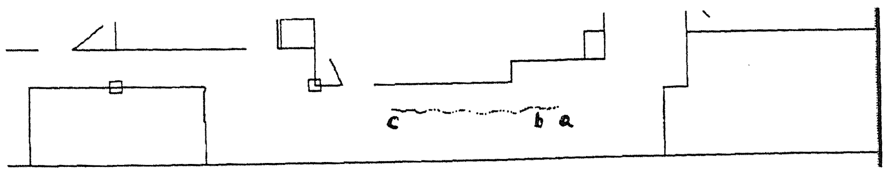
6.2.4.2 A Behaviour Activation Level Trace

In the last section, the task execution of the sequencing layer has been traced. However, the actual robot control is finalised in the low level control layer, which always consists of a cluster of fuzzy behaviours. The detailed trace of these fuzzy behaviours activity will help us to check how the behaviours and behaviour selection network work together to effectively control the robot to complete the given task. It is difficult to record the behaviours' activation graphs. Therefore, the behaviours' activation levels at each cycle of the task execution are first recorded. The activity data vs. time step are then plotted into an activity graph for every fuzzy behaviour. The robot takes about 24850 time steps to finish the whole concrete slab finishing operation. Each step in the simulation measures 0.1s. This section will trace fuzzy behaviours' activities for the most parts of the robot operation as described in the trace of the sequencing activities. This makes it easy to cross-examine the actions in the two layers. Some of typical activities pattern of the behaviours will also be discussed. The traces presented here are plotted with 20 steps, i.e. 2s scale. To distinguish from the sequencing trace, a list of lower case characters, (a, b, c, d, e, f, g, h, i, j, k, l), are used as approximate reference positions in graphs and nearby figures. The sketched robot routes in Fig. 6-7 can also be referred. The first trace starts from the robot's initial position (a), when it is only controlled by three reactive behaviours, and ends when the door of room F10 is found in the position (c) and Follow Corridor behaviour is terminated. This is indicated in Fig. 6-8.

The robot started from almost the central line of corridor #2 facing the other end of the corridor. No obstacles present nearby and the walls at the two sides are quite far. Avoid Obstacle and Recover Stall behaviours will not be active under such circumstances. The robot is started only by the Keep Moving behaviour. The trace graph shows that between positions (a) and (b), only the Keep Moving behaviour is active and has a full activation strength 1.0. The other two reactive behaviours, though running, play no part in the control. After travelling a short distance, corridor #2 is sensed and Follow Corridor behaviour is brought up immediately. This task-oriented behaviour has a promotion/inhibition link to Keep Moving behaviour, which



(a)

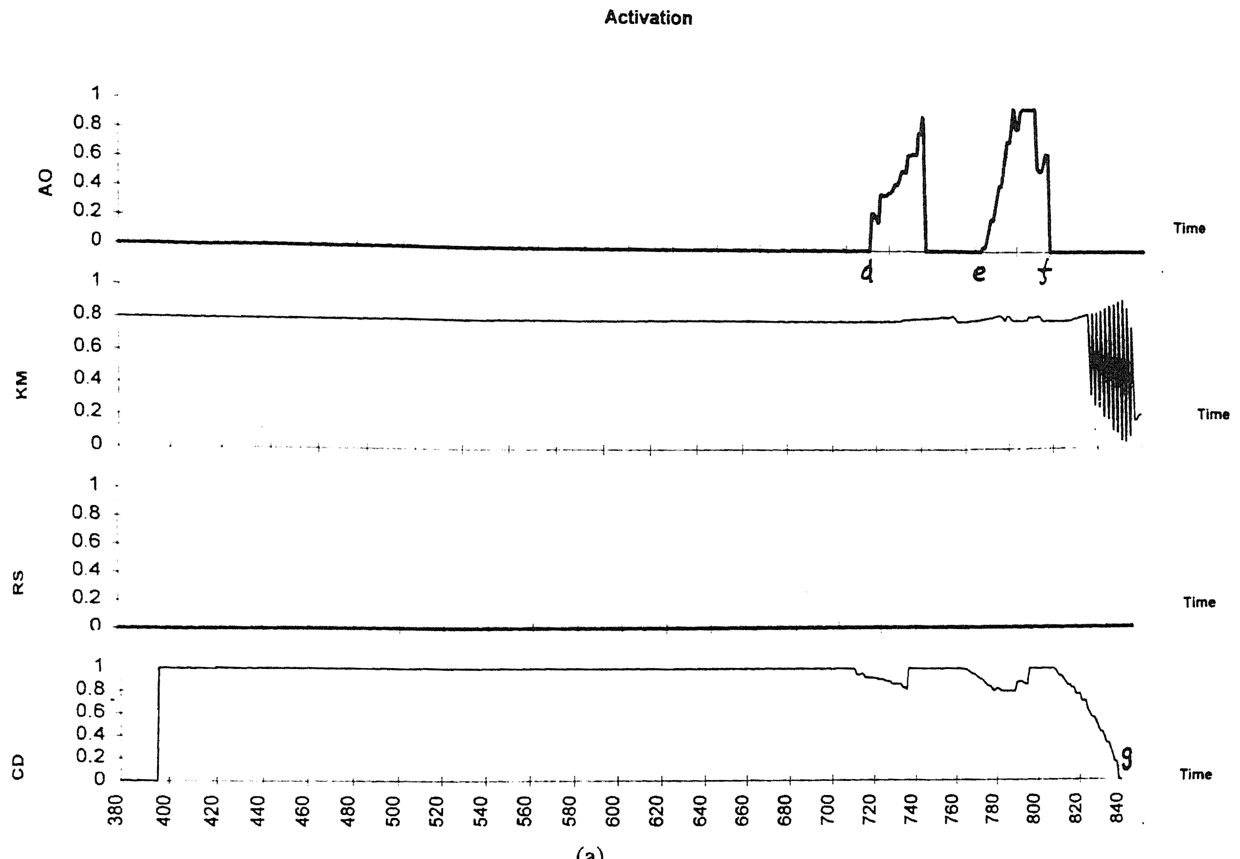


(b)

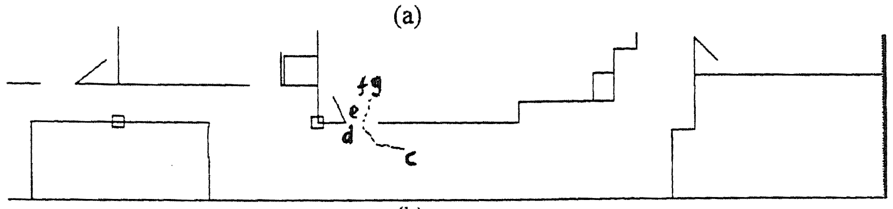
Fig. 6-8 The Trace of Fuzzy Behaviour Activity for Following Corridor
(a) activation graph; (b) course and positions.

at this time, subdues Keep Moving behaviour and takes over the control of the robot from the position (b). The other two reactive behaviours still remains silent. The robot is controlled by Follow Corridor behaviour until it is terminated at the position (c). During this part of the robot movement, Keep Moving behaviour is suppressed at an activation level around 0.8 while Follow Corridor Behaviour exerts the full activity strength. The trace continues in Fig. 6-9.

From the position (c), the robot begins to perform room entering activities, controlled by Cross Door behaviour. Things go smoothly until the robot is guided to the position (d), where Avoid Obstacle behaviour detects that the robot is too close to the obstacle, the door edge at the left. The activation level of the behaviour increases, which also results in the decrease of the other behaviours' activation energy. Finally, Avoid Obstacle behaviour takes over the control and slows down the robot and turns it to the right. The danger of colliding at the left side of the door diminishes and Avoid Obstacle behaviour's activation level decreases as the robot turns away. The interactions between the two behaviours also causes some fluctuation in Keep Moving behaviour's activation level. However, it is not enough for the behaviour to dominate. Cross Door behaviour regains the control and the robot continues to cross the door. When the robot moves near the position (e), a similar thing happens at the right side, which can be seen in the plot between (e) and (f). After the robot turns back to the right course, the entering room activities are near completion. From then, the robot continues moving to the end of the door way and Cross Door behaviour weakens as the end position draws near. At the position (g), the door entering activity is finished and Cross Door behaviour is removed. The robot is temporarily controlled by Keep Moving behaviour. From the trace, we can see the door entering activities are mainly controlled by Cross Door behaviour, while Avoid Obstacle assists to keep the robot safe. Through the energy redistribution between the behaviours, the robot is always controlled by the most favourable behaviour and manages to safely and effectively go through the door, even though an individual behaviour, Cross Door, in this case, may not perform perfectly well and the position of the door was provided approximately. This is the advantage of a behaviour-based architecture. From the



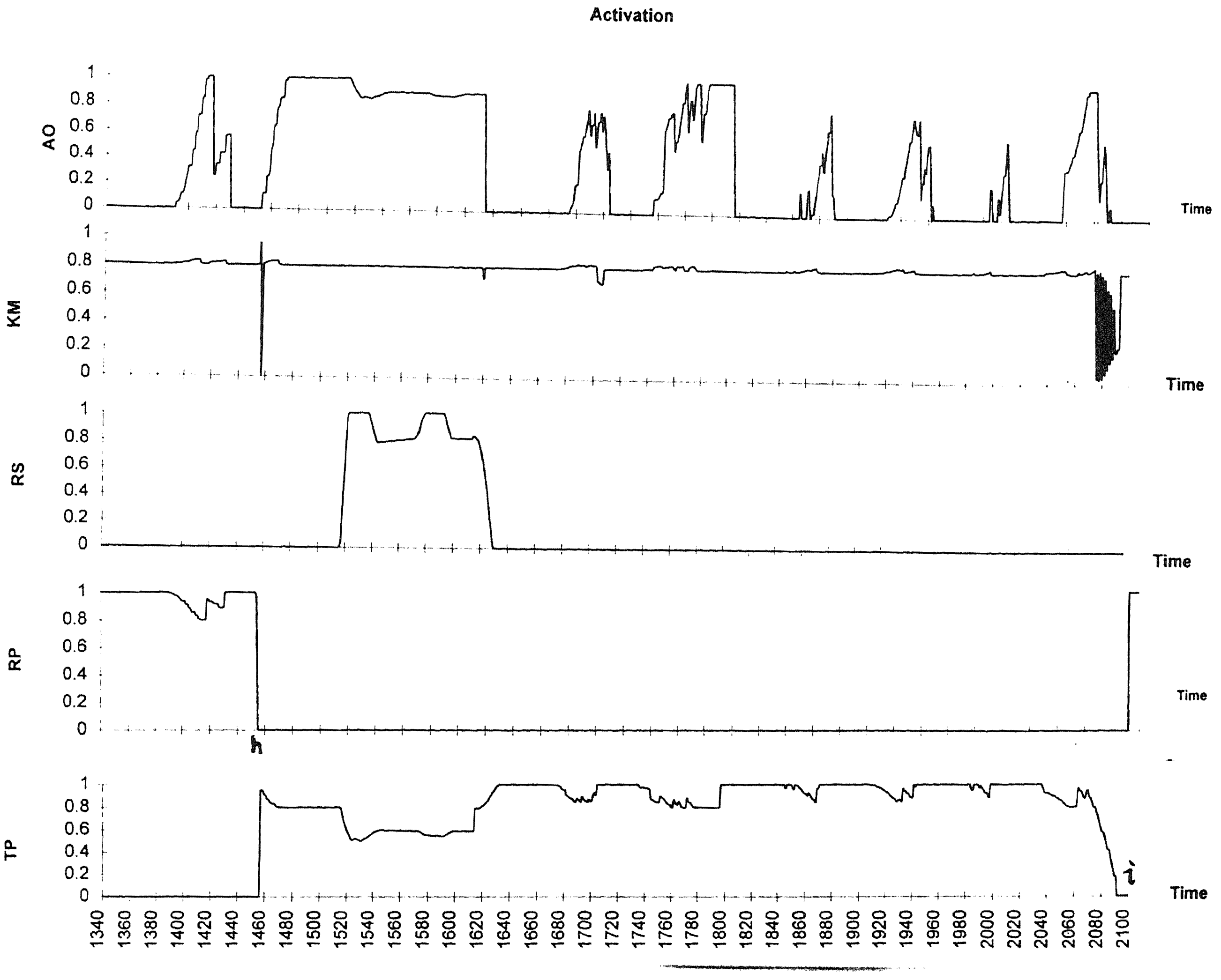
(a)



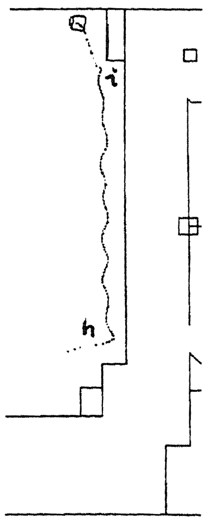
(b)

Fig. 6-9 The Trace of Fuzzy Behaviours Activity for Crossing the Door
(a) Activation Graph;
(b) Robot Course.

position (g), the robot carries out the first goal of the task, going to the position (h) before starting troweling the floor. There is no obstruction between (g) and (h) and the robot moves straight towards the position, only controlled by Reach Position behaviour. The most part of the trace for this period is skipped and the trace continues when the robot moves close to the position as shown in Fig. 6-10. As the robot approaches the corner position, Avoid Obstacle behaviour gradually builds up its activation energy. Just before the arrival, the robot moves close to the convex corner of the wall at the right. This causes further increase of the activation level of Avoid Obstacle behaviour which then takes over the control from Reach Position behaviour. The robot finally arrives at the position (h), safeguarded by Avoid Obstacle and



(a)



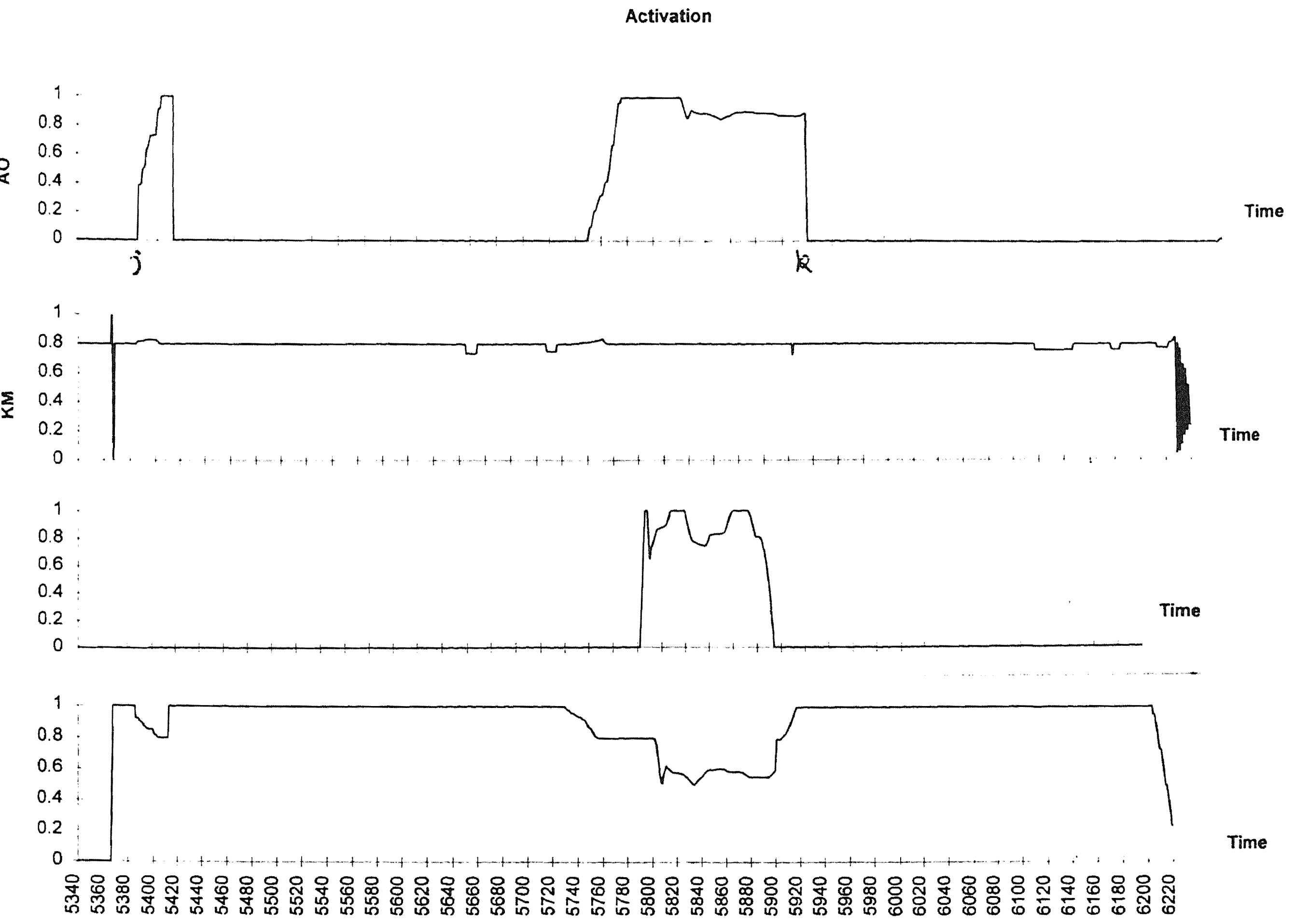
(b)

Fig. 6-10 The Trace of Fuzzy Behaviour Activity for Reaching the First Goal.
(a) activation graph;
(b) robot course

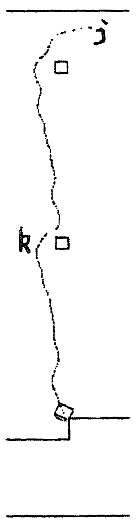
directed by Reach Position behaviour. The traces that follow show the behaviour activities in finishing the first troweling path. The operation is performed by Track Path behaviour and three reactive behaviours. Right after gaining the position (h), the robot is in immediate danger of collision with the wall because its initial heading is towards the wall and is being turned towards the troweling path by Track Path behaviour. The danger is released by Avoid Obstacle behaviour. However, this sequence of actions results in a very slow speed of the robot movement, causing the firing of Recover Stall behaviour to pull the robot out of the unfavourable state. After two such triggering actions, the robot is back to the course and starts troweling the path. The first path is finished at the position (i), though there are still occasions when Avoid Obstacle behaviour shows up to release the danger of collisions.

Tracking an open path is much easy than a blocked one. The trace skips over several troweling operations in the open area and proceeds to show the behaviours' activities in troweling a path with two pillars in the way. Fig. 6-11 presents the trace starting from the position (j). After it leaves the position (j), the robot approaches the first pillar, which is not shown in the provided feature map. Troweling action is immediately replaced by collision avoidance actions, as shown in the traces of two behaviours' activation levels. The collision avoidance action causes the robot to drift away from the troweling path. Passing by the first pillar, the robot is again controlled by Track Path behaviour, trying to steer back to the course. However, because there is no specific positioning control involved, the resulting path is not as desired. The robot keeps moving and approaches the second pillar. Its approaching angle causes the robot to spend more time in avoiding collision with the pillar than the last one. It also brings up Recover Stall behaviour several times to trigger the robot out of stalling states. In the meantime, Track Path behaviour and Keep Moving behaviour are subdued. The troweling actions are resumed after the robot escaped from the pillar in the approximate position (k).

The final trace shows how Follow Edge behaviour and the three reactive behaviours co-operate to finish the first pillar area, starting from the position (l). Follow Edge behaviour takes over the robot control after the robot is anchored to the position (l).

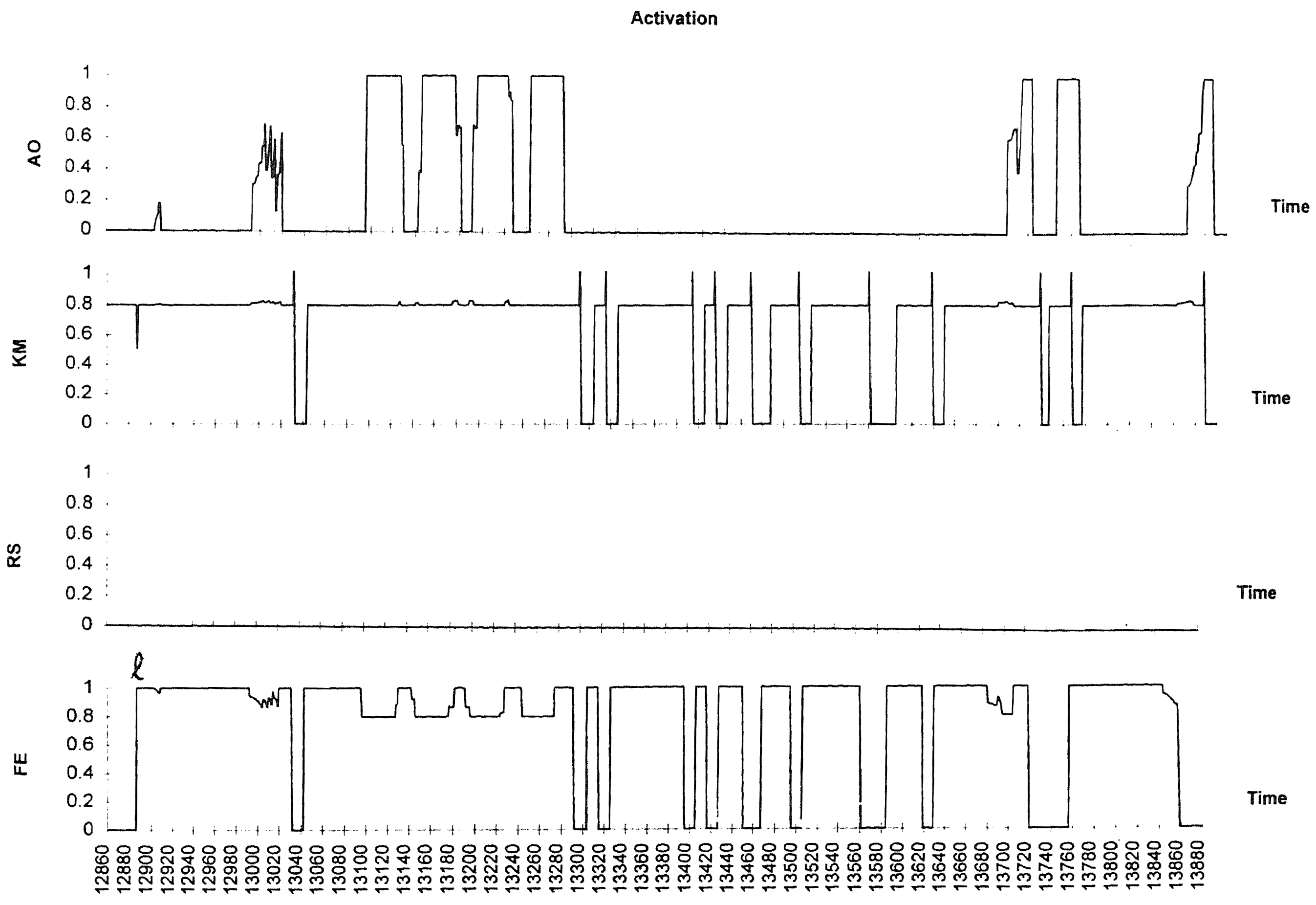


(a)

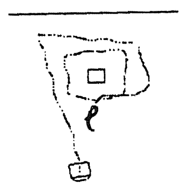


(b)

Fig. 6-11 The Trace of Behaviour Activity for
Troweling a Path with Two Pillars
(a) activation graph;
(b) robot course.



(a)



(b)

Fig. 6-12 The Trace of Behaviour Activity in Finishing the First Pillar
(a) activation graph;
(b) robot course.

The behaviour is required to control the robot operation for 1000 time steps in finishing the pillar area as close as possible. In doing so, it requires Avoid Obstacle behaviour's support for safety when the robot gets too close. Keep Moving behaviour also comes up from time to time to increase the speed when the robot moves slowly, but not too slow to trigger Recover Stall behaviour. From Fig. 6-12, we can see that the robot trowels around the pillar about twice and finishes the work reasonably well under the behaviours' control. This special type of operation is also used in finishing the second pillar and all of the inside edge of the room. These traces of the behaviour's activity are skipped because they are similar to the previous one. So were the traces of the robot leaving the room.

6.2.5 Two Complete Robot Courses

The last sections have presented insight into the activities of a MARCO control system in completing a simulated concrete slab finishing task. The low level control layer of the system is developed based on the manually designed fuzzy behaviours. The parts of the robot course have been displayed in the discussion. The complete snapshot of the robot's movement course is presented in Fig. 6-13. With the same sequencing layer and other components, another control system was developed, using a learned low level control layer consisting learned fuzzy behaviours and behaviour selection network. The navigation and the task execution plan are the same as the last one. The task execution is sequenced in the exactly the same way as the previous one.

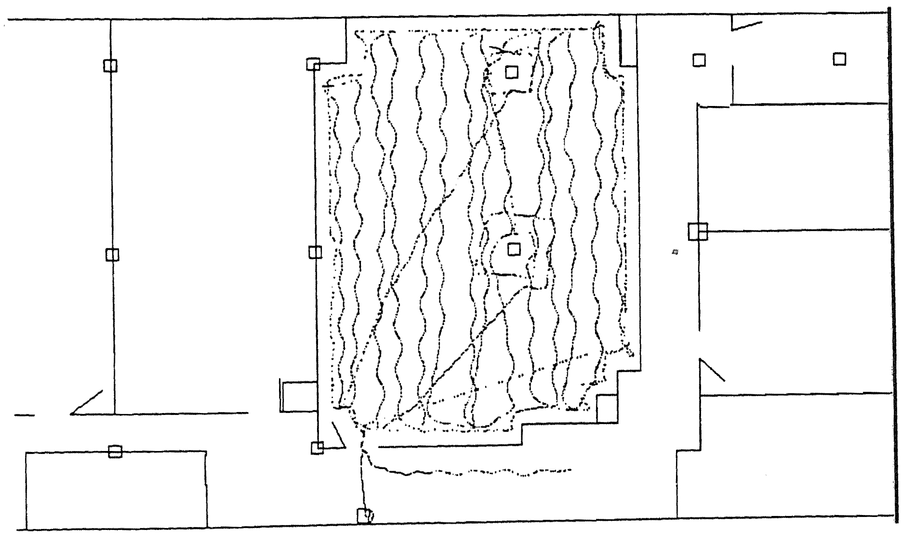


Fig. 6-13 Snapshot of Complete Robot Course in Concrete Slab Finishing Task with Manually Designed Low Level Control System.

There are some differences in the low level controlling activities because of the use of learned behaviours and the selection network. Fig. 6-14 provides the complete picture of the robot course in finishing the task in order to show the effects of the learned low level control layer.

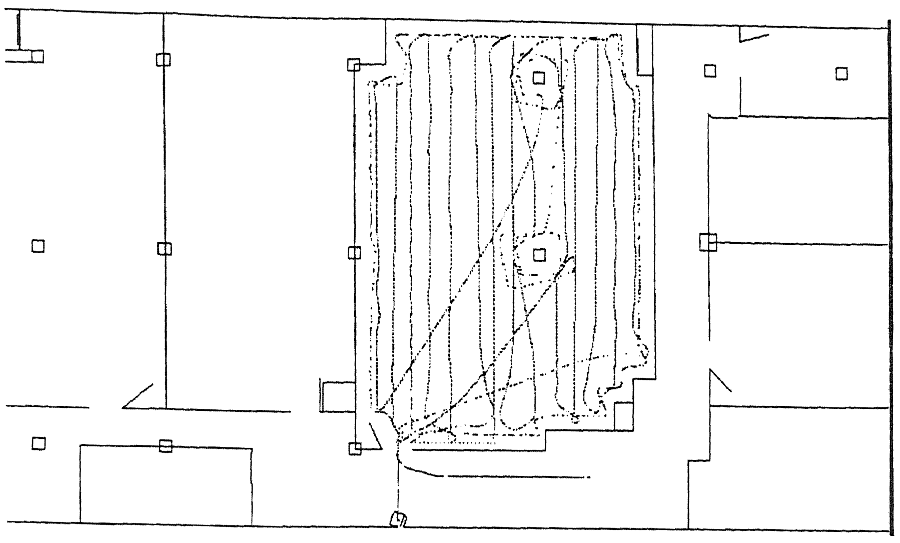


Fig. 6-14 Snapshot of Complete Robot Course in Concrete Slab Finishing Task with Learned Low Level Control System.

6.2.6 Discussion of the Detailed Courses

From Fig. 6-13, we can see that, despite the success in sequencing and controlling, the system presents some problems. The main troweling course is swinging, indicating that the Track Path behaviour cannot stabilise the robot movement and produce a straight troweling path. At the end of a troweling path, the robot is often unable to position itself perfectly at the start position of the next path before further troweling starts, causing a deviation of the troweling actions at the beginning. When approaching an obstacle and escaping afterwards, the robot cannot immediately come back to the path, making the troweling inefficient. In finishing the wall edge, a similar swing to that exhibited by Track Path behaviour also exists, which also prolongs the whole operation of the robot. These problems may be solved in the following ways. First, the undesirable actions caused by the behaviours can be corrected through learning. Fig. 6-14 clearly shows the effects of using a learned low level control layer. The main troweling path becomes straight in the open area with the robot only controlled by Track Path behaviour. Troweling the edge of the wall also goes more stable by Follow Edge behaviour. Swinging has been eliminated from the operations. The whole operation is completed in 21869 time steps, 2981 steps less than the previous one. These facts indicate an significant improvement in the performance by the learned low level control system. The reason is because both fuzzy behaviours and the behaviour selection network used in the low level control layer are obtained through systematic learning processes in much more complex and versatile environments. The learned low level control system is more capable of coping relatively simple and structured environments. The result clearly demonstrates the effectiveness of the simple-to-complex multistage learning methodology.

Second, more detailed planning can be introduced to cope with obstacles during the operation. In Fig. 6-14, we can see that the robot still drifts from the pillar, instead of moving around it. This can be solved by introducing Follow Edge behaviour after the robot detects the pillar. The behaviour can guide the robot to the other side of the pillar and then start the Track Path behaviour again. Another possible way is to plan more subgoal positions and initiate goal reaching behaviours. Finally, more behaviours can be developed to perform versatile tasks. For instance, using the learned system, the control system is still unable to position the robot to the starting position perfectly and some deviation still occurs. However, it is possible to develop a docking behaviour which performs a precise positioning task.

6.3 A Building Security Patrolling Task

In the second experiment, SIMAR is used to patrol the corridors of the first floor of the Spire Research Centre. Instead of tracing the sequencing and behaviours' activity, the actions of the MARCO control system will be traced in the term of physical movement of the robot during the whole operation. The same control system, carrying out the task displayed in Fig. 6-14, will be employed for the patrolling task. The system consists of the same sequencing layer and the learned low level control layer. Unlike this first experiment, noise will be introduced in the robot's movement in this task in order to examine how the perceptual subsystem works to correct and localise the robot position. The following sketch plan is supposed to have been provided by a planning system.

1. go to position A;
2. go to position B;
3. patrol corridor #1 back to position A;
4. go to position C;
5. patrol corridor #2 to the end;
6. go to position D;
7. patrol corridor #3 to the end;
8. follow wall until near the door of F15;
9. enter the room F15.

Information about three corridors is approximate, provided in the feature based map which is shown below, together with precise position data:

```
CORRIDOR(1) -3000,  -500, -90, 12000, 1600  # -2754, -486, -90, 12312, 1620
CORRIDOR(2) -3000,   350,   0, 12500, 2000  # -2916, 364, 0, 12555, 2187
CORRIDOR(3)  7000, -1250,   0,  7500, 1000  # 6723, -1256, 0, 7533, 1053
```

The robot movement is traced through comments, based on the course shown in Fig. 6-15 and other figures.

The trace starts. The robot moves from its initial position and pursues the first goal of the task, going to the position (a). Because its initial heading is opposite to the goal, the robot turns right, controlled by the Reach Position behaviour and the three reactive behaviours, Avoid Obstacle, Keep Moving and Recover Stall. Soon afterward, the robot moves close to the wall at the left, which partly obstructs its direct path towards the goal position. However, Avoid Obstacle and Reach Position behaviour co-operate with each other and the robot clears out the wall and arrives at the position A. Although it is only a short distance, the robot position error has already accumulated. The dead reckoning error and the approximate map result in the disparity between stored and sensed corridor as shown in Fig. 6-16(a). The robot, however, proceeds with the erroneous map maintained in long term model. The robot's perceptual subsystem continues to interpret sensor data. After moving along the corridor for a while, the real corridor is discerned and the perceptual subsystem

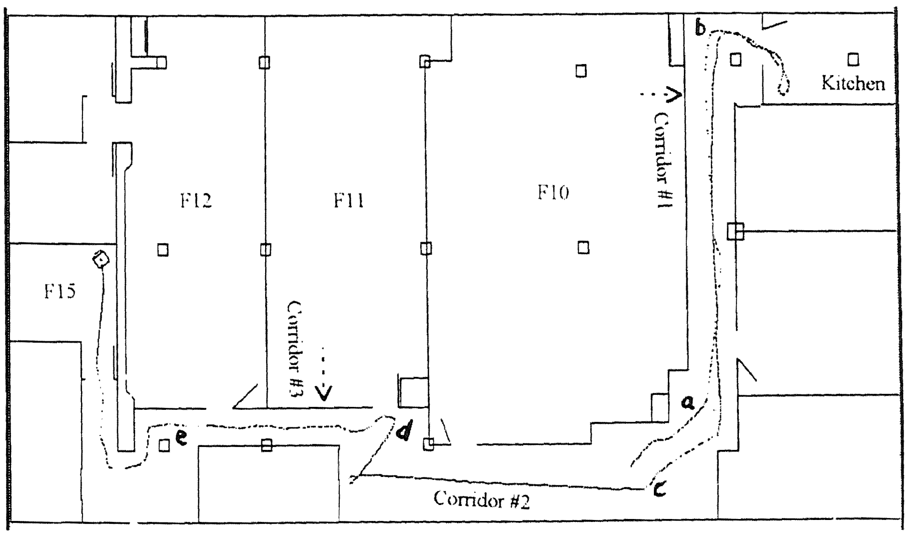


Fig. 6-15 Snapshot of the Robot Course in Corridor Patrolling

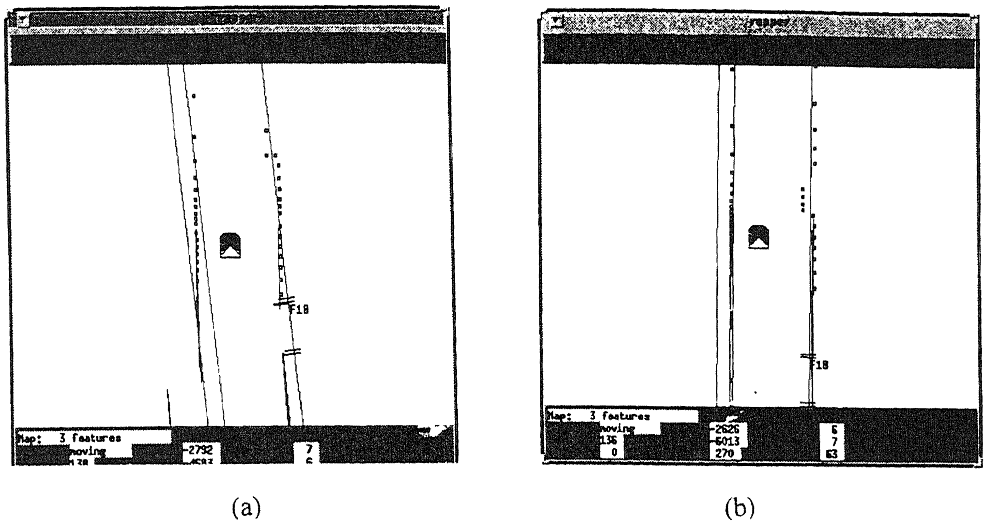


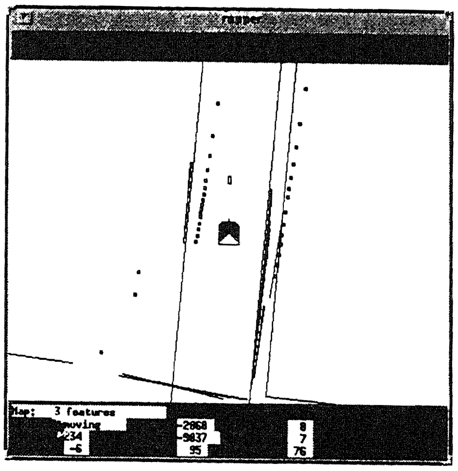
Fig. 6-16 Corridor Perception at the Entry of Corridor #1:
(a) before the corridor is perceived:
(b) after the corridor is perceived.
..... : sensed location;
— : location in map.

performs the following steps:

1. match the found corridor to an existing one;
2. correct MARCO's sensor model;
3. localise the robot position.

The result of the localisation process is shown in Fig. 6-16(b). The robot moves on and the localisation process is repeated. After it arrives in the position (b), the robot should immediately follow back along the same corridor according to the plan. In doing so, the robot gets too close to the upper side wall of the corridor, and slips into the room F19, the kitchen of the 1st floor under the control of Follow Corridor and Avoid Obstacle behaviour. Generally, it will be more desirable that a failure recovery task template be designed and its task be invoked to cope with the situation. Unfortunately, it has not been available for this experiment. The robot still carries on the corridor following task and is gradually dragged back towards the corridor. Interestingly, the robot is able to leave the room and go through the door, under the combined effort of Follow Corridor and Avoid Obstacle behaviour, having not even invoked Cross Door behaviour. After leaving the room, the robot encounters another problem; facing the pillar right in front of the room. It manoeuvres around and finally leaves the obstacle and returns to the correct course. The robot, during this period, especially, exhibits robust goal-directed reactive navigation abilities. It strayed away into a wrong place, survived in the complex environment, came back to the right course and continued with its task. This sequences of activities can be very difficult to sustain if using a traditional control approach.

The trace continues as the robot returns back to follow the corridor #1. The error accumulated when the robot got lost causes the robot to follow an inaccurate corridor as shown in Fig. 6-17(a). However, the robot is able to cope with the inaccuracy under the fuzzy behaviours control and return to correct course when the perceptual subsystem matched the corridor and thus performed localisation. This is indicated in Fig. 6-17(b). The robot then smoothly follows back corridor #1 and travels through the corridor #2. Following corridor #3 is more difficult than the previous ones as it is more narrow. The robot is first guided to the approximate position (d), and starts to be controlled by Follow Corridor Behaviour again when the sequencing layer terminates Reach Position and initiate corridor following activities. Fig. 6-18 shows


(a)

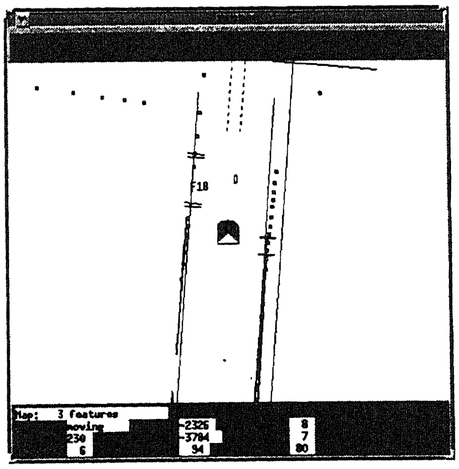
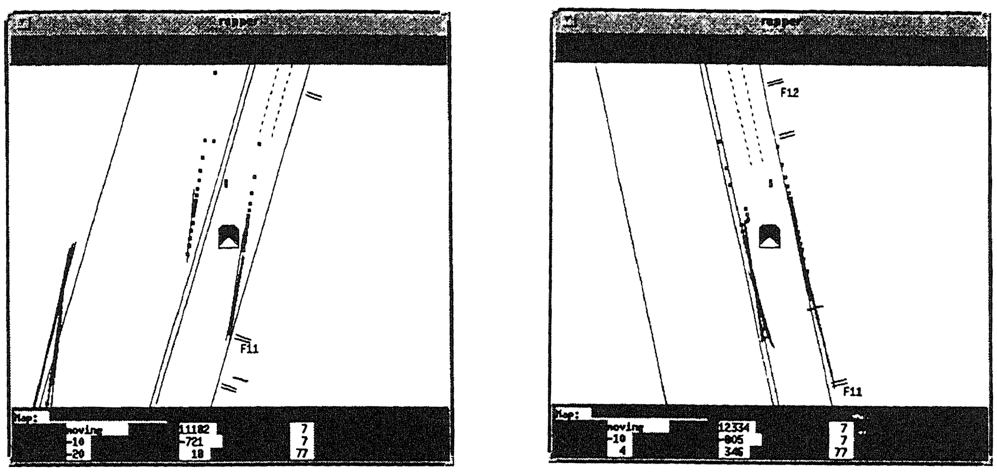

(b)

Fig. 6-17 Corridor Perception at the End of Corridor #1:
(a) before the corridor is perceived;
(b) after the corridor is perceived.
..... : sensed location;
— : location in map.
--- : assumed following lane

the local view of the robot about the stored and sensed corridor location at the beginning of the following activities. From Fig. 6-18(a), we can see the big disparity of the actual corridor and its map which is used by Follow Corridor behaviour. The robot manages to follow the corridor using the wrong map while avoiding collision with the wall which is mainly caused by the wrong map. After a short distance, the real corridor



(a) (b)

Fig. 6-18 Corridor Perception at the Entry of Corridor #3:
(a) before the corridor is perceived;
(b) after the corridor is perceived.
...... : sensed location;
— : location in map.
⚌ : assumed following lane

is perceived and its map is corrected. This is shown in Fig. 6-18(b). The robot is able to follow the corridor from then until it comes across the pillar half embedded in the middle of the left wall. This part of the corridor makes straight line following impossible. The robot negotiates with the pillar at the left and the wall at the right, and squeezes through the narrow path. This pillar is not indicated in the provided simply approximate map. However, with the support of reactive behaviours, the robot is able to cope with such incomplete information and still carries out the task.

The robot successfully finishes all the corridor following tasks at the end of the corridor #3, where Reach Position behaviour is started in order to anchor the robot to the wall edge. There is another pillar right in the front of the robot, which is also unknown in the robot sensor model. The robot has to avoid the obstacle before anchored to the wall edge. From the anchoring position (e), the robot starts to follow the edge of the wall, controlled by Follow Edge behaviour and other reactive behaviours. The robot follows the edge of the wall, turns at the concave corner and continues to follow the adjacent edge, then turns at the convex corner and keeps following. When the perceptual subsystem detects the door of room F15, the robot is relocated. The sequencing layer stops Follow Edge and starts Cross Door behaviour. Finally, the robot enters the room and the patrolling task is terminated.

6.4 Summary of Experiments

This chapter offers evidence that MARCO control architecture is an effective approach to implement execution systems that can control the robot navigation in a complex environment with uncertain and approximate information. The evidence consists of three types of information gathered using actual implementation of two control systems to perform two simulated real world tasks. The first type of evidence presented consists of trace of the sequencing activities, following the execution of a concrete floor slab finishing task. The trace shows the way the sequencing layer tailored the system to the need of the task execution. The second trace shows how the fuzzy behaviours in the low level control layer interact with each other to provide the system with survival and purposeful control abilities. The third type of trace presented consists of the robot movement trail to demonstrate the system's behaviour in coping with unexpected control results and incomplete and noisy information, in performing a building security patrolling task.

The traces show that the MARCO control system can organise efficiently the sequences of the robot activities and carry out robust low level control. The sequencing layer is able to initiate, terminate and monitor the controlling activities to adapt the systems to the requirement of current task execution, while the low level control layer, tailored by the sequencing layer, performs robust goal-directed reactive navigation in the face of imperfect actions, uncertainty, incomplete and noisy information.

The experiments are far from perfection as they do not cover every situations in the concrete slab finishing and building security patrolling domains. Many issues, especially failure recovery and dynamic situations have not been presented in the experiments. Nevertheless, they present the evidence for the validity of MARCO approach, a framework which can be further improved.

Chapter 7 Conclusions and Future Work

This chapter begins with the summary of MARCO control architecture. It will then discuss the major lessons to be learned from the work presented in the thesis and suggest some directions for future research.

7.1 Summary

7.1.1 The Argument

This thesis addressed the problem of how to control an autonomous mobile robot navigation in the real world, mainly indoor environments. There are several major problems which need to be dealt with by a navigation control system: sensor noise, imprecise information, uncertainty and limited response time.

The equivalent analogous example which was used to identify the needs of a robot system facing these problems was driving a car to an unfamiliar destination. To accomplish such a task, a driver must be able to react quickly to the situations on the road, and must mediate these reactions for the purpose of arriving at the destination according to a sketch plan produced by a slow deliberative considering process such as reading a map. The driver must do this in the limited time available resulting from the time constraints of driving. The driver must be able to perform the task despite the fact that many of the situations encountered along the way are not presented in the map and cannot be predicted in advance. All these problems will also be encountered by an autonomous mobile robot, whether in indoor or across country navigation.

Using driving a car as an analogy to controlling a mobile robot, the thesis argued in Chapter 1 that local sensor data and a world model are both needed to provide the immediate local feedback and the necessary global information. Sensor noise should be dealt with in both the robot local movement and in its long term world model. An effective robot control system should also be organised to effectively accommodate imprecise information by incorporating heuristic control which is often exercised in human control behaviours, such as driving a car. The thesis further argued that a robot should not follow a prescribed step-by-step plan trying to postulate all the aspects of a navigation task due to the uncertainties in the real world. Environment contingencies can arise suddenly. They are unpredictable but need to be dealt with quickly. Many aspects of a navigation problem cannot be exactly planned. While a plan is necessary to guide a robot, it has to be sketchy, leaving low level details to be filled when they are encountered. To accommodate such a plan, an effective control system should be best organised into levels, with the high levels providing guidance and the low level realising plan execution and taking care of detailed interactions in real time. The thesis also argued that a behaviour-based control approach should be used in organising the low level control because of the fast response and the simple design of a control system. The thesis further stated that learning should be introduced to help the design and improvement of a control system.

The argument presented in the earlier Chapters provides the intuitions for the development of a two layer control architecture, a task template based sequencing layer and a fuzzy behaviour based low level control layer, as well as a learning methodology. They are summarised in the following sections.

7.1.2 MARCO control architecture

The central topic of this thesis is that a successful mechanism for controlling mobile robots must be organised into a hierarchy. There are two types of activities involved in controlling mobile robots. High level activities contain decision making computational processes which initiate or terminate low level activities. They are used to organise the correct sequences of controlling activities for the purpose of achieving task goals, monitoring and intervening in the task execution. Low level activities are responsible for realising the detailed execution of the task. They contain two types of computational processes: reactive and task-oriented. The reactive controlling activities take care of all interactions with environment contingencies, provide basic functions and guarantee the safety of the robot. The task-oriented controlling activities are to carry out the task currently assigned by the high level activities. MARCO is a two layer architecture to support such high and low level robot control activities. It consists of two layers: a sequencing layer and a low level control layer. The sequencing layer works at the pace of the high level of abstraction, interpreting a task plan, providing high level goals and commands, initiating, mediating and monitoring the controlling activities, while the low level control layer performs fast computation at the pace of changes in the real world and controls direct physical actions to finish a task given by the high level.

7.1.3 Fuzzy Behaviour-based Low Level Control Layer

The organising of MARCO control architecture was approached bottom-up, beginning with the low level control layer. Low level controlling activities contain simple decision-making computation and are required to have a fast response time. A computational mechanism for controlling such activities should also be able to work properly in the face of sensor noise, uncertainty and imprecision and take advantage of heuristic knowledge of such a control process. In order to support the development of such control structure, the notions of a behaviour and fuzzy logic control approach are employed to implement the basic control entities of the low level control layer, fuzzy behaviours. MARCO's fuzzy behaviours have the following features that make them suitable for controlling the low level activities:

- Behaviours are organised based on the sphere of influence of environment features, which lend themselves to directly interact with environments;
- A fuzzy logic controller allows the accommodation of sensor noise, approximate and imprecise information, as well as the easy introduction of heuristic control knowledge into the robot control;
- A fuzzy singleton representation of output allows fast computation for the output of a fuzzy behaviour;
- A soft channel structure allows the direct communications of a behaviour with the higher level systems;
- A behaviour link structure allows the energy redistribution between behaviours for

the effective control of the robot.

Fuzzy behaviours must be fused or selected to produce a single set of outputs for the robot control. A dynamic behaviour selection network was developed, inspired by Maes's approach[Maes90]. Fuzzy behaviours are connected through the network based on promotion and inhibition links. The selection network provides the dynamic support for the motivations of the control system under the current environment states and task execution conditions. The most suitable behaviour is always selected through activation energy redistribution among the behaviours by the network to control the robot. The behaviour selection network has the following features which makes it different from other methods and suitable for the MARCO low level control:

- Fuzzy predicates-based behaviour activation level allows the smooth flow of activation energy between behaviours and results in smooth transition of behaviour control;
- Promotion and inhibition links are set-up according to the motivations of the control system;
- Open structure of the selection network allows the easy introduction of error recovery behaviours and also direct control of the robot.

An experimental low level control layer, consisting of several fuzzy behaviours and a behaviour selection network, was implemented for indoor navigation tasks [Qiu96a] [Qiu96b].

7.1.4 Task Template-based Sequencing Layer

Fuzzy behaviours are simple computation structures and are not suitable for controlling sequencing activities since these activities involve the dealing of temporal and other constraints which are usually not fuzzy. To control the higher level activities, a different control structure, called a sequencing layer was developed. The main tasks of the sequencing layer are the initiation, monitoring and termination of controlling activities realised in the low level control layer. The sequencing layer is mainly based on Firby's Reactive Action Package[Firby89] system, and is modified to support simultaneous execution of tasks. The basic block of the sequencing layer is the control structure called task template, an extended RAP. The following points make task template more suitable to control MARCO's higher level activities:

- A task template contains only one method and eliminates method selection during task execution;
- The task net in a method consists of both task templates and ordinary functions to reduce amount of tasks in the task queue;
- Tasks are executed equally by a task scheduler to eliminate task selection overhead and support concurrent execution of fuzzy behaviours at the low level;
- Each task is assigned a state which can represent a step or pre-defined states for effective scheduling and processing.

These modifications result from the different applications of RAP and MARCO. RAP systems are mainly intended to control discrete actions involving many objects manipulation. MARCO is mainly used to control mobile robot navigation from place to place.

7.1.5 Learning of Low Level Control Layer

Although a fuzzy behaviour-based control system is easier to develop and more robust than a traditional sense-model-plan-act system, it can be difficult to configure such a system to obtain optimal control behaviours. The problem has two aspects: to obtain optimal individual fuzzy behaviours and to configure the behaviour selection network. The performance of individual fuzzy behaviours relies on an optimal set of membership functions for their fuzzy control variables. Manual tuning of fuzzy membership functions is time-consuming and cannot guarantee optimal solutions. Adjusting one fuzzy rule may interfere with other rules. Parameters tuned to work well in one environment may have adverse effects in other environment. The manual trial and error methods are only suited to very simple behaviours and do not lend themselves well to compose more complicate behaviours. On the other hand, individual fuzzy behaviours must be fused or combined to produce single set of control output. How to always choose the best actions under various circumstances is unlikely to be solved well using a manual tuning approach.

A learning methodology was developed to address the problem of learning an optimal low level control layer of MARCO architecture. This methodology contains the following principles which guide the automatic learning process to produce general and real useful results:

- A learning process is started from scratch;
- Learning emphasises the functionality of individual components, either behaviours or subnets of behaviour selection network;
- Learning environments are generalised;
- A simple-to-complex multistage learning course is followed.

Learning algorithms were developed for the automatic learning of individual fuzzy behaviours and the behaviour selection network of the MARCO low level control layer, using the learning principles. Genetic algorithms were used as population search methods and were designed to enable an efficient exploration and exploitation of the search population for optimal solutions. The learning algorithms were used to learn several fuzzy behaviours and the behaviour selection network in the experimental implementation of MARCO's low level control layer. The results demonstrated the effectiveness of the learning methodology[Qiu97a][Qiu97b].

7.1.6 Experiments

MARCO was used to control SIMAR, a simulated indoor robot, performing two tasks. One is a simulated concrete floor slab finishing task and the other is a building security patrolling task. Three types of traces were recorded to demonstrate the effectiveness of the MARCO architecture. The first type of trace showed the task execution of the sequencing layer. The detailed trace demonstrated that the task template-based sequencing layer can be used to effectively organise the control activities in the low level control layer, performing a fairly complex concrete slab finishing task in a simulated real world. It further demonstrated a layered control architecture where the higher layer provided guidance to low layer which operated at different paces of computation. The second type of trace presented the controlling activities of the fuzzy behaviour-based low level control layer. The trace showed how the fuzzy behaviours competed and co-operated to complete the task assigned by the higher layer while the robot survived in the environment. The effectiveness of the low level control layer was demonstrated by the robust individual fuzzy behaviours and their selection network. It also demonstrated that a learned low level control layer performed significantly better than a manually designed one in the concrete slab finishing operation. An example of the fuzzy behaviour's ability to deal with approximate information was also demonstrated. The third type of trace, SIMAR's movement trail, was used to demonstrate the overall abilities of a MARCO control system in performing another task in a complex environment. It showed that the robot was able to work robustly under sensor noise, incomplete information and uncertainty. On one occasion, the robot got lost from the desired course to an unfamiliar area, survived and came back to carry on its task. The trace also demonstrated a rudimentary example of MARCO's perceptual subsystem performing sensing, matching and localisation.

7.2 Evaluation

7.2.1 Robust Goal-directed Behaviours

A robust control architecture must enable the robot to complete a given task in the real world, under sensor noise, uncertainty and imprecision. The robot must be involved in the two types of activities: take decisions and execute actions. These two types of controlling activities need to be performed at different time scales to adequately cope with the robot operation in the real world. The solution to the dual need for taking decisions and executing actions is to adopt a two level model, MARCO: the higher level decides the correct sequences of task goals to be achieved, based on the available knowledge; the lower level achieves these goals while dealing with the environmental contingencies. The robot operation is goal-directed by the higher level toward the accomplishment of a task. Robust control behaviours are realised by the lower level, implemented by fuzzy behaviours exploiting the flexibility of fuzzy logic for dealing with the imprecision and errors in the prior knowledge, in the sensed information , and in the robot's movement. Purposeful task-oriented behaviours and innate reactive behaviours are combined by the behaviour selection network into goal-directed actions.

These abilities have been exhibited by SIMAR, a simulated robot with a MARCO control system in two experiments. In the concrete slab finishing task described in Chapter 6, SIMAR strayed away several times from the starting position of a new troweling path when moving close to the edge of the wall. However, it always came back to perform the troweling actions. One of the most difficulty parts in the operation was to trowel a path with two pillars in it. The system's world model did not include the pillar features. However, SIMAR was able to avoid the two pillars and still move along the presumed track under the control of a task-oriented behaviour and three reactive behaviours. The most interesting example was in the second task shown in Fig. 6-14 when SIMAR patrolled the corridor and got into the kitchen of the first floor unplanned. It is a difficult task to get out without a reactive planning or recovery process. SIMAR exhibited robust goal-directed control behaviours during escaping. In the low level control layer, the three reactive behaviours, Avoid Obstacle, Keep Moving and Recover Stall took care of the robust survival control, while Follow Corridor behaviour, initiated by the sequencing layer, directed the robot out of the room and back to the right track in the face of sensor noise, uncertainty and incomplete information. Although MARCO was only tested in two tasks, the realistic and rich types of the environment conditions illustrated the clear validity of this performance.

7.2.2 Fast Response Time

The computation power of a mobile robot is always limited. Yet, a mobile robot is still required to respond quickly in the real world. MARCO deals with this issue by separating time critical controlling activities, such as avoiding collision, from non critical ones. The low level control layer, which is the most important for the functioning of the robot, is developed using fuzzy behaviours. These fuzzy behaviours produce fast computation through fuzzy reasoning process based on singleton representation. The sequencing layer is also structured in a way which makes it efficient to execute tasks. By constructing steps of task net using both task templates and functions, the amount of tasks in a task queue is significantly reduced.

The testing of this performance is best carried out in real experiments. Unfortunately this has not been available due to the amount of the work and the lack of the time. However, the fast response performance has been observed from the simulated experiments. SIMAR's control system can finish all the tasks in the task queue in less than 100ms. The robot was able to quickly move away from obstacles to avoid collision, which is the most important of the fast response behaviours. In a real implementation, the number of the tasks in a MARCO control system will be expected to be reduced and some time-consuming tasks, such as perception tasks can be carried out through parallel processing.

7.2.3 Uncertainty, Sensor Noise and Imprecision

Uncertainty arises in many different ways. Some aspects of the environments cannot be predicted because the information required is not available. Even though many can be predicted, predictions are mostly mingled with uncertainty. Prior knowledge can be incomplete and approximate. Sensed information is not always accurate. Errors accumulate in the robot movement. These are the realities a mobile robot control system must deal with. In MARCO, these problems are approached from several aspects. The prediction or task plans made by a planning system or human do not control the robot directly, but function as input to the sequencing and control layer which actually control the robot. This is adopted from plan guided reaction theory, advocated by [Payton90]. In this case, a plan consists of a sequences of task goals, such as, a position to reach or environment features to interact with. These goals are dispatched by the sequencing layer through the invocation of low level control activities, not pursued directly by the high level of systems. MARCO also employs a sensor model consisting of a local sensor model and a long term model, which is shared by the whole system. The sensor model does not abstract all the details of the environment, but holds both little interpreted data and environment features provided by a map and a perceptual subsystem. This perceptual information can be approximate and gradually corrected by the perceptual processing. The sensor model allows more perceptual processing power to be integrated in order to facilitate accurate modelling of the world and reduce the uncertainty and imprecision. In the low level control layer, fuzzy behaviours constitute noise-tolerable computation modules, using the elasticity of fuzzy control rules to reduce the adverse effects caused by sensor noise and approximate information. The open structure of the behaviour selection network also allows the introduction of error recovery behaviours and other direct control of the robot when things go wrong. Finally, MARCO's sequencing layer has the abilities to allow and recover task execution failure by introducing more error monitoring and recovery task templates.

These abilities have also been demonstrated in SIMAR's experiments, especially in the second task. SIMAR's motion system was inaccurate and accumulated position errors over time. The feature map provided was approximate. Several pillars in the corridor were not indicated. Yet, the robot was able to successfully deal with these problems and finish the task. Its perceptual subsystem demonstrated the abilities to find and extract environment features which were, in turn, used to correct the approximate map and localise the robot. The robot, under all this erroneous information, was still able to robustly follow the corridors and avoid the collisions with unexpected obstacles. Its ability to cope with uncertainty was especially shown in escaping from the kitchen after becoming accidentally trapped.

7.2.4 Limitations

There are four major limitations in the MARCO architecture which need to be addressed in the future research. First, the structure between the perceptual subsystem and MARCO has not been clearly defined. In this thesis, the perceptual subsystem is independent of MARCO's two layers. This is because the subsystem performs only routine perception tasks. Such routine processing will be inadequate if planning is introduced. How to organise the structure of the perceptual subsystem will affect the performance of a MARCO system substantially, especially in the real time response of the control system.

Second, an error recovery mechanism has not been completely provided. One of the important characteristics of a robot control system is the ability to recover from errors. MARCO has currently presented an incomplete solution through the introduction of error recovery behaviour at the low level control, such as Recover Stall behaviour. Some simple execution failures can also be partly solved by the sequencing layer using error recovery task templates, such as for escaping from the kitchen. However, these are inadequate to cope with execution failures in more complicate circumstances. Planning has to be involved to deal with these situations, together with direct control at the low level control layer. The interruption mechanism needs to be set-up to allow such emergent tasks to be processed first.

Third, a deliberative layer has not been provided. MARCO's sequencing layer is currently only responsible for organising the correct order of a task execution at a higher level. A deliberative layer needs to be introduced to provide and modify a navigation plan during a task execution. The relationship between the deliberative and the sequencing layer should be clearly defined. Some possible solutions will be presented later in this Chapter.

Finally, MARCO architecture has not considered manipulation tasks. It is inevitable that manipulation will also be involved even in a corridor navigation task. In the first floor of the Spire research centre, the doors in the corridors are closed most of time. Marco will need arms to open them and get through. Although this type of manipulation will be very different from those of Firby's, it will increase the complexity of a task execution dramatically. MARCO has provided basic control structures for dealing with such operations. However, the impact of such tasks on the architecture has not been investigated thoroughly. One thing is certain: a MARCO control system must include object recognition to perform such tasks, which will inevitably require vision system and parallel processing mechanisms.

7.3 Discussion

The main contributions of this thesis are as follows:

- a two layer control architecture, MARCO, with a task template-based sequencing layer and a fuzzy behaviour-based low level control layer;
- a promotion/inhibition network-based behaviour selection approach;
- a simple-to-complex multistage learning approach for learning the low level control layer.

It also contributes to the methods implemented to organise fuzzy behaviours, based on the sphere of the influence of environment features and to design a task template.

There are many famous robot control architectures developed and reported in the literature. Three layer architecture, deliberative / sequence / reactive control, seems to be the current state of art. MARCO resides as the two lower layers. Combining a RAP-like sequencing layer and fuzzy behaviour-based low level control layer has not been reported in the literature. Comparisons with some common control architectures are necessary to shed some light on MARCO's unique points.

MARCO is an extension of existing methodologies and technologies. The subsumption architecture of Brooks and his students[Brooks89][Connell89] provide the methodology to develop a behaviour-based control system. In part this has been borrowed in developing MARCO, especially the subsumption idea of decomposing complex task into simple behaviours. In part MARCO is in conflict with the essence of the methodology, in preferring also to incorporate a model-based representation of the world as part of the control system, together with simple direct sensor data. Without such a representation, it is difficult to co-ordinate reactive and purposeful behaviours in a general way to complete complex tasks. The feature-based representation plays an important role to bridge our fuzzy behaviours to direct interactions with environments.

MARCO's sequencing layer is essentially a RAP system and its sensor model is also heavily influenced by the RAP memory model. However, there are some important differences. MARCO's task template is an extended RAP in the sense that it helps reduce the amount of tasks in the task queue substantially, eliminate the need for method selection and task selection. A task template is used to control the concurrent execution of fuzzy behaviours, while the RAP is used to control discrete actions. These differences have resulted from the different purpose of applications. RAPs are developed for dealing with applications involving many object manipulations. MARCO is mainly intended for mobile robot indoor navigation.

MARCO also bears some resemblance to the low layers of ATLANTIS[Gat92], a three layer architecture, in using a RAP as sequencing layer. However, its sequencing layer is strictly a RAP system, which controls the subsumption-like "circuit" behaviours in the low level layer.

MARCO's sequencing layer is also similar to TCA[Simmons90], in that TCA allows steps in a task net to include ordinary computation and also physical tasks. The principle difference is that TCA follows the traditional sense-model-plan-act approach and then uses tasks to add concurrence. TCA is mainly a sequencing layer and does not specify the structure of a low level control mechanism. TCA has complete facilities to distribute processes through message passing.

The main difference between MARCO and AuRA[Arkin90] is in the way the low level control is implemented. MARCO uses fuzzy behaviours and AuRA uses potential field based motor schema. AuRA uses a planner to control the motor schema while MARCO employs the sequencing layer to arrange the low level activities. AuRA does not possess a typical layer structure in three layer architecture's terms. MARCO is less committed to a complete, accurate world model than AuRA does.

Like RAP, TCA and ATLANTIS, MARCO's sequencing layer is also similar to PRS [Georgeff87]. PRS can be flexibly specified for different tasks in different domains using meta-KAs and is therefore less committed to the robot control than a RAP-like sequencing layer[Firby89]. It does not specify the low level control structure.

MARCO's low level control layer is similar to the FLAKEY architecture[Saffiotti et al 93a]. Like FLAKEY, MARCO uses fuzzy logic control to implement fuzzy behaviours. The main difference is that FLAKEY uses a blended-behaviour approach to fuse behaviour and MARCO employs a behaviour selection network and synthesises behaviours through activation energy redistribution. This approach allows more effective transition of the robot control and error recovery and also unifies the method for introducing direct control to deal with failures in the low level layer. Moreover, FLAKEY is a single low level control layer and MARCO has a higher sequencing layer.

Different from all these architectures, MARCO was developed along with a learning methodology for systematically learning a low level control layer.

The above discussions highlight the main differences and also the similarities between MARCO and several main robot control architectures. Many ideas have been inspired by these works. MARCO is also influenced by other fuzzy behaviour-based control methods[Goodridge94][Garcia-Alegre93][Reignier94] and control architectures [Nilsson94] [Schoppers87][Kaelbling88]. MARCO is intended to contribute as an extension or implementation, especially conjoining a RAP-like sequencing layer and a fuzzy behaviour-based low level control layer.

MARCO has been implemented in a simulated robot, SIMAR, which has demonstrated the navigation and control abilities in performing two tasks. SIMAR's counterpart, Marco, a real robot, was originally intended to be a concrete slab finishing robot, which should autonomously finish not only a rectangular floor but also the edge of walls, pillars and corners. These abilities have been realised in SIMAR through the introduction of Track Path, Reach Position and Follow Edge fuzzy behaviours. Compared to other similar robots[Arai89][Thau97], SIMAR showed improved navigation and control abilities in that those robots are teleoperated in operating at wall edges, corners and pillars. Although the simulation has not incorporated any physical aspects of the slab finishing domain, the principles of this control approach can be generally applied to other physical tasks as argued in Chapter 1, which, of course, has to be further assessed in real applications.

7.4 Future Work

This thesis still leaves many interesting research issues. In addition to the section 7.2.4, some more discussions are presented below.

7.4.1 Extension

The most immediate need for more research is to test MARCO on real mobile robots. To do this, the structure of the architecture should be more clearly defined. More task templates need to be developed, especially ones to deal with monitoring and failure recovery. In the low level control layer, more fuzzy behaviours need to be developed and incorporated into the system. Urgently needed are those for perception purposes, such as recognition, which can be used as task-oriented behaviours. A better perceptual subsystem, incorporating vision and a sonar ring should be developed. The current 2D world model used in the simulated control system is inadequate in the real environments. A better localisation scheme, such as those employing extended Kalman filter[Brussel93] or other methods[Borenstein94], also needs to be developed.

7.4.2 Deliberative Layer

To develop an autonomous mobile robot control system, planning must be used to produce a task execution plan which can be used by the sequencing layer. This is the responsibility of the highest deliberative layer. The task plan must be sketchy, similar to the ones used in experiments or in other forms. This thesis has not touched any of these planning issues. A possible way of planning in MARCO is the use of a task template as a planning operator, similar to that introduced by Firby[Firby89]. Some of the promising planners can be goal-regression planner GAPPS[Kaelbling90], IxTeT[Ghallab94] and SIPE/SIPE2[Wilkins94,95]. The relationship between the sequencing layer and the deliberative layer should also be more clearly defined if planning is to be incorporated. Generally, the sequencing layer should not only arrange the right sequences of controlling activities in the low level control layer, but also monitor the activities and initiate the request for planning to the deliberative layer if the current plan is no longer appropriate to the task execution.

7.4.3 Learning

MARCO provides interesting results for learning the low level control layer. However, the learning of individual fuzzy behaviours is only carried out in the fuzzy membership functions of behaviours. A more interesting question is how to automatically learn the fuzzy control rules of a behaviour. Some results have been reported in how to learn a complete fuzzy logic controller[Cooper93][Cupal94] [Bonarini93]. These can also be the possible ways for learning a fuzzy behaviour. Another possible direction is to learn different sets of control parameters of behaviours for different situations. A behaviour can then be configured by the sequencing layer in face of these different conditions in order to carry out the control more robustly. Similar research has been reported in [Pearce92][Ram92][Ram93].

7.5 Conclusions

Autonomous mobile robot navigation involves a continuous combination of local and global controlling activities. Interactions with environments and sensing happens locally, in the here and now of the robot, but task goals the robot pursues may lie far away in time and space. By organising the right sequences, the robot tries to connect its current actions to its task goals. But the results of this organisation has to become physical activity. The work described in this thesis focused on the relation between task-oriented organisation and physical controlling actions that resides in a robot control system. My approach, presented as a two layer control architecture, MARCO has focused on the separation and co-operation between sequencing and executing. The results of organising the control actions at the higher level are grounded in the low level action. MARCO was started from the definition of basic types of low level control structure, fuzzy behaviours, using fuzzy logic as computing method. Then, it was developed with the method that fuzzy behaviours can be combined to form a complete low level control layer. Here, two types of behaviours were used: reactive and task-oriented. The basic control structure, task template, was further defined in the higher level. Finally, the low level layer was linked to the higher level layer through task templates and the behaviours' soft channel. The result is a two layer, task template/fuzzy behaviour based architecture.

The way to improve low level control layer was further explored. Here, several key concepts were employed in the learning processes: general, functionality focused and a simple-to-complex multistage learning course. It has been shown how the learning processes improved the performance of the low level control systems.

Some of the good properties of MARCO are robust goal-directed control behaviours, fast response time and the abilities to work in the face of sensor noise, uncertainty and imprecision. These abilities have been tested in SIMAR, performing two complex tasks in complex environments.

The work presented in this thesis is not a radical departure from many other control methods now prevalent in the literature. Rather, it is more an extension of two significant methodologies: RAP-like sequencing and fuzzy behaviour-based approaches.

Although the results obtained up to now are promising, this study has left a number of issues that need a deeper investigation. Among the most urgent ones is the test in real mobile robots. I anticipate that studying these aspects will result in a robust, complete three layer control architecture with better representation of the environment.

Reference

[Anderson90] Tracy Anderson and Max Donath, "Animal Behavior as a Paradigm for Developing Robot Autonomy", Robotics and Intelligent Systems, vol. 6, pp. 145-168, 1990.

[Arai89] K. Arai, "Construction Robots that Aid in Advanced Construction", Technical Report, Kajima Corporation, 1989.

[Arbib85] M. Arbib and D.House, "Depth and detours: an essay on visually guided behavior". Technical Report 85-20, COINS, University of Massachusetts, 1985.

[Arkin87] R.C. Arkin, "Motor Schema Based Navigation for a Mobile Robot", Proceedings of the 1987 IEEE International Conference on Robotics and Automation, pp. 264-271.

[Arkin90] Ronald C. Arkin, "Integrating Behavioral, Perceptual and World Knowledge in Reactive Navigation", Robotics and Autonomous Systems, Vol.6, pp. 105-122, 1990.

[Barto et al. 82] A. Barto, C. Anderson, and R. Sutton, "Synthesis of Nonlinear Control Surfaces by a Layered Associative Search Network", Biological Cybernetics, Vol. 43, 1982, pp. 175-185.

[Bernard88] J. A. Bernard, "Use of a rule-based system for process control", IEEE Control Systems Magazine, pp. 3-13, 1988.

[Bonarini93] A. Bonarini, "ELF: Learning Incomplete Fuzzy Rule Sets for an Autonomous Robot", Proceedings of EUFIT'93 - First European Congress on Fuzzy and Intelligent Technologies, Aachen, Germany, Sept. 7-10, 1993.

[Bonasso91] R.P.Bonasso, "Integrating Reaction Plans and Layered Competences Through Synchronous Control", Proceedings of the 1991 International Joint Conference on Artificial Intelligence, Sydney, Australia, August 24-30, 1991, pp. 1225-1231.

[Bonasso94] R.P.Bonasso, D.Kortenkamp, "An Intelligent Agent Architecture in Which to Pursue Robot Learning", Working Notes: MCL-COLT'94 Robot Learning Workshop, July, 1994.

[Bonasso95] R.P. Bonasso, D. Kortenkamp, "Characterizing an Architecture for Intelligent, Reactive Agents", Working Notes: 1995 AAAI Spring Symposium on Lessons Learned from Implemented Software Architecture for Physical Agents, March, 1995.

[Borenstein94] L.Feng, J.Borenstein, H.R. Everett, Ed. J. Borenstein, "Where am I": Sensors and Methods for Mobile Robot Positioning, Technical Report, The University of Michigan, 1994.

[Brooks86] R.A.Brooks, "A Robust Layered Control System for a Mobile Robot", IEEE Journal on Robotics and Automation, Vol. RA-2, No.1, March 1986.

[Brooks89] R.A.Brooks, “A Robot that walks: Emergent Behaviours from a Carefully Evolved Network”, Neural Computation, Vol. 1, pp. 253-262, 1989.

[Broverman87] Carol Broverman and W. Bruce Croft, “Reasoning About Exceptions During Plan Execution”, Proceedings of the National Conference on Artificial Intelligence(AAAI), 1987.

[Brussel93] H.V. Brussel, J. Vandorpe, G.J. Huang, “An Integrated Control System for Enhanced Autonomous Navigation of Mobile Robots”, Proceedings of the Second International Conference on Mechatronics and Robotics, Sept. 27-29, 1993, Duisburg, Germany, pp. 297-318.

[Chatila92] R. Chatila, R. Alami, B.Degallaix, H. Laruelle, “Integrated Planning and Execution Control of Autonomous Robot Actions”, Proceedings of the 1992 IEEE International Conference on Robotics and Automation, May 12-14, 1992, Nice, France, pp. 2689-2696.

[Cheng97] Gordon Cheng and Alexander Zelinsky, “Supervised Autonomy: A Paradigm for Teleoperating Mobile Robots”, Proceedings of the 1997 IEEE/RSJ International Conference on Intelligent Robots and Systems, Sept. 7-11, 1997, Grenoble, France, pp. 1169-1176.

[Connell89] J.Connell, “A Colony Architecture for an Artificial Creature”, Technical Report 1151, Massachusetts Institute of Technology Artificial Intelligence Laboratory, 1989.

[Connell90] Jonathan Connell and Paul Viola, “Cooperative Control of a Semi-Autonomous Mobile Robot”, Proceedings of the IEEE International Conference on Robotics and Automation, 1990.

[Connell92] Jonathan Connell, “SSS: A Hybrid Architecture Applied to Robot Navigation”, Proceedings of the 1992 IEEE International Conference on Robotics and Automation, May 12-14, 1992, Nice, France, pp. 2719-2724.

[Cooper93] Mark G. Cooper and Jacques J. Vidal, “Genetic Design of Fuzzy Controllers”, Proceedings of the Second International Conference on Fuzzy Theory and Technology, Durham, NC, October, 1993.

[Cox90] I.J.Cox, G.T.Wilfong, Autonomous Robot Vehicles, Springer-Verlag, USA, 1990.

[Cox94] Earl Cox, Ed., The Fuzzy Systems Handbook: A Practitioner’s Guide to Building, Using, and Maintaining Fuzzy Systems. Academic Press, Inc., 1994.

[Crowley87] J.L.Crowley, “Coordination of Action and Perception in a Surveillance Robot”, IEEE Expert, pp. 32-43, Winter 1987.

[Cupal94] J.J. Cupal, B.M. Wilamowski, "Selection of Fuzzy Rules Using a Genetic Algorithms", Proceedings of the World Congress on Neural Networks, June 5-9, 1994, San Diego, CA.

[Davis91] L. Davis, Ed., Handbook of Genetic Algorithms, Van Nostrand Reinhold, New York, 1991.

[Dorigo91] M. Dorigo and U. Schnepf, "Organization of Robot Behavior Through Genetic Learning Processes", Proceedings of the 5th International Conference on Advanced Robotics, 1991, Vol. 2, pp. 1456-1460.

[Musliner93] David J. Musliner, Edmund H. Durfee, and Kang G. Shin. "CIRCA: A Cooperative Intelligent Real-Time Control Architecture." IEEE Transactions on Systems, Man, and Cybernetics (Special Issue on Planning, Scheduling, and Control) SMC-23(6):1561-1574, 1993.

[Fikes et al 72] R. Fikes, P. Hart, and Nils Nilsson, "Learning and Executing Generalized Robot Plans", Artificial Intelligence, Vol. 3, 1972, pp. 251-288.

[Firby89] R.J. Firby, "Adaptive Execution in Complex Dynamic Worlds", Technical Report YALEU/CSD/RR#672, Yale University, 1989.

[Firby95] R.J. Firby, "An Architecture for a Synthetic Vacuum Cleaner", Proceedings of the 1995 AAAI Spring Symposium: Lessons Learned from Implemented Software Architectures for Physical Agents, Stanford University, March 27-29, 1995.

[Flynn89] A. Flynn, R. Brooks, "Building Robots: Expectations and Experiences", Proceedings of the 1989 IEEE/RSJ International Workshop on Intelligent Robots and Systems(IROS89), Sept. 4-6, Tsukuba, Japan.

[Garcia-Alegre93] M.C. Garcia-Alegre, et al., "Optimazation of Fuzzy Behaviour-Based Robots Navigation in Particially Known Industrial Environments", Proceedings of the Third International Conference on Industrial Fuzzy Control and Intelligent Systems, December 1-3, 1993, Houston, USA, pp. 50-54.

[Gat90] Erann Gat, et al., "Path Planning and Execution Monitoring for a Planetary Rover", Proceedings of the IEEE International Conference on Robotics and Automation, 1990.

[Gat91a] Erann Gat, "ALFA: A Language for Programming Reactive Robotic Control Systems", Proceedings of the IEEE Conference on Robotics and Automation, 1991.

[Gat91b] Erann Gat, "Robust, low-computation, sensor driven control for task-directed navigation", Proceedings of the 1991 IEEE Conference on Robotics and Automation, 1991.

[Gat92] Erann Gat, "Integration Reaction and Planning in a Heterogeneous Asynchronous Architecture for Controlling Real World Mobile Robots", Proceedings of the Tenth National Conference on Artificial Intelligence(AAAI), Menlo Park:AAAI Press, 1992.

[Gat93] Erann Gat, "On the Role of Stored Internal State in the Control of Autonomous Mobile Robot", AI Magazine, Spring 1993.

[Gat94] Erann Gat, et al., "Behavior Control for Robotic Exploration of Planetary Surfaces", IEEE Transactions on Robotics and Automation, August 1994.

[Georgeff87] Michael Georgeff and Amy Lanskey, "Reactive Reasoning and Planning", Proceedings of the National Conference on Artificial Intelligence(AAAI), 1987.

[Ghallab94] M. Ghallab, H. Laruelle, "Representation and Control in IxTeT, a Temporal Planner", Proceedings of the 2nd International Conference on Artificial Intelligence Planning Systems(AIPS'94), June 13-15, 1994, Chicago, Illinois.

[Giralt90] G.Giralt, R.Chatila, M.Vaiset, "An Integrated Navigation and Motion Control System for Autonomous Multisensory Mobile Robot", Autonomous Robot Vehicles, I.J.Cox and G.T.Wilfong, Ed., Springer-Verlag, pp. 420-443, 1990.

[Goldberg89] D.E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, Reading, MA, 1989.

[Goodridge94] S. Goodridge, R. Luo, "Fuzzy Behavior Fusion for Reactive Control of an Autonomous Mobile Robot: MARGE", Procs. of the 1994 IEEE International Conference on Robotics and Automation, San Diego, CA, May, 1994.

[Grefenstette et al. 90] J.J.Grefenstette, C.L.Ramsey, and A.C.Schultz, "Learning Sequential Decision Rules using Simulation Models and Competition", Machine Learning, Vol. 5, No.4, pp. 355-381, 1990.

[Hammond90] Kristian Hammond, et al., "Towards a Theory of Agency", Proceedings of the DARPA Workshop on Innovative Approaches to Planning, Scheduling and Control, 1990.

[Hasemann95] Jorg-Michael Hasemann, "Robot Control Architectures: application requirement, approaches, and technologies", Technical Report, Technical Research Center of Finland, 1995.

[Hexmoor93] H. Hexmoor, J. Lammens, S.C. Shapiro, "Embodiment in GLAIR: A Grounded Layered Architecture with Integrated Reasoning for Autonomous Agents", Technical Report, TR-93-10, Computer Science Department, SUNY at Buffalo, USA.

[Hexmoor95] H. Hexmoor, "Smart are in the Architecture", Proceedings of the 1995 AAAI Spring Symposium: Lessons Learned from Implemented Software Architectures for Physical Agents, Stanford University, March 27-29, 1995.

[Hinkel88] R. Hinkel, et al, "A Rotating Laser Range Finder and Attached Data Interpretation for Use in an Autonomous Mobile Robot", Microprocessing and Microprogramming, 24, pp. 411-418, 1988.

[Hinkel89] R. Hinkel, T.Knieriemen, "Environment Perception with a Laser Radar in a Fast Moving Robot", Robot Control 1988(SYROCO88) selected papers from the 2nd IFAC Symposium, Pergamon, Oxford, UK, 1989.

[Holland75] J.H. Holland, ed., Adaptation in Natural and Artificial Systems, MIT Press, Cambridge, MA, 1975.

[Hoppen89] P. Hoppen, T. Knieriemen, E. von Puttkamer, "Sensor Data Processing and Navigation in a Laser-Radar based Autonomous Robot", Proceedings of the Second International Conference on Intelligent Autonomous Systems, Dec. 11-14, 1989, Amsterdam.

[Hu94] H. Hu, M. Brandy, "Sensor-based Control Architecture", in Advanced Guided Vehicles, World Scientific Press, 1994.

[Ingrand90] Francois Felix Ingrand and Michael Georgeff, "Managing Deliberating and Reasoning in Real Time AI systems", Proceedings of the DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control, 1990.

[Ingrand95] F.F. Ingrand, R. Chatila, R. Alami, F. Robert, "Embeded Control of Autonomous Robots using Procedural Reasoning", Proceedings of the 1995 International Conference on Robotics and Automation(ICRA'95), Nagoya, Japan.

[Janikow91] C. Janikow and Z. Michalewicz, "An Experimental Comparison of Binary and Floating Representations in Genetic Algorithms", Proceedings of the 4th International Conference on Genetic Algorithms, 1991, pp. 31-36.

[Kaelbling88] Leslie Kaelbling, "Goals as parallel program specifications", In Proceedings of the AAAI Conference, Minneapolis-St.Paul, MN, 1988.

[Kaelbling90] Leslie Kaelbling and Stanley Rosenschein, "Action and Planning in Embedded Agents", Robotics and Autonomous Systems, vol.6, pp. 35-48, 1990.

[Kajima89] Kajima Corporation, New Technology Leaflet No. 070: Concrete Slab Finishing Robot, 1989.

[Khatib86] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots", The International Journal of Robotics Research, 5(1):90-98, 1986.

[King77] P. J. King, E. H. Mamdami, "The application of fuzzy control systems to industrial processes", Automatica, Vol. 13, pp. 235-242, 1977.

[Konolige92] Kurt Konolige, et al., "FLAKEY, an Autonomous Mobile Robot", Technical Report, Stanford Research Institute International, July 20, 1992.

[Kruse94] R.Kruse, J.Gebhardt, F. Klawonn, ed. "Foundations of Fuzzy Systems", 1994, Published by John Wiley & Sons Ltd., UK.

[Laird91] John Laird, et al., "Robo-Soar: An Integration of External Interaction, Planning and Learning using Soar", Robotics and Autonomous System, 1991.

[Latombe91] J. C. Latombe,Ed., Robot Motion Planning, Kluver Academic Publishers, Boston, MA, 1991.

[Lee90] Chuen Chien Lee, "Fuzzy Logic in Control Systems: Fuzzy Logic Controller - Part I, Part II", IEEE Transactions on Systems, Man, and Cybernetics, vol. 20, No. 2 March/April 1990.

[Lembeck93] Michael F. Lembeck, "Fuzzy Logic Control of the Commercial Refrigeration/Incubation Module(CRIM)", AIAA Space Programs and Technology Conferences, 1993.

[Leonard89] John J. Leonard and Hugh F. Durrant-Whyte, "Active Sensor Control for Mobile Robotics", Technical Report, No. OUEL 1756/89. University of Oxford, UK.

[Luo97] Ren C. Luo, "Remote Supervisory Control of A Sensor Based Mobile Robot Via Internet", Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, September 7-11, 1997, Grenoble, France, Vol. 2, pp. 1163-1168.

[Maes90] Pattie Maes, "A Bottom-Up Mechanism for Behavior Selection in an Artificial Creature", From Animals to Animats, J.-A. Meyer and S.W. Wilson, Eds.(MIT Press, Cambridge, MA, 1990), pp. 238-246.

[Mamdani75] E. H. Mamdani, S. Assilian, "An experiment in linguistic synthesis with a fuzzy logic controller", International Journal of Man-Machine Studies, 7, pp. 1-13, 1975.

[Marks94] Robert J. Marks II, "Fuzzy Logic Technology and Applications", IEEE Technology Update Series, Ed. , 1994.

[Martinez93] A. Martinez, et al., "Fuzzy Logic Based Collision Avoidance for a Mobile Robot", Proceedings of the Third International Conference on Industrial Fuzzy Control and Systems, December 1-3, 1993, Houston, USA, pp. 66-69.

[Mataric90] M. Mataric, "A Distributed Model for Mobile Robot Environment Learning and Navigation", Technical Report 1228, MIT Artificial Intelligence Laboratory, 1990.

[Michalewicz92] Z. Michalewicz, C.Z. Janikow and J.B. Krawczyk, "A Modified Genetic Algorithm for Optimal Control Problems", Computers and Mathematics with Applications, Vol. 23, No. 12, pp. 83-94, 1992.

[Miller89] David P. Miller, "Execution Monitoring for a Mobile Robot System", Proceedings of the SPIE Conference on Intelligent Control and Adaptive Systems, vol. 1196, pp. 36-43, Philadelphia, PA, Noverber 1989.

[Moravec83] H.P.Moravec, "The Stanford Cart and the CMU Rover", Proc. IEEE 71, pp. 872-884, July 1983.

[Moravec88] H.P.Moravec, "Sensor Fusion in Certainty Grids for Mobile Robots", AI Magazine, Vol. 9, No. 2, pp.61-74, Summer, 1988.

[Nilsson69] N.J. Nilsson, "A mobile automation: an application of artificial intelligence techniques", Proceedings of the First IJCAI, Washington D.C., May 1969.

[Nilsson94] N.J. Nilsson, "Teleo-Reactive Programs for Agent Control", Journal of Artificial Intelligence Research, vol.1, pp. 139-158, 1994.

[Noreils90] Fabrice Noreils, "Integrating Error Recovery in a Mobile Robot Control System", Proceedings of the IEEE International Conference on Robotics and Automation, 1990.

[Payton86] D. W. Payton, "An Architecture for Reflexive Autonomous Vehicle Control", Proceedings of the 1986 IEEE International Conference on Robotics and Automation, April 7-10, 1986, San Francisco, California, pp. 1838-1845.

[Payton90] D. W. Payton, J. K. Rosenblatt, D.M. Keirsey, "Plan Guided Reaction", IEEE Transactions on Systems, Man and Cybernetics, Vol. 20, No. 6, Nov./Dec. 1990.

[Pearce92] Michael Pearce, Ronald Arkin, Ashwin Ram, "The Learning of Reactive Control Parameters Through Genetic Algorithm", Proceedings of the 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems, 1992.

[Pin92] F.G. Pin, et al., "Using Custom-Designed VLSI Fuzzy Inferencing Chips for the Autonomous Navigation of a Mobile Robot", Proceedings of IROS 92, the 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems, July 7-10, 1992, Raleigh, North Carolina.

[Qiu96a] Jiancheng Qiu, Michael Walters, "Fuzzy Behaviour Organisation and Fusion for Mobile Robot Reactive Navigation", Proceedings of the Ninth International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, Fukuoka, Japan, June 4-7, 1996, pp.719-724.

[Qiu96b] Jiancheng Qiu and Michael Walters, "Mobile Robot Reactive Navigation based on Fuzzy Behaviour Organisation and Fusion", Proceedings of the 29th International Symposium on Automative Technology and Automation, Florence, Italy, June 3-6, 1996.

[Qiu97a] Jiancheng Qiu and Michael Walters, "A GA-based Learning Algorithm for the Learning of Fuzzy Behaviour of a Reactive Robot Control System", Proceedings of the Second IEE/IEEE International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications, 2-4 September, 1997, Glasgow, UK.

[Qiu97b] Jiancheng Qiu and Michael Walters, "Learning of Membership Functions of Fuzzy Behaviours for a Mobile Robot Control System", Proceedings of the Tenth IEEE/RSJ International Conference on Intelligent Robots and Systems: Innovative Robotics for Real-World Applications, September 7-11, 1997, Grenoble, France, pp. 772-777.

[Ram92] A.Ram, et al, "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, Atlanta, Georgia, 1992.

[Ram93] A.Ram and Juan Carlos Santamaria, "Multistrategy Learning in Reactive Control Systems for Autonomous Robotic Navigation", Informatica 17, 1993, pp.347-369.

[Reignier94] Patrick Reignier, "Fuzzy Logic Techniques for Mobile Robot Obstacle Avoidance", Robotics and Autonomous Systems, vol. 12, 1994, pp. 143-153.

[Saffiotti93] Alessandro Saffiotti, "Some Notes on the Integration of Planning and Reactivity in Autonomous Mobile Robots", Proceedings of the AAAI Spring Symposium on Foundations of Autonomous Planning, Standford, CA, 122-126.

[Saffiotti et al 93a] A. Saffiotti, E. Ruspini, and K. Konolige, "A fuzzy controller for flakey, an autonomous mobile robot," Technical Note 529, AI Center, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025, April 1993.

[Saffiotti et al 93b] A. Saffiotti, E. Ruspini, and K.Konolige, "Integrating Reactivity and Goal-Directedness in a Fuzzy Controller", Procs. of the 2nd Fuzzy-IEEE Conference. San Francisco, CA ,1993, pp. 134-139.

[Saffiotti95] A. Saffiotti, K. Konolige, E. H. Ruspini, "A Multivalued Logic Approach to Integrating Planning and Control", Artificial Intelligence, 76(1-2), 1995, pp. 481-526.

[Schoppers87] M.J. Schoppers, "Universal Plans for Reactive Robots in Unpredictable Environments", Proceedings of the Tenth International Joint Conference on Artificial Intelligence(IJCAI'87), Milan, Italy, August 23-28, 1987, pp. 1039-1046.

[Shafer89] Steve Shafer and William Whittaker, "Development of an Integrated Mobile Robot System at Carnegie Mellon University: June 1988 Annual Report", Tehnical Report CMU-RI-TR-89-22, The Robotics Institute, Carnegie Mellon University.

[Simmons90] Reid Simmons, "An Architecture for Coordinating Planning, Sensing and Action", Proceedings of the DARPA Workshop on Innovative Approahes to Planning, Scheduling, and Control, 1990.

[Simmons94] Reid Simmons, "Structured Control for Autonomous Robots", IEEE Transactions on Robotics and Automation, vol. 10, No. 1, Feburary 1994, pp. 34-43.

[Skubic94] M. Skubic, "Design of a Two-Level Fuzzy Controller for a Reactive Miniature Mobile Robot", Proceedings of the Third International Conference on Industrial Fuzzy Control and Intelligent Systems, December 1-3, 1993, Houston, 1993, pp. 224-227.

[Slack90] Marc G. Slack, "Situationally Driven Local Navigation for Mobile Robots", JPL Publication 90-17, California Institute of Technology Jet Propulsion Laboratory, April, 1990.

[Slack93] Marc G. Slack, "Navigation Templates: Mediating Qualitative Guidance and Quantitative Control in Mobile Robots", IEEE Transactions on System, Man, and Cybernetics, Vol. 23, No. 2, March/April, 1993.

[Smith89] D.B. Smith and J.R. Matijevic, "A System Architecture for a Planetary Rover", Proceedings of the NASA Conference on Space Telerobotics, Vol. 1, JPL Publications 89-7, 1989.

[Soldo90] Monnett Soldo, "Reactive and Preplanned Control in a Mobile Robot", Proceedings of the IEEE International Conference on Robotics and Automation, 1990.

[Song92] K.Y. Song, J.C. Tai, "Fuzzy navigation of a mobile robot", Procs. of the 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems, Raleigh, North Carolina, July, 1992.

[Sugeno85] M. Sugeno, M. Nishida, "Fuzzy control of a model car", Fuzzy sets and Systems, 16, pp. 103-113, 1985.

[Sugeno89] M. Sugeno, et.al, "Fuzzy algorithmic control of a model car by oral instructions", Fuzzy sets and Systems, 31, pp.207-219, 1989.

[Summers95] Della Summers, et al., Ed., Longman Dictionary of Contemporary English, Third Edition, 1995, Longman Group Ltd.

[Tanenbaum92] Andrew S. Tanenbaum, Ed., Modern Operating Systems, Prentice-Hall International, 1992.

[Thau97] R. S. Thau, profile: R.A. Brooks, http://www.ai.mit.edu/people/brook.

[Thorpe90] C.E.Thorpe, "Vision and Navigation -- The Carnegie Mellon Navigation Laboratory", Kluwer Academic Publishers, USA, 1990.

[Tilford97] C. F. Neves and J. O. Gray, "Architecture for Advanced Robotic Operation", http://www.salford.ac.uk/telford/Cybernetics/Cybernetics_Home.html

[Tong80] R.M. Tong, "Fuzzy Control of Activated Sludge Wastewater Treatment Process", Automatica, Vol. 16, 659-701.

[Vandorpe94] J. Vandorpe, "A Reflexive Navigation Algorithm for an Autonomous Mobile Robot", Proceedings of the 1994 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems, October 2-5, 1994, Las Vegas, USA, pp. 251-258.

[Weiβ94] Gerhard Weiβ, et al., "Keeping Track of Position and Orientation of Moving Indoor Systems by Correlation of Range-Finder Scans", Proceedings of the 1994 IEEE/RSJ International Conference on Intelligent Robots and Systems, Sept. 12-14, 1994, Munich, Germany, pp. 595-601.

[Weisbin89] C.R.Weisbin, et al, "Autonomous Mobile Robot Navigation and Learning", Computer, pp. 29-35, June 1989.

[Whitley89] D. Whitley, "The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best", Proceedings of the Third International Conference on Genetic Algorithms, pp. 116-121, 1989.

[Wilkins94] D.E. Wilkins, et al, "Planning and Reacting in Uncertain and Dynamic Environment", Journal of Experimental and Theoretical AI, Vol.6, 1994, pp. 197-227.

[Wilkins95] D.E. Wilkins, K.L. Myers, "A Common Knowledge Representation for Plan Generation and Reactive Execution", Journal of Logic and Computation, 1995.

[Wing89] R.D.Wing, "Robotics in Construction — a State of the Art Review", Proc. Instn Civ. Engrs., Part I, 1989, 86, Oct., pp. 953-961.

[Yasunodu85] S. Yasunodu, S. Myamoto, "Automatic train operation by predictive fuzzy control", Industrial applications of fuzzy control, M. Sugeno, ed., Amsterdam, North Holland, 1985.

[Zadeh65] L.A. Zadeh, "Fuzzy Sets", Information and Control, 8, pp.338-353, 1965.

Appendix A: Review of Fuzzy Logic Control

Introduction

During the past several years, fuzzy logic control has emerged as one of the most successful methods to solve control problem, including mobile robotics. The pioneering research of Mamdani and Assilian on fuzzy control[Mamdani75] was motivated by Zadeh's research on the linguistic approach and system analysis based on theory of fuzzy sets[Zadeh65]. Applications of fuzzy logic control has considerably increased recently[King77][Yasunodu85][Bernard88][Lembeck93] [Tong80] [Mark94]. These applications have proved effective utilisation of fuzzy control in the context of complex, ill-defined processes that can be controlled by a skilled human operator without the knowledge of their underlying dynamics. In mobile robotics, the lack of a precise model of a mobile robot's environment, noise sensor and uncertainty provides considerable incentive to the use of fuzzy control. Fuzzy control has been explored for mobile robot guidance by many researchers[Song92] [Goodridge94] [Sugeno85] [Reignier94][Garcia-Alegre93] [Martinez93]. Successful hardware implementations have been realised by Sugeno[Sugeno89], Pin[Pin92], Konolige [Konolige92] and Goodridge[Goodridge94][Luo97], etc. One of the most successful robot, Konolige's FLAKEY, uses fuzzy logic to define control behaviours for a variety of tasks. These behaviours are combined through context dependent blending behaviour approach to create single set of control output. In the following sections, fuzzy set and fuzzy logic control are briefly introduced.

1. Fuzzy Sets and Operations

Let X be a domain of objects, called the universe of discourse, whose generic elements are denoted by x. Thus, X = {x} and X could be discrete or continuous.

Definition 1.1: Fuzzy Sets

A fuzzy set A in an universe of discourse X is characterised by a membership function, $\mu_A(x)$, which maps the domain X to a real number in the interval [0,1], namely, $\mu_A: X \rightarrow [0,1]$. The membership function $\mu_A(x)$ is the degree of membership of x in A. A fuzzy set can be considered as a generalisation of the concept of a classical set whose membership function μ_A is from X to $\{0, 1\}$, with $\mu_A(x) = 1$ or 0 according as x does or does not belong to A. Thus, a fuzzy set A in X can be represented as a set of ordered pairs of x and $\mu_A(x)$, such as the fuzzy set $A = \{(x, \mu_A(x)) | \forall x \in X\}$.

Let A and B be two fuzzy sets in X with membership function μ_A and μ_B, respectively. The traditional set theory operations of union, intersection and complement of classical subsets of X can be extended for fuzzy sets via their membership functions as proposed by Zadeh[Zadeh65].

Definition 1.2: Union, Intersection, and Complement

The union $C = A \cup B$ or A OR B with the membership function $\mu_C(x)$ is defined by

$$C = \{(x, \mu_C(x) | \mu_C(x) = \max(\mu_A(x), \mu_B(x), \forall x \in X\};$$

The intersection $D = A \cap B$ or A AND B with the membership function $\mu_D(x)$ is defined by

$$D = \{(x, \mu_D(x) | \mu_D(x) = \min(\mu_A(x), \mu_B(x), \forall x \in X\};$$

The complement $E = \not\subset A$ or NOT A with the membership function μ_E is defined by

$$E = \{(x, \mu_E(x) | \mu_E(x) = 1 - \mu_A(x), \forall x \in X\}.$$

For example, suppose that the universe of discourse X = {age}= {10, 20, 30, 40, 50, 60, 70, 80, 90, 100}, A = {old-age} = {(40, 0.1), (50, 0.3), (60, 0.5), (70, 0.8), (80, 1.0), (90, 1.0), (100, 1.0)}, B = { middle-aged} = {(20, 0.2), (30, 0.5), (40, 1.0), (50, 1.0), (60, 0.7), (70, 0.4)}.

From the definitions of union, intersection, and complement, the union C = A $\cup$ B = {(20, 0.2), (30, 0.5), (40, 1.0), (50, 1.0), (60, 0.7), (70, 0.8), (80, 1.0), (90, 1.0), (100, 1.0)}, the intersection D = A $\cap$ B = {(40, 0.1), (50, 0.3), (60, 0.5), (70, 0.4)} and the complement E = $\not\subset$ A = {(10, 1.0), (20, 1.0), (30, 1.0), (40, 0.9), (50, 0.7), (60, 0.5), (70, 0.2)}.

The above defined are Zadeh's conventional operators, called T-operators. Other types of T-operators are also used to define the connections AND($\cup$), OR($\cap$), and NOT($\not\subset$) for fuzzy reasoning applications. The commonly used are listed in Table 2-1.

Table 1 Definitions of Some T-operators

z	x AND y	x OR y	NOT x
=	min(x, y)	max(x, y)	1-x
=	xy	x+y-xy	1-x
=	max(x+y-1,0)	min(x+y, 1)	1-x

The choice of an operator depends on the applications and computation reasons. However, for the design of fuzzy logic controllers(FLC), Zadeh's conventional T-operators provides simple and fast computation and are already widely used. They have proved to work well in many applications.

2. Fuzzy Logic and Fuzzy If-Then Rules

In classical two-valued logic, a proposition P is either true or false. In fuzzy logic, a proposition P is assigned a degree of truth or false with fuzzy sets involved. In a fuzzy logic controller, a proposition P is a control rule, expressed as "If X is A Then Y is B". The portion on the left side of *Then* is called the antecedent part of the rule, while the portion on the right is the action or consequent part. Because control rules are expressed using linguistic terms, it is easy to express human experiences and knowledge of the control process. In most FLC applications, X is usually represented in multiple input variables and Y is a single output variable. A fuzzy rule can be written as "If X1 is A1 and(or) X2 is A2 and(or)......... Xn is An Then Y is B". The composite multiple input fuzzy sets are computed through T-operators in the antecedent portion and is contributed to output fuzzy set through fuzzy implication.

Suppose R represents the antecedent part of a rule and S represents the consequent part, then the rule is expressed as "If R Then S" or "R $\rightarrow$ S", where $\rightarrow$ denotes a fuzzy implication, a function which associates the input fuzzy set to output fuzzy set. There are seven families of fuzzy implication functions described in [Lee90]. Two commonly used implication functions are:

(1) min-operation rule of a fuzzy implication

$$R \rightarrow S = R \times S = \int_{u \times v} \frac{\mu_R(u) \wedge \mu_S(v)}{(u,v)}, \text{ where } u \in U, v \in V;$$

(2) product-operation rule of a fuzzy implication

$$R \rightarrow S = R \times S = \int_{u \times v} \frac{\mu_R(u) * \mu_S(v)}{(u,v)}, \text{ where } u \in U, v \in V;$$

3. Fuzzy Logic control

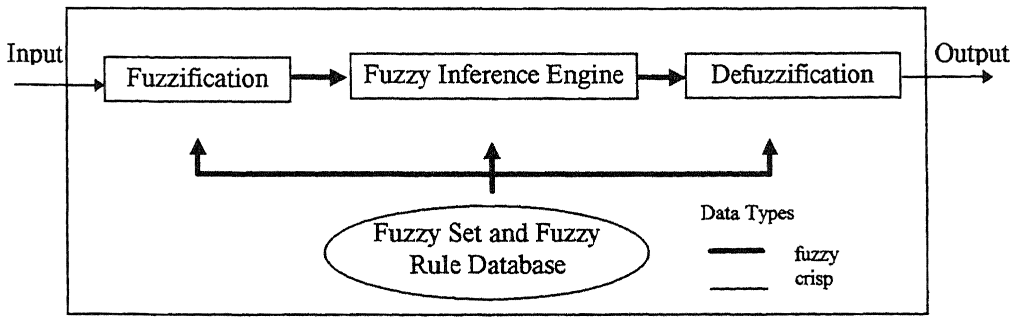


Fig.1 Fuzzy Logic Controller Diagram

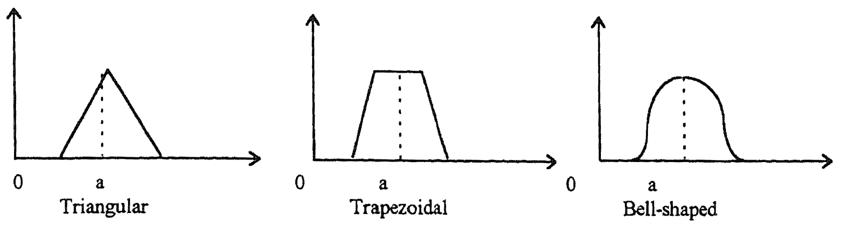


Fig. 2 Membership Function Types

A fuzzy logic controller, such as shown in Fig.1, involves receiving the input values and converting the signals to fuzzy variables. The fuzzy control rules relate input fuzzy variables to an output fuzzy variables using the compositional rules of inference. The output control action is determined by defuzzification process to obtain crisp values. The main processes of fuzzy logic control are fuzzification, fuzzy rule inference and defuzzifcaiton.

3.1 Fuzzification

Before setting the fuzzy levels such as BIG, MEDIUM, and CLOSE, one must evaluate the range of input variables. There are three types of membership functions used in the design of FLC, a bell-shaped function, a triangular-shaped function and a trapezoid-shaped function as shown in Fig. 2. How to select the membership function is based on applications. Basically, a triangular type of the membership function supports a simple representation and fast computation and therefore is widely used. Through a membership function, the fuzzification process transforms the range of values of input variables into corresponding universe of discourse and the values of fuzzy variables can be determined.

3.2 Fuzzy Inference Process

The inference engine is the heart of an FLC. The inference process is based on fuzzy rules and deduces fuzzy control actions by using the fuzzy implication and the compositional rules of inference in fuzzy logic. In FLC applications, sup-min and sup-product compositional operators are commonly used.

3.3 Defuzzification

Because a nonfuzzy control action from a controller is required, it is necessary to map from fuzzy control actions into nonfuzzy control actions, called defuzzification. The purpose of defuzzification is to produce a crisp control action that best represents the possibility distribution of an inferred fuzzy control action. There are many defuzzification methods found in the literature. Here, the most commonly used centroid method is described.

Suppose that a fuzzy control action with a discrete membership function μ_C has been produced. The centroid method calculates the center gravity of the distribution for the control action. In the case of a discrete universe, this method yields

$$\text{control action} = \sum_{i=1}^{q} \mu_C(z_i) \cdot z_i \Big/ \sum_{i=1}^{q} \mu_C(z_i) \qquad (1)$$

where q is the number of quantification levels of the output space, z_i is the amount of control action at the quantification level I, and $\mu_C(z_i)$ is the degree of membership of z_i in C.

4. Fuzzy Singleton Representation of Output

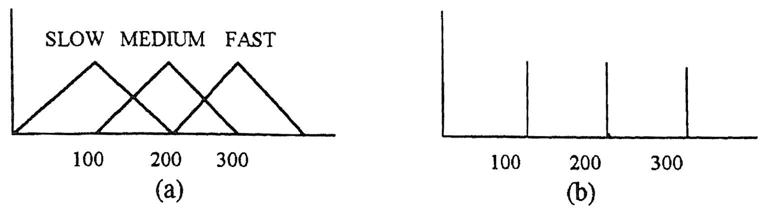


Fig. 3 Fuzzy Output Representation
(a) Standard Representation;
(b) Singleton Representation

In order to reduce the computation expense, we consider a simplified centroid method. This method is based on the fuzzy singleton representation of output. This representation allows us to use a special from of fuzzy set with only one pair having a value and full degree of truth and zero for the rest of pairs in a fuzzy set. This representation is illustrated in Fig. 3. Fuzzy reasoning methods for the two types of representation are the same but have different effects as shown in Fig. 4.

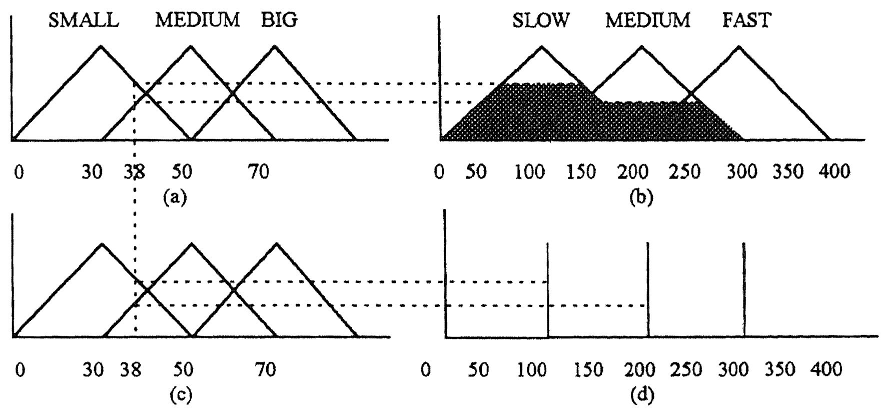


Fig. 4 Fuzzy Inference Results for Singleton Representation
(a) (c) Membership Functions of Input Fuzzy Set;
(b) Output Fuzzy Space;
(d) Output Fuzzy Singleton Values.

Fuzzy rules for (a)-(b):
if dist is SMALL than speed is SLOW
if dist is MEDIUM than speed is MEDIUM
if dist is BIG than speed is FAST

Fuzzy rules for (c)-(d):
if dist is SMALL than speed is 100
if dist is MEDIUM than speed is 200
if dist is BIG than speed is 300

Given a speed value 38, for the standard form, the resulted output fuzzy set is truncated as the shaded area after inference process, while for the singleton form, it is two clipped single values.

The advantage of the representation is obvious. It can greatly reduce computation time to produce the output fuzzy set, especially when quantification level of output is big. Furthermore, defuzzification is simplified with the formula (1) transformed to formula (2) because $\mu_C(z_i)$ is 1.0 at singleton points and 0 otherwise:

$$C = \sum_{i=1}^{m} w_i * z_i \Big/ \sum_{i=1}^{m} w_i \qquad (2),$$

where m becomes the number of rules, w_i is the weight of the antecedent of the ith rule and z_i is the singleton value for the ith rule output.

We can compare the computation process for output values in Fig. 4. For the standard form, we know that each output fuzzy set has 5 members and output quantification level is 9. Therefore, 3*5 = 15 fuzzy set operations are needed to create output fuzzy space as the shaded area in reasoning stage. In the defuzzifiation stage, the output value is calculated with the formula (1) as:

(0*0 + 0.5*50 + 0.6*100 + 0.5*150 + 0.3*200 + 0.3*250 + 0*300 + 0*350 + 0*400)/(0 + 0.5 + 0.6 + 0.5 + 0.3 + 0.3 + 0 + 0 + 0) = 134.09.

For the singleton from, the output value is simply calculated with the formula (2) as:

(0.6*100 + 0.3*200 + 0*300)/(0.6 + 0.3 + 0) = 133.33.

The reduction of computation time can be significant with singleton representation when the number of fuzzy rule and quantification level increases. The drawback of the singleton representation is that fuzziness is lost at the output evaluation and transfer function becomes linear.

Appendix B:

Initial population of Avoid Obstacle for both multistage and non-multistage learning

0th	side_low	side_high	front_low	front_high	turn	speed
50 members	577.311401	753.700301	431.071324	622.401424	0.907611	-33.163138
	734.762928	1127.761728	372.913446	1092.665046	1.387465	-69.310339
	149.48342	180.30482	566.080659	1333.906359	2.909277	-62.136789
	471.218704	1200.481504	852.450154	946.734454	1.837355	21.58374
	161.714494	1010.087794	648.365833	1012.052833	2.351816	-92.30295
	96.397641	600.333441	808.111097	1629.894797	1.906284	51.136032
	444.964334	1178.755634	658.117948	1518.892348	2.281362	-67.719098
	401.052593	683.231693	897.768581	986.465381	2.158327	-67.403244
	664.571383	1516.585183	932.60376	1252.53846	1.790922	-26.053337
	243.319343	957.766943	700.263506	1476.134906	1.55521	-1.980579
	153.843004	661.848004	725.590876	1375.078876	1.036345	-61.783059
	791.006344	1518.296644	756.458348	1208.291348	1.997005	99.467332
	927.714106	1524.685006	644.650635	1433.244735	2.213875	29.399357
	68.993969	222.869969	976.387835	1585.606835	1.192182	54.318558
	607.559571	949.609371	576.80755	921.10465	2.346262	-51.591235
	490.09427	684.33857	617.491846	668.049946	2.931422	26.108663
	375.078059	470.058959	866.789942	1668.369542	0.145056	3.966197
	267.054311	344.751611	829.93544	1467.23204	0.277383	-35.75767
	342.429063	1170.556563	775.244756	1634.362556	0.599365	7.075061
	387.762919	1233.200119	920.353096	1216.315096	0.771294	20.266342
	425.47658	1089.38258	945.686355	1389.357855	0.833381	-55.863422
	613.334997	925.233297	829.724895	1072.597995	0.844965	-36.681899
	954.206144	1739.489444	782.189597	929.127197	0.315619	11.560489
	146.96979	174.21429	604.958321	648.135221	1.116843	-97.842641
	368.805402	685.615302	581.660724	879.933024	0.180574	-3.751172
	940.797877	1726.980877	793.802126	900.592226	2.926632	85.034558
	911.992821	1502.096121	996.97932	1522.29192	0.544851	64.807966
	697.044826	1193.384326	345.526171	988.433071	0.107152	81.810677
	705.704146	1267.296646	321.393297	929.395197	2.57162	71.544936
	403.024858	953.168458	381.567531	1021.850931	0.65778	15.752215
	882.66445	1439.11975	843.455381	1460.219681	2.350348	-67.137448
	9.319878	416.188578	224.322457	654.321457	2.108141	82.471358
	427.926242	516.535142	768.972994	1411.063594	1.418311	-12.320966
	716.776163	1235.369363	241.169369	1109.160869	1.817913	31.352487
	730.745253	1455.372153	750.075263	801.737963	1.079661	98.839842
	815.416714	849.083014	584.046511	1447.865311	2.418536	43.676045
	310.015437	661.394937	599.912908	868.232608	2.785255	55.503354
	884.735878	1706.922478	15.237739	483.393439	0.7777	65.812069
	267.901646	454.812446	176.451272	874.058372	0.674187	11.577303
	863.372418	1317.387918	147.84375	258.90945	0.473481	40.382182
	129.214855	1018.797055	313.667268	1045.525068	2.39921	16.656939
	25.621054	765.892354	105.142191	223.411791	0.388935	-94.659118
	618.095841	1393.468341	967.276897	1166.446897	2.17384	-39.75118
	725.980293	1344.003393	982.701085	1386.816085	1.258283	1.263239
	705.806913	1307.680113	557.93695	775.59805	0.117115	-83.454556
	201.062616	619.004316	596.912877	1327.750077	1.254567	-65.450607
	794.828693	1072.234193	138.914572	847.592872	0.151766	-46.663858
	739.272865	1563.563965	838.91798	1559.57558	1.442871	-49.821468
	211.504984	881.035084	317.64487	807.67867	0.315876	-92.2987
	325.148754	897.531354	778.839393	1618.638093	0.274414	-37.290193
Average Deviation	49%	35%	35%	27%	56%	940%

Appendix C: A Complete Sequencing Trace Log

```
&& Starting pulse, top level
&& Starting motor, top level
&& Starting clamp, top level
&& Starting laser, top level
&& Starting wake, top level
&& Starting side segs, top level
&& Starting test wall, top level
&& Starting test wall break, top level
&& Starting check behavior links, top level
&& Starting test where, top level
&& Starting test control, top level
&& Starting test matching, top level
&& Starting draw, top level

&& Starting sequence it, top level

-- Step: 10, state: #[Marco state X: 20.41m, Y: 1.46m, TH: 180.0]

&& Starting following, dad sequence it
-- Current behaviour cluster:
.....Avoid Obstacle
.....Follow Corridor
.....Keep Moving

-- Step: 20, state: #[Marco state X: 20.40m, Y: 1.46m, TH: 180.0]
-- Found a corridor

&& Starting follow it, dad following
&& Starting behavior Follow Corridor
-- Current behaviour cluster:
.....Avoid Obstacle
.....Follow Corridor
.....Keep Moving
.....Recover Stall

-- Step: 15, state: #[Marco state X: 16.91m, Y: 1.43m, TH: 195.8]
-- Found a door

&& Deleting follow it
&& Deleting behavior Follow Corridor
-- Current behaviour cluster:
.....Avoid Obstacle
.....Keep Moving
.....Recover Stall

&& Starting enter it, dad sequence it
&& Starting behavior Cross Door
-- Current behaviour cluster:
.....Avoid Obstacle
.....Cross Door
.....Keep Moving
.....Recover Stall

-- Step: 20, state: #[Marco state X: 14.94m, Y: 2.56m, TH: 90.6]

&& Deleting enter it
&& Deleting behavior Cross Door
-- Current behaviour cluster:
.....Avoid Obstacle
.....Keep Moving
.....Recover Stall

-- Step: 40, state: #[Marco state X: 14.94m, Y: 2.56m, TH: 90.6]
-- Had a position goal
```

```
&& Starting go to pos, dad sequence it
&& Starting behavior Reach Position
-- Current behaviour cluster:
.....Avoid Obstacle
.....Reach Position
.....Keep Moving
.....Recover Stall

-- Step: 85, state: #[Marco state X: 21.77m, Y: 4.78m, TH: 17.0]

&& Deleting go to pos
&& Deleting behavior Reach Position
-- Current behaviour cluster:
.....Avoid Obstacle
.....Keep Moving
.....Recover Stall

-- Step: 40, state: #[Marco state X: 21.77m, Y: 4.78m, TH: 17.0]
-- Had a track goal

&& Starting track it, dad sequence it
&& Starting behavior Track Path
-- Current behaviour cluster:
.....Avoid Obstacle
.....Track Path
.....Keep Moving
.....Recover Stall

-- Step: 50, state: #[Marco state X: 21.67m, Y: 12.01m, TH: 108.7]

&& Deleting track it
&& Deleting behavior Track Path
-- Current behaviour cluster:
.....Avoid Obstacle
.....Keep Moving
.....Recover Stall

-- Step: 40, state: #[Marco state X: 21.66m, Y: 12.04m, TH: 109.0]
-- Had a position goal

&& Starting go to pos, dad sequence it
&& Starting behavior Reach Position
-- Current behaviour cluster:
.....Avoid Obstacle
.....Reach Position
.....Keep Moving
.....Recover Stall

-- Step: 85, state: #[Marco state X: 21.36m, Y: 13.45m, TH: 94.3]

&& Deleting go to pos
&& Deleting behavior Reach Position
-- Current behaviour cluster:
.....Avoid Obstacle
.....Keep Moving
.....Recover Stall

-- Step: 40, state: #[Marco state X: 21.36m, Y: 13.45m, TH: 94.3]
-- Had a track goal

&& Starting track it, dad sequence it
&& Starting behavior Track Path
-- Current behaviour cluster:
.....Avoid Obstacle
.....Track Path
.....Keep Moving
.....Recover Stall
```

```
-- Step: 50, state: #[Marco state X: 21.38m, Y: 4.34m, TH: 267.3]

&& Deleting track it
&& Deleting behavior Track Path
-- Current behaviour cluster:
.....Avoid Obstacle
.....Keep Moving
.....Recover Stall

-- Step: 40, state: #[Marco state X: 21.38m, Y: 4.33m, TH: 263.4]
-- Had a position goal

&& Starting go to pos, dad sequence it
&& Starting behavior Reach Position
-- Current behaviour cluster:
.....Avoid Obstacle
.....Reach Position
.....Keep Moving
.....Recover Stall

-- Step: 85, state: #[Marco state X: 20.79m, Y: 3.58m, TH: 238.0]

&& Deleting go to pos
&& Deleting behavior Reach Position
-- Current behaviour cluster:
.....Avoid Obstacle
.....Keep Moving
.....Recover Stall

-- Step: 40, state: #[Marco state X: 20.79m, Y: 3.58m, TH: 238.0]
-- Had a track goal

&& Starting track it, dad sequence it
&& Starting behavior Track Path
-- Current behaviour cluster:
.....Avoid Obstacle
.....Track Path
.....Keep Moving
.....Recover Stall

-- Step: 50, state: #[Marco state X: 20.61m, Y: 13.49m, TH: 100.4]

&& Deleting track it
&& Deleting behavior Track Path
-- Current behaviour cluster:
.....Avoid Obstacle
.....Keep Moving
.....Recover Stall

-- Step: 40, state: #[Marco state X: 20.61m, Y: 13.51m, TH: 98.7]
-- Had a position goal

&& Starting go to pos, dad sequence it
&& Starting behavior Reach Position
-- Current behaviour cluster:
.....Avoid Obstacle
.....Reach Position
.....Keep Moving
.....Recover Stall

-- Step: 85, state: #[Marco state X: 20.30m, Y: 13.69m, TH: 202.7]

&& Deleting go to pos
&& Deleting behavior Reach Position
-- Current behaviour cluster:
.....Avoid Obstacle
```

```
.....Keep Moving
.....Recover Stall

-- Step: 40, state: #[Marco state X: 20.30m, Y: 13.69m, TH: 202.7]
-- Had a track goal

&& Starting track it, dad sequence it
&& Starting behavior Track Path
-- Current behaviour cluster:
.....Avoid Obstacle
.....Track Path
.....Keep Moving
.....Recover Stall

-- Step: 50, state: #[Marco state X: 19.88m, Y: 3.52m, TH: 285.2]

&& Deleting track it
&& Deleting behavior Track Path
-- Current behaviour cluster:
.....Avoid Obstacle
.....Keep Moving
.....Recover Stall

-- Step: 40, state: #[Marco state X: 19.89m, Y: 3.49m, TH: 287.0]
-- Had a position goal

&& Starting go to pos, dad sequence it
&& Starting behavior Reach Position
-- Current behaviour cluster:
.....Avoid Obstacle
.....Reach Position
.....Keep Moving
.....Recover Stall

-- Step: 85, state: #[Marco state X: 19.70m, Y: 3.31m, TH: 159.2]

&& Deleting go to pos
&& Deleting behavior Reach Position
-- Current behaviour cluster:
.....Avoid Obstacle
.....Keep Moving
.....Recover Stall

-- Step: 40, state: #[Marco state X: 19.70m, Y: 3.31m, TH: 159.2]
-- Had a track goal

&& Starting track it, dad sequence it
&& Starting behavior Track Path
-- Current behaviour cluster:
.....Avoid Obstacle
.....Track Path
.....Keep Moving
.....Recover Stall

-- Step: 50, state: #[Marco state X: 19.58m, Y: 13.48m, TH: 75.9]

&& Deleting track it
&& Deleting behavior Track Path
-- Current behaviour cluster:
.....Avoid Obstacle
.....Keep Moving
.....Recover Stall

-- Step: 40, state: #[Marco state X: 19.59m, Y: 13.51m, TH: 75.3]
-- Had a position goal

&& Starting go to pos, dad sequence it
&& Starting behavior Reach Position
-- Current behaviour cluster:
......Avoid Obstacle
.....Reach Position
.....Keep Moving
.....Recover Stall

-- Step: 85, state: #[Marco state X: 19.10m, Y: 13.67m, TH: 195.8]

&& Deleting go to pos
&& Deleting behavior Reach Position
-- Current behaviour cluster:
.....Avoid Obstacle
.....Keep Moving
.....Recover Stall

-- Step: 40, state: #[Marco state X: 19.10m, Y: 13.67m, TH: 195.8]
-- Had a track goal

&& Starting track it, dad sequence it
&& Starting behavior Track Path
-- Current behaviour cluster:
.....Avoid Obstacle
.....Track Path
.....Keep Moving
.....Recover Stall

-- Step: 50, state: #[Marco state X: 18.93m, Y: 3.52m, TH: 270.3]

&& Deleting track it
&& Deleting behavior Track Path
-- Current behaviour cluster:
.....Avoid Obstacle
.....Keep Moving
.....Recover Stall

-- Step: 40, state: #[Marco state X: 18.93m, Y: 3.52m, TH: 270.3]
-- Had a position goal

&& Starting go to pos, dad sequence it
&& Starting behavior Reach Position
-- Current behaviour cluster:
.....Avoid Obstacle
.....Reach Position
.....Keep Moving
.....Recover Stall

-- Step: 85, state: #[Marco state X: 18.48m, Y: 2.91m, TH: 208.7]

&& Deleting go to pos
&& Deleting behavior Reach Position
-- Current behaviour cluster:
.....Avoid Obstacle
.....Keep Moving
.....Recover Stall

-- Step: 40, state: #[Marco state X: 18.48m, Y: 2.91m, TH: 208.7]
-- Had a track goal

&& Starting track it, dad sequence it
&& Starting behavior Track Path
-- Current behaviour cluster:
.....Avoid Obstacle
.....Track Path
.....Keep Moving
.....Recover Stall
```

```
-- Step: 50, state: #[Marco state X: 18.08m, Y: 13.48m, TH: 105.5]

&& Deleting track it
&& Deleting behavior Track Path
-- Current behaviour cluster:
.....Avoid Obstacle
.....Keep Moving
.....Recover Stall

-- Step: 40, state: #[Marco state X: 18.08m, Y: 13.48m, TH: 105.5]
-- Had a position goal

&& Starting go to pos, dad sequence it
&& Starting behavior Reach Position
-- Current behaviour cluster:
.....Avoid Obstacle
.....Reach Position
.....Keep Moving
.....Recover Stall

-- Step: 85, state: #[Marco state X: 17.89m, Y: 13.73m, TH: 197.4]

&& Deleting go to pos
&& Deleting behavior Reach Position
-- Current behaviour cluster:
.....Avoid Obstacle
.....Keep Moving
.....Recover Stall

-- Step: 40, state: #[Marco state X: 17.89m, Y: 13.73m, TH: 197.4]
-- Had a track goal

&& Starting track it, dad sequence it
&& Starting behavior Track Path
-- Current behaviour cluster:
.....Avoid Obstacle
.....Track Path
.....Keep Moving
.....Recover Stall

-- Step: 50, state: #[Marco state X: 17.48m, Y: 2.96m, TH: 281.7]

&& Deleting track it
&& Deleting behavior Track Path
-- Current behaviour cluster:
.....Avoid Obstacle
.....Keep Moving
.....Recover Stall

-- Step: 40, state: #[Marco state X: 17.49m, Y: 2.93m, TH: 284.1]
-- Had a position goal

&& Starting go to pos, dad sequence it
&& Starting behavior Reach Position
-- Current behaviour cluster:
.....Avoid Obstacle
.....Reach Position
.....Keep Moving
.....Recover Stall

-- Step: 85, state: #[Marco state X: 17.29m, Y: 2.74m, TH: 154.8]

&& Deleting go to pos
&& Deleting behavior Reach Position
-- Current behaviour cluster:
.....Avoid Obstacle
.....Keep Moving
```

```
.....Recover Stall

-- Step: 40, state: #[Marco state X: 17.29m, Y: 2.74m, TH: 154.8]
-- Had a track goal

&& Starting track it, dad sequence it
&& Starting behavior Track Path
-- Current behaviour cluster:
.....Avoid Obstacle
.....Track Path
.....Keep Moving
.....Recover Stall

-- Step: 50, state: #[Marco state X: 17.44m, Y: 13.48m, TH: 92.7]

&& Deleting track it
&& Deleting behavior Track Path
-- Current behaviour cluster:
.....Avoid Obstacle
.....Keep Moving
.....Recover Stall

-- Step: 40, state: #[Marco state X: 17.44m, Y: 13.51m, TH: 95.7]
-- Had a position goal

&& Starting go to pos, dad sequence it
&& Starting behavior Reach Position
-- Current behaviour cluster:
.....Avoid Obstacle
.....Reach Position
.....Keep Moving
.....Recover Stall

-- Step: 85, state: #[Marco state X: 16.73m, Y: 13.64m, TH: 184.1]

&& Deleting go to pos
&& Deleting behavior Reach Position
-- Current behaviour cluster:
.....Avoid Obstacle
.....Keep Moving
.....Recover Stall

-- Step: 40, state: #[Marco state X: 16.71m, Y: 13.64m, TH: 184.1]
-- Had a track goal

&& Starting track it, dad sequence it
&& Starting behavior Track Path
-- Current behaviour cluster:
.....Avoid Obstacle
.....Track Path
.....Keep Moving
.....Recover Stall

-- Step: 50, state: #[Marco state X: 16.60m, Y: 2.98m, TH: 253.1]

&& Deleting track it
&& Deleting behavior Track Path
-- Current behaviour cluster:
.....Avoid Obstacle
.....Keep Moving
.....Recover Stall

-- Step: 40, state: #[Marco state X: 16.59m, Y: 2.96m, TH: 252.2]
-- Had a position goal

&& Starting go to pos, dad sequence it
&& Starting behavior Reach Position
```

```
-- Current behaviour cluster:
.....Avoid Obstacle
.....Reach Position
.....Keep Moving
.....Recover Stall

-- Step: 85, state: #[Marco state X: 16.10m, Y: 2.75m, TH: 164.2]

&& Deleting go to pos
&& Deleting behavior Reach Position
-- Current behaviour cluster:
.....Avoid Obstacle
.....Keep Moving
.....Recover Stall

-- Step: 40, state: #[Marco state X: 16.09m, Y: 2.76m, TH: 164.2]
-- Had a track goal

&& Starting track it, dad sequence it
&& Starting behavior Track Path
-- Current behaviour cluster:
.....Avoid Obstacle
.....Track Path
.....Keep Moving
.....Recover Stall

-- Step: 50, state: #[Marco state X: 15.69m, Y: 13.49m, TH: 96.8]

&& Deleting track it
&& Deleting behavior Track Path
-- Current behaviour cluster:
.....Avoid Obstacle
.....Keep Moving
.....Recover Stall

-- Step: 40, state: #[Marco state X: 15.69m, Y: 13.52m, TH: 93.8]
-- Had a position goal

&& Starting go to pos, dad sequence it
&& Starting behavior Reach Position
-- Current behaviour cluster:
.....Avoid Obstacle
.....Reach Position
.....Keep Moving
.....Recover Stall

-- Step: 85, state: #[Marco state X: 15.48m, Y: 13.73m, TH: 206.5]

&& Deleting go to pos
&& Deleting behavior Reach Position
-- Current behaviour cluster:
.....Avoid Obstacle
.....Keep Moving
.....Recover Stall

-- Step: 40, state: #[Marco state X: 15.48m, Y: 13.73m, TH: 206.5]
-- Had a track goal

&& Starting track it, dad sequence it
&& Starting behavior Track Path
-- Current behaviour cluster:
.....Avoid Obstacle
.....Track Path
.....Keep Moving
.....Recover Stall

-- Step: 50, state: #[Marco state X: 15.34m, Y: 2.99m, TH: 243.7]
```

```
&& Deleting track it
&& Deleting behavior Track Path
-- Current behaviour cluster:
.....Avoid Obstacle
.....Keep Moving
.....Recover Stall

-- Step: 40, state: #[Marco state X: 15.33m, Y: 2.96m, TH: 243.7]
-- Had a position goal

&& Starting go to pos, dad sequence it
&& Starting behavior Reach Position
-- Current behaviour cluster:
.....Avoid Obstacle
.....Reach Position
.....Keep Moving
.....Recover Stall

-- Step: 85, state: #[Marco state X: 14.89m, Y: 2.77m, TH: 155.2]

&& Deleting go to pos
&& Deleting behavior Reach Position
-- Current behaviour cluster:
.....Avoid Obstacle
.....Keep Moving
.....Recover Stall

-- Step: 40, state: #[Marco state X: 14.89m, Y: 2.77m, TH: 155.2]
-- Had a track goal

&& Starting track it, dad sequence it
&& Starting behavior Track Path
-- Current behaviour cluster:
.....Avoid Obstacle
.....Track Path
.....Keep Moving
.....Recover Stall

-- Step: 50, state: #[Marco state X: 14.71m, Y: 12.20m, TH: 104.6]

&& Deleting track it
&& Deleting behavior Track Path
-- Current behaviour cluster:
.....Avoid Obstacle
.....Keep Moving
.....Recover Stall

-- Step: 40, state: #[Marco state X: 14.70m, Y: 12.22m, TH: 105.1]
-- Had a position goal

&& Starting go to pos, dad sequence it
&& Starting behavior Reach Position
-- Current behaviour cluster:
.....Avoid Obstacle
.....Reach Position
.....Keep Moving
.....Recover Stall

-- Step: 85, state: #[Marco state X: 14.27m, Y: 12.40m, TH: 202.0]

&& Deleting go to pos
&& Deleting behavior Reach Position
-- Current behaviour cluster:
.....Avoid Obstacle
.....Keep Moving
.....Recover Stall
```

```
-- Step: 40, state: #[Marco state X: 14.26m, Y: 12.39m, TH: 202.0]
-- Had a track goal

&& Starting track it, dad sequence it
&& Starting behavior Track Path
-- Current behaviour cluster:
.....Avoid Obstacle
.....Track Path
.....Keep Moving
.....Recover Stall

-- Step: 50, state: #[Marco state X: 14.22m, Y: 3.58m, TH: 271.9]

&& Deleting track it
&& Deleting behavior Track Path
-- Current behaviour cluster:
.....Avoid Obstacle
.....Keep Moving
.....Recover Stall

-- Step: 40, state: #[Marco state X: 14.22m, Y: 3.55m, TH: 274.9]
-- Had a anchor goal

&& Starting go to pos, dad sequence it
&& Starting behavior Reach Position
-- Current behaviour cluster:
.....Avoid Obstacle
.....Reach Position
.....Keep Moving
.....Recover Stall

-- Step: 70, state: #[Marco state X: 18.85m, Y: 12.03m, TH: 71.4]
-- Anchored to a wall edge

&& Deleting go to pos
&& Deleting behavior Reach Position
-- Current behaviour cluster:
.....Avoid Obstacle
.....Keep Moving
.....Recover Stall

&& Starting hug it, dad sequence it
&& Starting behaviour Follow Edge
-- Current behaviour cluster:
.....Avoid Obstacle
.....Follow Edge
.....Keep Moving
.....Recover Stall

-- Step: 80, state: #[Marco state X: 16.14m, Y: 13.84m, TH: 173.8]

&& Deleting hug it
&& Deleting behavior Follow Edge
-- Current behaviour cluster:
.....Avoid Obstacle
.....Keep Moving
.....Recover Stall

-- Step: 40, state: #[Marco state X: 16.14m, Y: 13.84m, TH: 173.8]
-- Had a anchor goal

&& Starting go to pos, dad sequence it
&& Starting behavior Reach Position
-- Current behaviour cluster:
.....Avoid Obstacle
.....Reach Position
```

```
.....Keep Moving
.....Recover Stall

-- Step: 70, state: #[Marco state X: 18.80m, Y: 8.41m, TH: 301.3]

&& Deleting go to pos
&& Deleting behavior Reach Position
-- Current behaviour cluster:
.....Avoid Obstacle
.....Keep Moving
.....Recover Stall

-- Step: 70, state: #[Marco state X: 18.80m, Y: 8.41m, TH: 301.3]
-- Anchored to a wall edge

&& Starting hug it, dad sequence it
&& Starting behaviour Follow Edge
-- Current behaviour cluster:
.....Avoid Obstacle
.....Follow Edge
.....Keep Moving
.....Recover Stall

-- Step: 80, state: #[Marco state X: 18.48m, Y: 7.26m, TH: 118.1]

&& Deleting hug it
&& Deleting behavior Follow Edge
-- Current behaviour cluster:
.....Avoid Obstacle
.....Keep Moving
.....Recover Stall

-- Step: 40, state: #[Marco state X: 18.48m, Y: 7.26m, TH: 118.1]
-- Had a position goal

&& Starting go to pos, dad sequence it
&& Starting behavior Reach Position
-- Current behaviour cluster:
.....Avoid Obstacle
.....Reach Position
.....Keep Moving
.....Recover Stall

-- Step: 85, state: #[Marco state X: 15.45m, Y: 2.76m, TH: 240.0]

&& Deleting go to pos
&& Deleting behavior Reach Position
-- Current behaviour cluster:
.....Avoid Obstacle
.....Keep Moving
.....Recover Stall

-- Step: 40, state: #[Marco state X: 15.45m, Y: 2.76m, TH: 240.0]
-- Had a from goal

&& Starting go to pos, dad sequence it
&& Starting behavior Reach Position
-- Current behaviour cluster:
.....Avoid Obstacle
.....Reach Position
.....Keep Moving
.....Recover Stall

-- Step: 90, state: #[Marco state X: 16.12m, Y: 2.66m, TH:  7.0]
-- Anchored to a wall edge

&& Deleting go to pos
&& Deleting behavior Reach Position
-- Current behaviour cluster:
.....Avoid Obstacle
.....Keep Moving
.....Recover Stall

&& Starting hug it, dad sequence it
&& Starting behaviour Follow Edge
-- Current behaviour cluster:
.....Avoid Obstacle
.....Follow Edge
.....Keep Moving
.....Recover Stall

-- Step: 100, state: #[Marco state X: 14.10m, Y: 3.47m, TH: 259.6]

&& Deleting hug it
&& Deleting behavior Follow Edge
-- Current behaviour cluster:
.....Avoid Obstacle
.....Keep Moving
.....Recover Stall

-- Step: 150, state: #[Marco state X: 14.10m, Y: 3.47m, TH: 259.6]
-- Found a door

&& Starting go out, dad sequence it
&& Starting behavior Cross Door
-- Current behaviour cluster:
.....Avoid Obstacle
.....Cross Door
.....Keep Moving
.....Recover Stall

-- Step: 180, state: #[Marco state X: 14.80m, Y: 1.83m, TH: 254.5]

&& Deleting go out
&& Deleting behavior Cross Door
-- Current behaviour cluster:
.....Avoid Obstacle
.....Keep Moving
.....Recover Stall

-- Step: 200, state: #[Marco state X: 14.80m, Y: 1.81m, TH: 256.0]

&& Task succeeded, Sequence it!
&& Deleting sequence it
-- Current behaviour cluster:
.....Avoid Obstacle
.....Keep Moving
.....Recover Stall
```