A CONTROL SYSTEM USING BEHAVIOUR HIERARCHIES AND NEURO-FUZZY APPROACH

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF MIDDLE EAST TECHNICAL UNIVERSITY

BY

DİLEK ARSLAN

IN PARTIAL FULLFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE

IN

COMPUTER ENGINEERING

JANUARY 2005

Approval of the Graduate School of Natural and Aplied Sciences,

Prof. Dr. Canan Özgen
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. Ayşe Kiper
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Ferda N. Alpaslan
Supervisor

Examining Committee Members

Prof. Dr. Mehmet Tolun	(Çankaya Univ.)	
Assoc. Prof. Dr. Ferda Nur Alpaslan	(METU)	
Prof. Dr. Adnan Yazıcı	(METU)	
Assist. Prof. Dr. Halit Oğuztüzün	(METU)	
Dr. Ayşenur Birtürk	(METU)	

ABSTRACT

A CONTROL SYSTEM USING BEHAVIOUR HIERARCHIES AND NEURO-FUZZY APPROACH

Arslan, Dilek

M.Sc., Department of Computer Engineering

Supervisor: Assoc. Prof. Dr. Ferda Nur Alpaslan

December 2004, pages 59

In agent based systems, especially in autonomous mobile robots, modelling the environment and its changes is a source of problems. It is not always possible to effectively model the uncertainity and the dynamic changes in complex, real-world domains. Control systems must be robust to changes and must be able to handle these uncertainties to overcome this problem.

In this study, a reactive behaviour based agent control system is modelled and implemented. The control system is tested in a navigation task using an environment, which has randomly placed obstacles and a goal position to simulate an environment similar to an autonomous robot's indoor environment. Then the control system was extended to control an agent in a multi-agent environment. The main motivation of this study is to design a control system, which is robust to errors and easy to modify. Behaviour based approach with the advantages of fuzzy reasoning systems is used in the system.

Keywords: Behaviour hierarchy, behaviour-based robotics, neuro-fuzzy systems, autonomous navigation.

ÖZ

BULANIK MANTIK VE DAVRANIŞ SIRADÜZENİ KULLANAN BİR KONTROl SİSTEMİ

Arslan, Dilek

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi: Doc. Dr. Ferda Nur Alpaslan

Aralık 2004, 59 sayfa

Etmen tabanlı sistemlerde, özellikle özerk devingen robotlarda, ortamın modellenmesi ve ortamda meydana gelen değişiklikler bir sorun kaynağıdır. Karmaşık, gerçek dünya alanlarında belirsizlikleri ve dinamik değişimleri etkili şekilde modellemek her zaman mümkün değildir. Bu problemin üstesinden gelmek için kontrol sistemleri değişikliklere karşı dirençli olmalı ve belirsizlikleri ele alabilmelidir.

Bu çalışmada, tepkisel davranış tabanlı bir kontrol sistemi modellenmiş ve geliştirilmiştir. Bu kontrol sistemi gelişigüzel yerleştirilmiş engeller ve bir hedef nokta içeren ve bir özerk robotun kapalı bir alandaki çalışma ortamını taklit eden bir ortamda bir gezinim görevinde test edilmiştir. Daha sonra bu kontrol sistemi çok etmenli bir ortamdaki bir etmeni kontrol etmek üzere genişletilmiştir. Bu çalışmanın temel güdüsü hatalara karşı dirençli ve değiştirmesi kolay bir kontrol sistemi tasarlamaktır. Sistemde bulanık muhakemeden faydalanan davranış tabanlı bir yaklaşım kullanılmıştır.

Anahtar Kelimeler: Davranış sıradüzeni, davranış tabanlı robotbilim, nöral-bulanık sistemler, özerk gezinim.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In recent years, agent based systems gained great interest. This is because of their promise as a paradigm for conceptualising, designing and implementing systems. Agent based systems are being used in a variety of application domains such as software engineering, intelligent search techniques, and robotics.

Since the growing interest in agent based systems, many methods were developed for controlling autonomous intelligent agents, which are widely used for problem solving in Artificial Intelligence (AI). These methods can be categorized as deliberative and reactive approaches.

Deliberative approach, which is the classical way of controlling autonomous agents, relies on global planning method. A deliberative agent decides on which actions to take, by considering information about previous experiences and an overall goal as well as information from its current perception/situation. Deliberative agents build up maps of their environments and use them to plan routes to make their moves, and remember whether previous actions in a certain situation helped to achieve its goal or not. Using all of this information, the deliberative agent can calculate a plan of action, which it believes will most efficiently achieve the agent's goal.

However, deliberative approach has some drawbacks. For example, in a dynamic environment, some of the information, which the agent remembers from a previous experience, will be inconsistent with the way the world around it is currently. If a task is highly structured and predictable it makes sense to use a deliberative approach. For example, if an intelligent agent is embedded in an entirely virtual environment, then it is often possible to encode every aspect of the environment with some semantic representation. But in complex, real-world domains where uncertainty cannot be effectively modelled, agents must have a means of reacting to an infinite number of possibilities.

In reactive approach, actions of the agent are based completely on the changes of its environments. Reactive agents don’t use planning or internal models of the environment. Instead, they respond to apperception of the real world around them by using stimuli-response mechanism. Thus, in reactive approach, there is a direct connection between agent’s inputs and actions. This causes the main drawback of reactive approach; uncertain inputs lead reactive agents into wrong actions.

A behaviour-based approach that was proposed by Brooks [1] suggests a robust solution for perception errors and uncertainties of the environment for autonomous mobile robots, which are considered as reactive agents in Brooks’ work. Brooks’ approach is called subsumption architecture and became very popular in robot control since its proposal. In this approach, agent’s overall task is divided into smaller independent behaviours. For example, in a navigation problem, overall task can be divided into goal-search and avoid-obstacle behaviours. So these two smaller tasks can be handled separately.

Use of fuzzy logic [9] for dealing with uncertainties is also proved to be useful in recent years [2, 3, 4, 5, 6]. Fuzzy inference systems, unlike classical inference systems, can express human expert knowledge naturally without a need for an analytical model of the system. Since they don’t need exact mathematical models, fuzzy inference systems are powerful tools to be used in uncertain and not completely known environments.

In fuzzy inference systems, there is not always expert knowledge exist to find the proper rules and membership functions. To solve this problem, hybrid methods like neuro-fuzzy systems and genetic-fuzzy systems were proposed. Some examples of these hybrid systems are [7, 8, 11, 12, 13, 14]. These systems combine the advantages of fuzzy logic and neural networks.

In this study, a reactive behaviour based agent control system is modelled and implemented. The control system is tested for a navigation task in an environment, which has obstacles and a goal position, similar to an autonomous robot’s environment. Goal of the agent is to find and reach the goal position while avoiding obstacles and trying to follow a path as close as possible. As a second phase, the control system is extended to a multi-agent domain were the agents’ goal are to search a goal as well as avoiding obstacles and other agent(s). Agents are supposed to stay as far from each other as possible so that the agents can search different parts of the area and find the goal faster. The system uses a neuro-fuzzy system called Adaptive Network Fuzzy Inference System (ANFIS) for the rule bases of the behaviours. ANFIS is applied in its offline-learning mode. No planning or global world modelling has been used. Behaviour hierarchies proposed by Tunstel [4] was used for the behaviour coordination.

The thesis is organized as follows. Chapter 2 gives the background about behaviour-based robotics, fuzzy control in robotics, neuro-fuzzy systems, and genetic-fuzzy systems. Chapter 3 gives details of neuro-fuzzy systems and ANFIS. Chapter 4 introduces design of the agent control system proposed in this thesis and gives results of the system. Chapter 5 concludes the thesis and gives future work.

CHAPTER 2

BACKGROUND

2.1 Behaviour Based Agent Control

Autonomous agents are used in many application domains such as software development, web services, robotics, decision support systems, etc. In the environments they operate, they sometimes must be able to consider multiple concurrent requirements and make their decisions in real-time.

To achieve these complex tasks they're given, agents must have some level of intelligence and ability to learn. Classical AI spent decades trying to model human-like intelligence, using knowledge-based systems that processed representation at a high, symbolic level. Symbolic representation was considered as the most important problem because it allowed agents to operate on sophisticated human concepts and report on their action at a linguistic level. Since the goal of early AI was to produce human-like intelligence, researchers used human-like approaches. Early researchers believed an intelligent machine should, like a human, first build a model of its environment and then explore solutions abstractly before enacting strategies in the real world. This emphasis on symbolic representation and planning had a great effect on agent control strategies. Although many of the strategies developed were both elaborate and elegant, the problem was that the intelligence in these systems belonged to the designer. The agent itself had little or no autonomy and often failed to perform if the environment changed.

Traditional way used to divide operations of an agent into a series of functional modules like, getting input from the environment, processing this input, planning the next action, and executing the result. Application of this approach, which is also called deliberative approach, to an autonomous mobile robot is given in Figure 1. A deliberative agent decides its actions by considering information from all of its sensors, together with information about previous experiences and overall goals for the agent. All things are considered before an action is invoked. A deliberative agent can build up maps of its environment and use these to plan routes around obstacles, and remember whether previous actions in a certain situation helped to achieve its goal or not. Using all of this information, the deliberative agent can calculate a plan of action, which it believes will most efficiently achieve its goal.

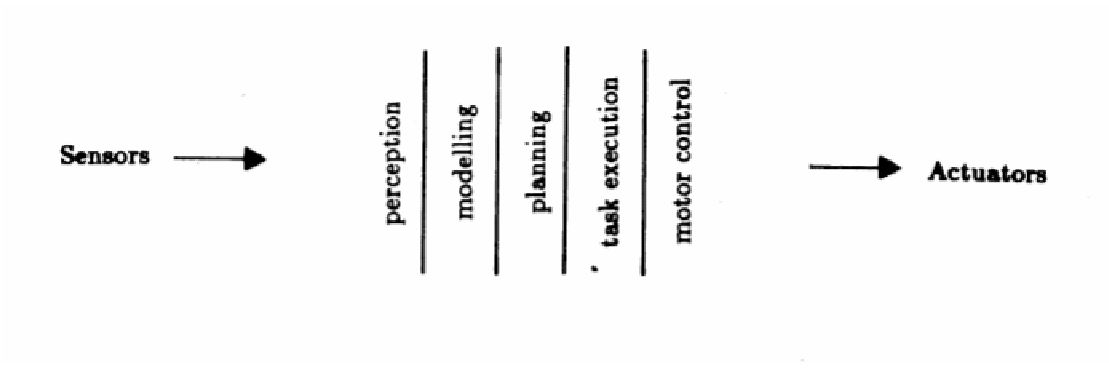

Figure 1-Traditional Decomposition of a Mobile Robot Control System (adopted from [1])

This approach of modelling the environment and making plans for the next actions is useful if prior knowledge about the environment is complete, changes in the environment are rare, and possibilities used for planning are finite. But in real world problems, agent generally must dial with a dynamically changing environment. For example, in case of autonomous mobile robots, prior knowledge about the environment is, in general, incomplete, uncertain, and approximate. Perceptually acquired information is also typically noisy and incomplete. [5]

To solve this problem, Brooks proposed a different decomposition of robot's task [1]. In his work, Brooks decomposed tasks of a robot into task achieving behaviours instead of functional units as in the classical Sense-Model-Plan-Act (SMPA) systems. In this architecture, behaviours are arranged in layers and each layer has a level of competence. Higher levels of competence imply higher priority behaviours. Higher-level layers subsume the roles of lower level layers when they want to take control. The subsumption architecture, offers a robust way to deal with environment changes and a flexible way to add new behaviours. The layered control system introduced by Brooks is shown in Figure 2. "Avoid obstacle" behaviour shown at the bottom has the lowest priority and "Reason about behaviours of objects" behaviour shown at the top of the layers has the highest priority.

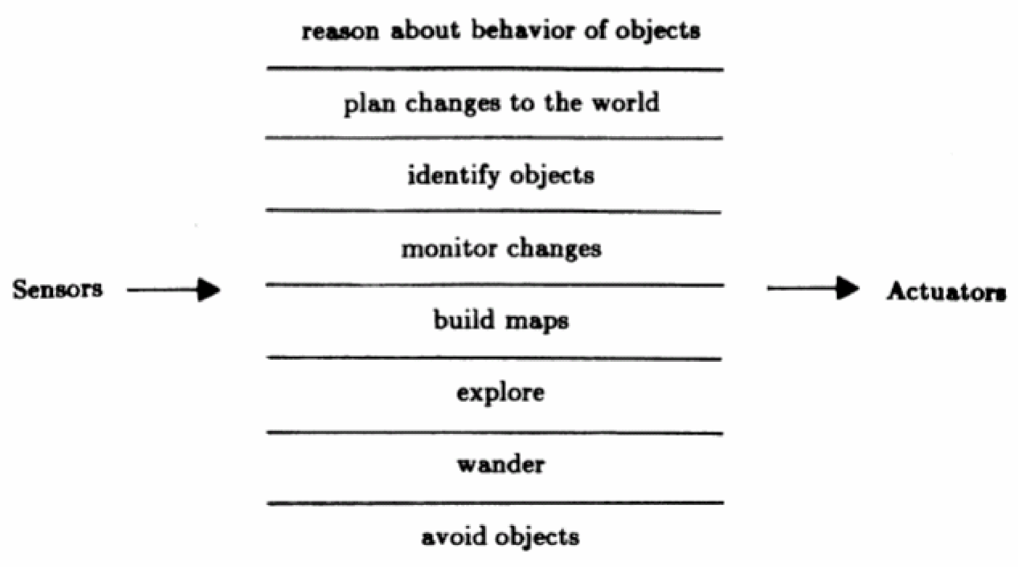

Figure 2- Subsumption architecture (adopted from [1])

Similar to Brooks' approach Minsky [10] claimed that, mind is made of many smaller processes called agents. According to Minsky, these agents are as simple as possible and each perform a small part of a complex task. When we join these agents in societies, this leads to intelligence. Minsky's this approach is well suited to behaviour-based agent control systems where agent's task is divided into simpler and smaller tasks. These tasks can be thought as Minsky's small processes/agents.

2.2 Behaviour Coordination

Behaviour based approaches, which offers to divide the complex tasks into many separate and independent behaviours, brought along an important problem: How to coordinate these simultaneously working behaviours and produce an overall coherent behaviour that achieves the intended task?

Since introduction of behaviour based approach in mid eighties, this problem becomes a major one and still many researcher works on this topic. Behaviour coordination architectures can be divided into two categories: behaviour arbitration and command fusion schemes.

Arbitration scheme is based on competition. In this architecture, in each step one behaviour is selected and this selected dominant behaviour solely controls the robot in that step. Outputs of other behaviours are completely ignored. Arbitration architectures can also be divided into two categories: Static arbitration policies and dynamic arbitration policies.

Early solutions, like subsumption architecture of Brooks [1], relied on a static arbitration policy, hard-wired into a network of suppression and inhibition links. But this static method is not enough to handle changes in the environment of an autonomous mobile robot. So, most current architectures use dynamic arbitration schemes where the decision of which behaviour to activate depends on the current plan and on the environmental changes. In dynamic schemes, methods like voting and action-selection are used. In voting method, behaviours vote for a predefined set of actions and the action receiving the highest vote is applied. In action-selection method, the activation level of each behaviour is determined according to agent's goals and inputs. Behaviour, which has the highest activation level, is applied in each step. This scheme is shown in Figure 3.

When number of behaviours increase, there are several criteria, which should be taken into account to accomplish a task. In this case, simply selecting one of the behaviours as the dominant one as in the arbitration method is not suitable. To solve this problem, command fusion schemes aggregate the control actions of multiple concurrently active behaviours into a consensual decision. The most popular approaches of command fusion uses vector summation scheme. Output of each behaviour is represented by a force vector and outputs from different behaviours are combined by vector summation. Output of the robot becomes the resulting force vector. So, outputs of all behaviours are fused and combined.

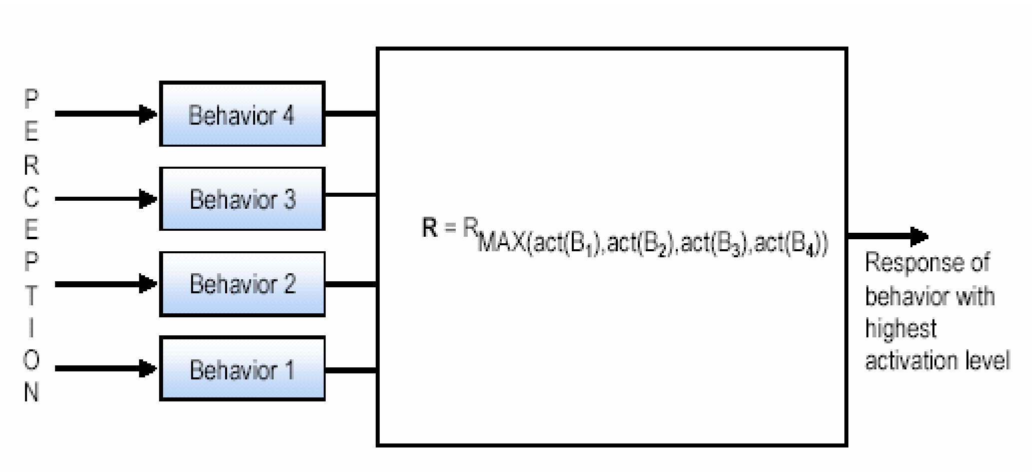
PERCEPTION
Behavior 4
Behavior 3
Behavior 2
Behavior 1
$R = R_{MAX(act(B_1),act(B_2),act(B_3),act(B_4))}$
Response of behavior with highest activation level

Figure 3- Arbitration via action-selection scheme (adopted from [21])

But vector summation is not always sufficient and does not contain enough information. To address this problem, many fuzzy behaviour coordination techniques were proposed in literature. Some of these solutions contain hybrid approaches like neuro-fuzzy systems and genetic-fuzzy systems. These approaches will be covered in sections 2.4, 2.5, and 2.6.

2.3 Behaviour Hierarchies in Robotics

Since the autonomous mobile robots are supposed to act in non-engineered, and constantly changing real-world environments, there are many uncertainties about the environment and often it is not possible to model or quantify this uncertainty.

Because of these uncertainties, it is difficult, if not impossible, to obtain a precise mathematical model of the robot's interaction with its environment in autonomous navigation. The lack of precise and complete knowledge about the environment limits the applicability of conventional control system design to the domain of autonomous robotics. What is needed are intelligent control and decision-making systems with the ability to reason under uncertainty.

It is unrealistic to assume that any learning algorithm is able to learn a complex robotic task, in reasonable learning time starting from scratch without prior knowledge about the task or the environment.

Task complexity can be reduced by a divide and conquer approach, which attempts to break down the overall problem into more manageable subtasks. This course is advocated by hierarchical behaviour architectures, in that they separate the design or adaptation of primitive behaviours from the task of learning a supervisory policy for behaviour coordination [4, 12, and 15].

Controlling agents by using behaviour hierarchies by Tunstel [4], like many other works, is basically inspired by Brooks' work and is parallel to Minsky's ideas. In this reactive approach, main idea is to divide a robot's task into a finite number of task-achieving behaviours and arrange these behaviours as a hierarchical network of distributed rule bases each responsible from a different part of the overall task.

There are two types of behaviours in the hierarchy: primitive and composite types of behaviours. Primitive behaviours are, at the bottom of the hierarchy and they are simple and self-contained behaviours which servers a single purpose. Primitive behaviours are independent from other behaviours and they focus on a part of the complex task. These behaviours act in a reactive and reflexive fashion. For example, in a navigation task, obstacle avoidance can be considered as a primitive behaviour.

Only primitive behaviours themselves wouldn't be sufficient enough to perform a complex task. They need coordination among them. Composite behaviours are used for behaviour modulation. A composite behaviour control two or more primitive behaviours and decide how true it is to let them affect the overall result of the agent. For example, in a navigation task, goal seeking can be considered as a composite behaviour and it may control primitive behaviours such as “go to a given coordinate” and “avoid obstacles”.

For behaviour modulation, composite behaviours use a concept called degree of applicability (DOA), which is a weighted control decision-making concept [4, 6]. Composite behaviours produce degree of applicability values for each primitive behaviour they control. These DOA values are a measure of instantaneous level of activation of primitive behaviours. Outputs of each primitive behaviour are multiplied with its degree of applicability value before adding this output into the overall result. Since degree of applicability values are used as percentages for desirability of the corresponding primitive behaviours, their values can only be between 0 and 1.

DOA values are determined dynamically for each step of the given complex task. This feature allows primitive behaviours to influence the overall behaviour to a greater or lesser degree as required by the current situation and goal. It serves a form of adaptation since it causes the control policy to dynamically change in response to goal information and inputs taken from the agent’s environment.

An example of behaviour hierarchies proposed by Tunstel is shown in Figure 4. It shows hierarchical decomposition of a robot’s indoor navigation task.

Behaviours “go-to-xy”, “avoid-collision”, “wall-follow”, and “doorway” are primitive behaviours. The other tree behaviours on the higher levels are composite behaviours. The lines connecting the behaviours indicate the dependencies between them. Circles between primitive behaviours and composite behaviours show DOAs of associated primitive behaviours. Composite behaviour “goal-seek” uses outputs of primitive behaviours “go-to-xy” and “avoid-collision” to accomplish task of going to the goal position without colliding any obstacles. Composite behaviour “routefollowing" uses outputs of all four primitive behaviours to accomplish task of going to a given coordinate (a sub-goal) without colliding any obstacles and by following any walls met towards the sub-goal and pass through the doorways.

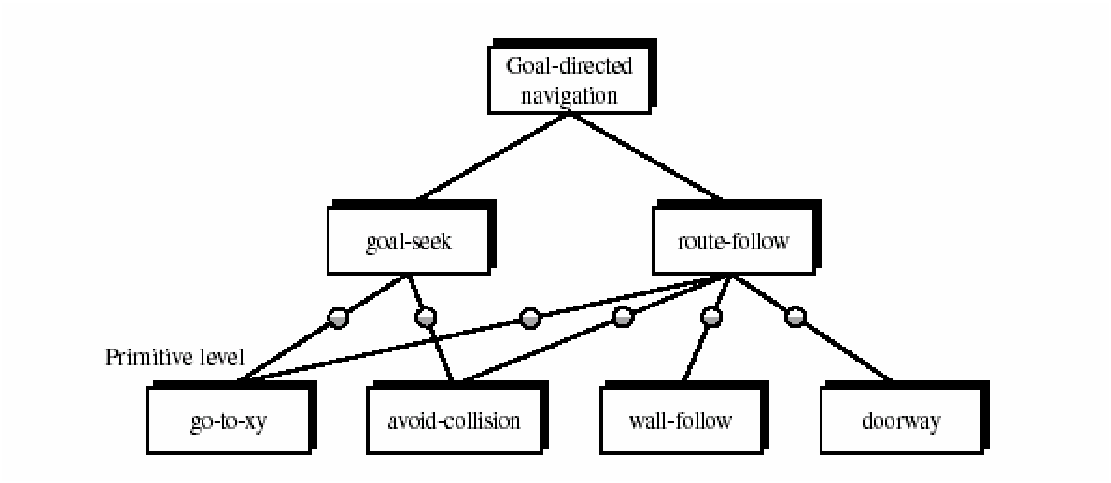

Figure 4- Mobile robot behaviour hierarchy (Adopted from [4])

Behaviour hierarchies can easily be extended to work in a multi-agent domain by adding some behaviour to the hierarchy for coordination and communication with the other agents.

2.4 Fuzzy Logic for Robot Control

One of the main problems in robotics is as explained above, the changes in dynamic environments and complexity of modelling the environment. There are also other problems like sensor errors, limited ranges of sensors, poor observation conditions, and impossibility of modifying/simplifying the environment for robot's needs. These problems cause imprecise and incomplete information.

To handle this erroneous information, traditional work in robotics tried to carefully design sensors, and engineer the environment. Some additions to the environment were made to ease the robot's task. But this solution increased the cost and was impossible to apply all domains. For ease of use of robots, environment should not be intervened. So the main problem becomes to produce robust robot control approaches which can handle incomplete information and which can be applied to other domains.

Fuzzy logic, introduced by Zadeh in mid sixties [9], was used to implement robust control techniques against incomplete and uncertain information. By using fuzzy logic, need for an exact model of the world was surpassed and because of its qualitative nature, fuzzy logic gave good results in control applications.

In fuzzy logic, unlike classical set theory, there are no crisp boundaries. Instead of a value's being a member of a set or not, in fuzzy logic there are degrees of membership. These degrees are determined using functions called a membership function. Outputs of memberships functions can be in the interval [0,1]. This output is a measure of degree of similarity of an instance to the fuzzy set as a numerical value. Fuzzy sets are described in linguistic terms such as "high", "low", "near", far, etc.

A generic fuzzy if-then rule is defined as follows;

If x is A then u is B

Where x and u represent input and output fuzzy linguistic variables respectively. A and B are fuzzy sets representing linguistic values of x and u. An example fuzzy if-then rule is given below:

If obstacle is close then velocity is small

This type of fuzzy if-then rules are known as Mamdani-type of fuzzy rules [31, 32].

Another form of fuzzy if-then rules is Takagi-Sugeno type of rules [18]. This type of fuzzy rules has fuzzy sets only in their premise parts. A generic Takagi-Sugeno type of fuzzy rule is given below;

If x is A then u is f(x)

Where x is input fuzzy linguistic variable and A is a linguistic label characterized by an appropriate membership function as in Mamdani-type of fuzzy rules. But f(x) in the consequent part of the rule is non-fuzzy equation and it is a linear or nonlinear function describing the dynamics of the system output u for the particular value A of the input x.

Both types of fuzzy if-then rules, due to their concise form, are often used to capture imprecise modes of reasoning in an environment of uncertainty and imprecision.

Fuzzy Logic Controllers (FLCs) are control systems which produce actions according to fuzzy rules based on fuzzy logic. Units of a Fuzzy Logic Controller based on Mamdani-type fuzzy rules are: fuzzifier, fuzzy rule base, fuzzy inference engine, and defuzzifier. This structure is shown in Figure 5.

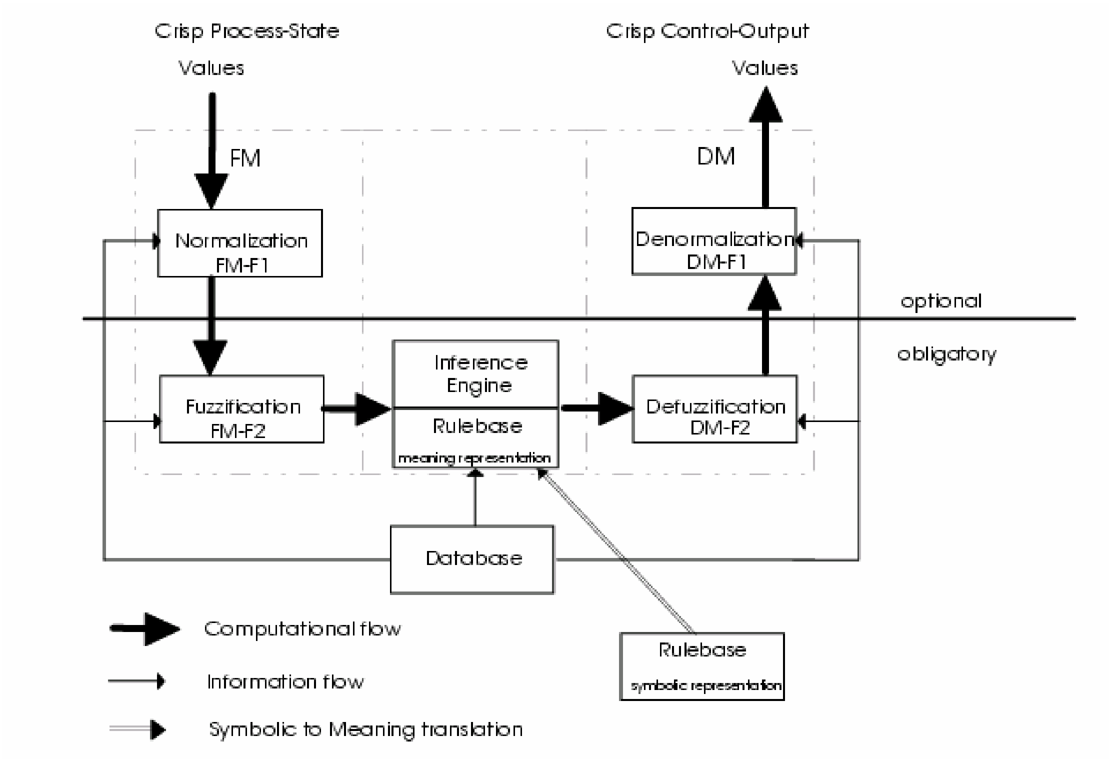

Figure 5-Structure of Fuzzy Logic Controllers (adopted from [17])

Normalization and denormalization modules are optional parts. Normalization is used to map the physical values of the current input variables into normalized domain. Denormalization maps the normalized value of the output variable onto its physical domain. When a non-normalized domain is used, there's no need for normalizaton and denormalization modules.

In fuzzifier module, fuzzification is performed which converts a crisp value of an input variable into a fuzzy set using membership functions, in order to make it compatible with the fuzzy set representation of the input variable in the rule's premise part.

The rule base consists of membership functions of fuzzy if-then rules, which specifies behaviour of the system. Inference engine module maps fuzzy input sets to fuzzy output sets. In design of inference engines there are two basic types:

(1) Composition based inference,

(2) Individual rule-based inference:.

Defuzzifier module performs defuzzification, which converts the set of modified output values (fuzzy results) into a single crisp value.

Fuzzy logic controllers were first used for single primitive behaviour implementations. Some early examples are; model car of Sugeno and Nishida which tracks a path delimited by two walls [19] and Takeuchi's fuzzy controller for obstacle avoidance [20]. It is not enough to perform a simple behaviour to accomplish a complex task. To perform autonomous navigation, a robot should take multiple objectives into account.

Fuzzy logic controllers are typically designed to consider one single goal. If we want to design a fuzzy logic controller, which takes several goals into consideration, there are two ways;

i) One set of complex rules are written. Antecedents of these rules consider all goals simultaneously. This type of rule sets are called monolithic rule sets.

ii) Rule set is divided into number of goals. Each subsets contains rules concerning only a single sub goal. Outputs of these subsets must be combined somehow.

Choosing one of these options is difficult. First solution should be preferred if interactions between the sub goals are important. But as the number of sub goals increase, number of rules tends to grow exponentially and rule set becomes intracable. Second solution has also an important problem; if we choose the second solution, we must also find a way to combine the outputs of the rule subsets.

Fuzzy logic is also used for coordination of behaviour, in other words; for combining outputs of independent rule subsets. Behaviour coordination methods in literature are mentioned in section 2.2. Fuzzy logic has been used to implement both static and dynamic arbitration schemes. The first example of static fuzzy arbitration scheme is work of Berenji et al. [22]. Dynamic arbitration scheme for fuzzy logic can be implemented using context rules. Fuzzy context rules are used to arbitrate behaviours, which are themselves implemented by fuzzy logic.

One of the first examples of command fusion by using fuzzy logic is Ruspini's work [23]. In fuzzy command fusion, each behaviour is seen as an agent and they express preferences as to which command to apply. These preferences are represented as fuzzy sets. Fuzzy operators are used to combine these preferences and produces a collective preference.

While producing a single preference, combining the individual results by using for example vector summation and using the collective preference from fuzzy logic operators can give different results. Figure 6 shows an example to this situation. Combining the individual decisions does not satisfy behaviour B2 in the first case. But taking collective preferences into account satisfies both behaviours B1 and B2 in the second case. This shows that fuzzy command fusion is fundamentally different from vector summation and it may produce decisions that better satisfy all the behaviours.

2.5 Neuro-Fuzzy Systems

Neuro-fuzzy systems incorporate the knowledge representation of fuzzy logic with the learning capabilities of artificial neural networks. The power of neural networks comes from the distributed processing capability of a large number of computationally simple elements. In contrast fuzzy logic is closer related to reasoning on a higher level. Pure fuzzy systems do not possess the capabilities of learning, adaptation or distributed computing that characterize neural networks. On the other hand, neural networks lack the ability to represent knowledge in a manner comprehensible to humans, a key feature of fuzzy rule based systems. Neuro-fuzzy systems bridge the gap between both methodologies, as they synthesize the adaptation mechanisms of neural networks with the symbolic components of fuzzy inference systems, namely membership functions, fuzzy connectives, fuzzy rules and aggregation operators.

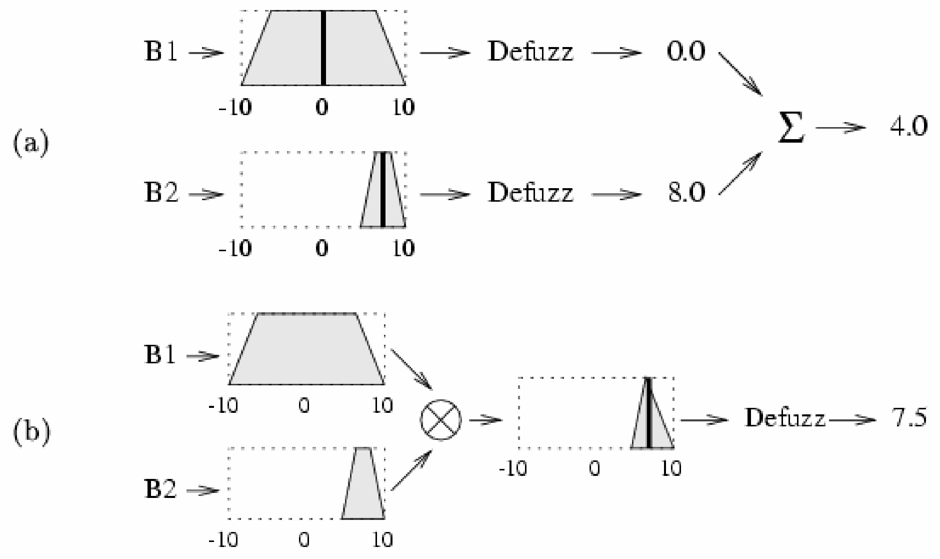

Figure 6-Two approaches of command fusion (adopted from [16])

Two examples of neuro-fuzzy systems on behaviour-based robotics are; the systems in Ahrns et al [24] and in Godjavec et al [25]. Ahrns et al apply neuro-fuzzy control to learn collision avoidance behaviour. Their approach relies on reinforcement learning for behaviour adaptation. The learner incrementally adds new fuzzy rules as learning progresses and simultaneously tunes the membership functions. Godjavec et al present a neuro-fuzzy approach to learn obstacle avoidance and wall-following behaviour on a small size robot. Their scheme allows it to seed an initial behaviour with expert rules, which are refined throughout the learning process. During training the robot is controlled either by a human or a previously designed controller. The recorded state-action pairs serve as training examples during supervised learning of neuro-fuzzy control rules.

2.6 Genetic-Fuzzy Systems

Several authors proposed evolutionary algorithms for learning and tuning of fuzzy controllers for robotic behaviours [26, 27, 28, 29]. In a genetic fuzzy system the evolutionary algorithm evolves a population of parameterise fuzzy controllers. Candidate controllers share the same fuzzy inference mechanism, but establish different input-output mappings according to their genetically encoded knowledge base. The evolutionary algorithm adapts all or part of the components that constitute the knowledge base, namely membership functions, scaling factors and fuzzy rules. The same mechanism is used for the adaptation of primitive behaviours and learning of supervisory fuzzy controllers. The performance of the fuzzy controller while it governs the behaviour of the robot is described by a scalar fitness function. Those fuzzy controllers that demonstrate behaviours of higher fitness compared to their competitors are selected as parents for reproduction. Novel candidate behaviours are generated through recombination and mutation of parent fuzzy rules and membership functions. The cycle of selection, reproduction and fitness evaluation progressively leads to improved fuzzy controllers that enable the robot to achieve the behaviour design goals in its environment.

Two examples of genetic-fuzzy systems are in Tunstel et al [3] and Hagras et al [27]. Tunstel et al apply genetic programming for off-line identification of supervisory fuzzy rules that coordinate primitive fuzzy behaviours [3]. Composite behaviours are evaluated according to their success in orchestrating the primitive behaviours such that they eventually navigate the robot to its goal location. The fitness is averaged over several trials in different simulated environments in order to obtain robust and reliable behaviours. The generalization capability of the highest scoring behaviours is tested in a more general environment unrelated to the test environments used during evolution. The off-line evolved behaviours are transferred to the physical robot for verification. The robot successfully navigates within close proximity to the goal. Hagras et al present a fuzzy classifier system that utilizes a genetic algorithm for online-learning of a goal seeking and a wall following behaviour. The fuzzy classifier system maintains a rule cache to store suitable fuzzy rules, that might be of benefit in future situations and which serve to seed the initial population when a new genetic learning process is invoked. This technique substantially speeds up the learning process, thus making the entire approach feasible for on-line learning of robotic behaviours.

CHAPTER 3

ADAPTIVE NETWORKS AND ANFIS

3.1 Introduction

An adaptive network is a multi-layered network structure, which consists of nodes and directional links through which the nodes are connected. A part of these nodes are adaptive; that is, outputs of these nodes depend on parameters of these nodes. A learning rule is used to specify how these parameters should be changed to minimize error of the output [7].

To produce an output, each node in an adaptive network uses a function called node function. Node function can be different for each node or each layer of the adaptive network. Links in an adaptive network have no weights associated to them. They only show flow direction of signals. An example of adaptive networks is given in Figure 7.

Two types of nodes are shown in the figure; circular and square nodes. Circular nodes have no parameters and they are called fixed nodes. Square nodes have parameters whose value change during learning process, so they are called adaptive nodes.

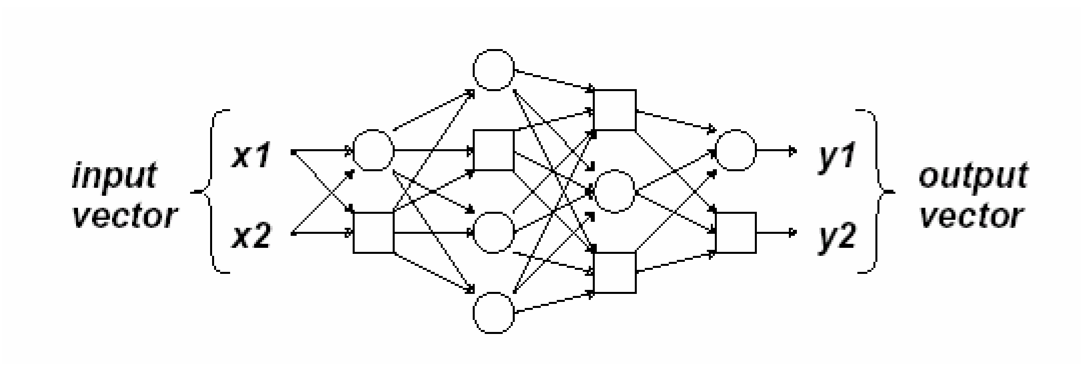

Figure 7- An example adaptive network (adopted from [7])

Basic learning rule of adaptive networks is based on gradient decent algorithm and chain rule, which was proposed by Webos [30] in 1970s.

3.2 ANFIS (Adaptive-Network-Based Fuzzy Inference System)

ANFIS is a fuzzy inference system implemented in the framework of adaptive networks by using a hybrid learning procedure. ANFIS was proposed by Jang [7] in 1993. By using the hybrid learning method of neural networks and fuzzy inference systems, ANFIS constructs an input-output mapping based on human knowledge (by using fuzzy if-then rules which captures human knowledge easily) and stipulated input-output data pairs.

Architecture of an ANFIS network is up to which type of reasoning system it uses. Types of fuzzy reasoning systems are explained in the following section.

3.2.1 Fuzzy Reasoning System Types

Several types of fuzzy reasoning have been proposed in the literature. Depending on the fuzzy if –then rules and types of fuzzy reasoning they use, these systems can be classified into three types.

In Type-1 systems, the overall output is the weighted average of each rule’s crisp output induced by the rule’s firing strength and output membership functions. The output membership functions used in this scheme must be monotonically non-decreasing.

In Type-2 systems, Mamdani type of fuzzy if-then rules are used. In these systems, the overall fuzzy output is derived by applying "max" operator to the fuzzy outputs. These fuzzy outputs are equal to the minimum of firing strength and the output membership function of each rule. To produce the final output, many schemes have been proposed. Some of these are center of area, mean of maxima, and bisector of area, etc. [31, 32].

In Type-3 systems, Takagi and Sugeno's type of fuzzy if-then rules [18] are used. Output of each rule is a linear combination of input variables plus a constant term. The final output is the weighted average of each rule's output.

Three types of fuzzy reasoning systems are shown in Figure 8 below.

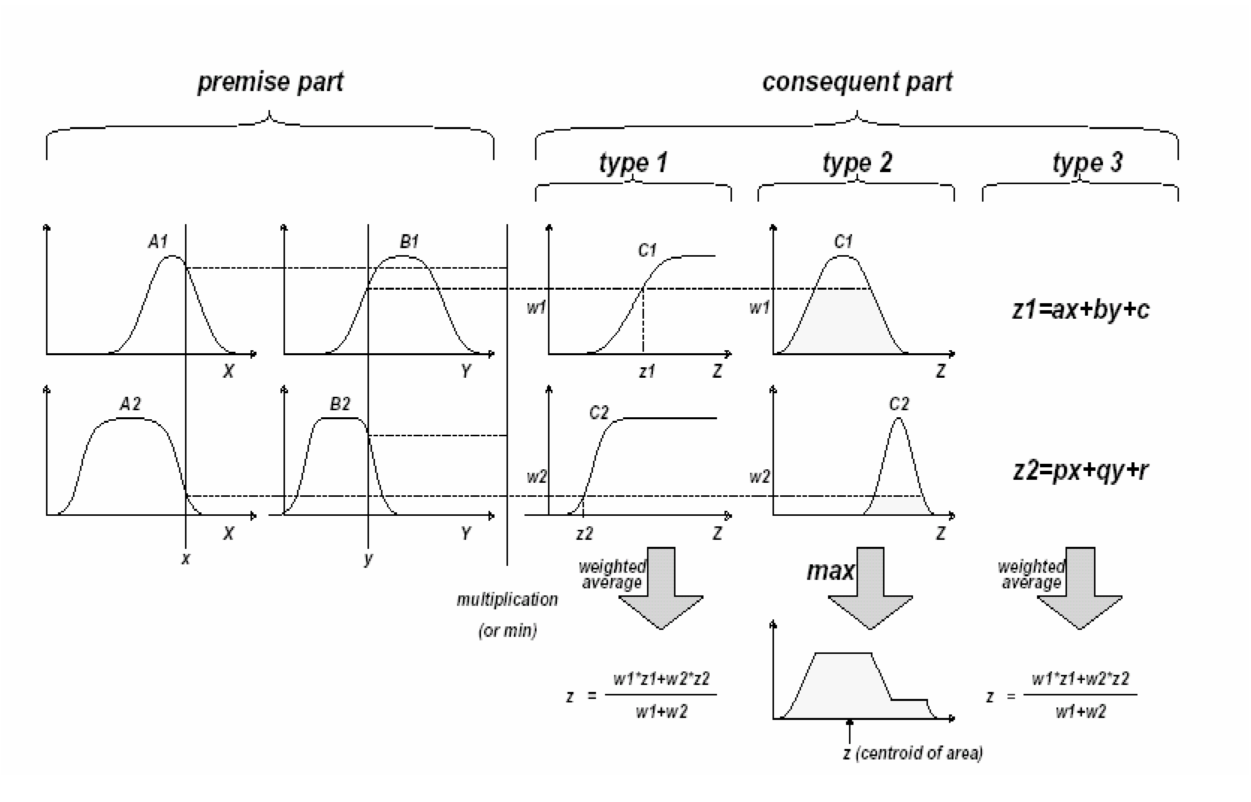

Figure 8- Types of fuzzy reasoning systems (adopted from [7])

3.2.2 ANFIS Architecture

ANFIS is a feed-forward network whose nodes are connected through weightless links. Some of the nodes in an ANFIS network are adaptable which has adaptable parameters. The other type of node in ANFIS architecture are fixed nodes which have no adaptable parameters. Example ANFIS architecture for Type-3 fuzzy reasoning systems is shown in Figure 9. Adaptive nodes are shown as square nodes and fixed nodes are shown as circular nodes in the figure.

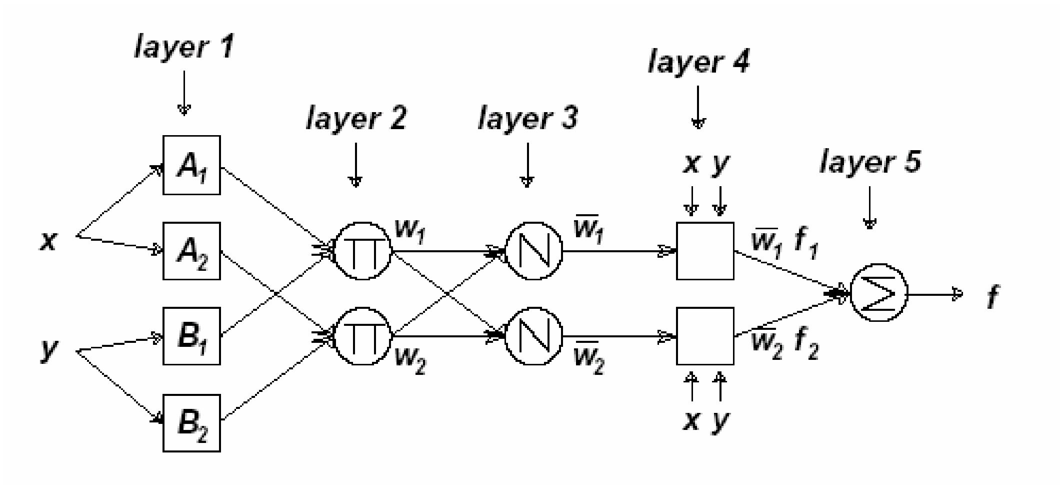

Figure 9-An example of Type-3 ANFIS architecture (adopted from [7])

ANFIS network shown in Figure 9 has two inputs (x and y) and one output (f). An ANFIS network has five layers and the ANFIS network in Figure 9 has two fuzzy rules of type Takagi and Sugeno rules [18]. ANFIS networks have five layers. Node functions in the same layer are the same function family. The layers in ANFIS networks are defined as follows:

Layer1 This layer represents membership functions used in the fuzzy rules of the network. Number of nodes in this layer is equal to number of inputs times number of rules. Every node in this layer is a square node with a node function:

$$O_i^1 = \mu A_i(x) \tag{1}$$

where x is the input to node *i*. A_i is the linguistic label (near, far, etc) associated with the node function of node *i*. The output of the node (o_i^1) is the membership function of A_i and it specifies the degree to which the given x satisfies the quantifier A_i. Membership function $\mu A_i(x)$ is generally chosen as a bell-shaped function with minimum equal to 0 and maximum equal to 1, such as:

$$\mu A_i(x) = \frac{1}{1 + \left[\left(\frac{x - c_i}{a_i} \right)^2 \right]^{b_i}} \tag{2}$$

where a_i, b_i, and c_i are parameters of node *i*. As the values of these parameters change, the bell-shaped membership functions change as well. Parameters in this layer are referred to as premise parameters.

Layer 2 Every node in Layer 2 are circular fixed nodes. Output of each node in this layer represents firing strength of a rule. Number of nodes in this layer is equal to number of rules in the network. Nodes in Layer 2 multiplies the incoming signals and sends the product out. Output of nodes in this layer are calculated using Equation 3 for the ANFIS shown in Figure 9.

$$O_i^2 = w_i = \mu A_i(x) * \mu B_i(y), i = 1,2 \tag{3}$$

Layer 3 Number of nodes in this layer is equal to number of rules in the network. i-th node in this layer calculates the ratio of the i-th rule's firing strength to the sum of all rules' firing strengths. Outputs of nodes in this layer are calculated using Equation 4 for the ANFIS shown in Figure 9.

$$O_i^3 = \overline{w}_i = \frac{w_i}{w_1 + w_2}, i = 1,2 \tag{4}$$

Outputs of Layer 3 are called normalized firing strengths.

Layer 4 Every node in this layer are adaptable nodes with parameters. Nodes in this layer calculate outputs of each rule. Node function of the nodes in Layer 4 is;

$$O_i^4 = \overline{w}_i f_i = \overline{w}_i\left(p_i x + q_i y + r_i\right) \tag{5}$$

where $\overline{w}_i$ is output of Layer 3 and p_i, q_i, and r_i are the parameter set of this layer. Parameters in this layer are called consequent parameters.

Layer 5 This layer has a single fixed node. It computes the overall output as the summation of all incoming signals;

$$O_1^5 = \sum_i \overline{w}_i f_i = \frac{\sum_i w_i f_i}{\sum_i w_i} \tag{6}$$

Architecture and node functions were given above for Type-3 fuzzy inference sytems. For Type-1 and Type-2 fuzzy inference systems, the architecture needs some extensions. For Type-1 fuzzy inference systems, the change is straightforward. A Type-1 ANFIS is shown in Figure 10 where the output of each rule is induced jointly by the output membership function and the firing strength. For Type-2 fuzzy inference systems, we need to replace the centroid defuzzification operator with a discrete version, which calculates the approximate centroid of area. Then Type-3 ANFIS can be constructed accordingly. But it is more complicated than Type-1 and Type-3 ANFIS networks.

Type-1 ANFIS shown in Figure 10 has two inputs (x and y) and two fuzzy rules in its database as in Type-3 ANFIS shown in Figure 9.

3.2.3 Learning Algorithm

ANFIS uses a hybrid learning algorithm which combines the gradient method and least squares estimates (LSE) to identify parameters of the nodes in the network.

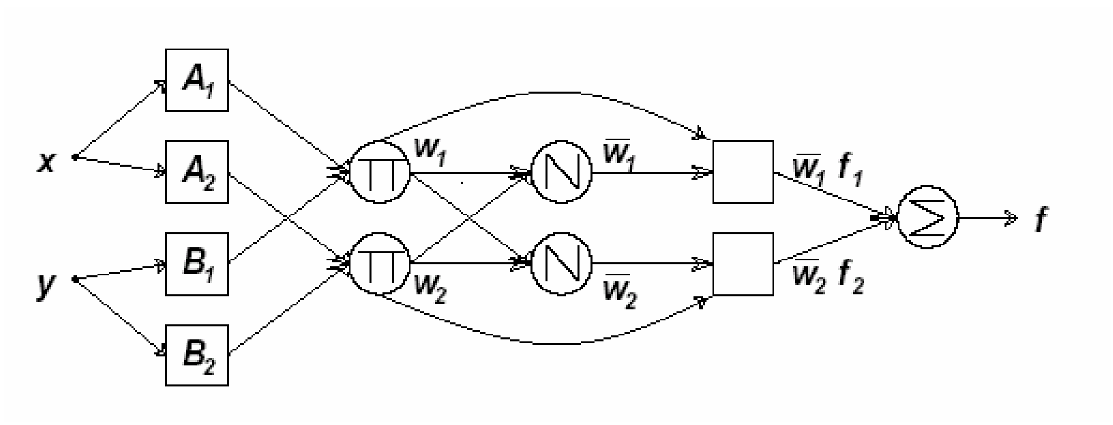

Figure 10- A Type-1 ANFIS network (adopted from [7])

ANFIS networks have only one output, which can be written as;

$$\text{Output} = F(\vec{I}, S) \tag{7}$$

Where $\vec{I}$ is the set of input variables and S is the set of all parameters in the ANFIS network. If there exist a function H such that the composite function H o F is linear in some of the elements of S, then these elements can be identified by the least squares method. If the parameter set S can be decomposed into two sets such as;

$$S = S_1 \oplus S_2 \tag{8}$$

where $\oplus$ shows direct sum and H o F is linear in the elements of S_2. So the Equation 7, which is linear in the elements of, S_2 becomes;

$$H(\text{output}) = H \text{ o } F(\vec{I}, S) \tag{9}$$

Given values of elements in S_1, training data can be added into Equation 9. Let number of these training data be P. Then the matrix equation below is obtained;

$$AX = B \tag{10}$$

Where X is an unknown vector whose elements are parameters in S_2 and size of S_2 is M. Since, number of training data (P) is usually greater than number of linear parameters (M), generally there is no exact solution to Equation 10. Instead, a least squares estimate of X, X^* is found to minimize the squared error. Equation for X^* is as follows;

$$X^* = \left(A^T A\right)^{-1} A^T B \tag{11}$$

where $\left(A^T A\right)^{-1} A^T$ is pseudo inverse of A if $A^T A$ is non-singular. Because it deals with matrix inverse, Equation 11 is expensive to compute and it becomes ill-defined when $A^T A$ is singular. So, sequential formulas to compute the LSE of X are used, which is more efficient. For writing these sequential formulas, let the *i*th row vector of matrix A be a_i^T and the *i*th element of B be b_i^T. Then X can be calculated iteratively using the sequential formulas;

$$\begin{aligned} X_{i+1} &= X_i + S_{i+1} a_{i+1} \left(b_{i+1}^T - a_{i+1}^T X_i\right) \\ S_{i+1} &= S_i - \frac{S_i a_{i+1} a_{i+1}^T S_i}{1 + a_{i+1}^T S_i a_{i+1}}, i = 0,1,....,P-1 \end{aligned} \tag{12}$$

where S_i is called covariance matrix and the least square estimate (LSE) X^* is X_P. Initial conditions of Equation 12 are, $X_0 = 0$ and $S_0 = \gamma \mathrm{I}$ where γ is a positive large number and I is the identity matrix of dimension MxM.

After gathering these equations, a hybrid of gradient method and the least squares estimate is applied in each epoch. This procedure is composed of a forward pass and a backward pass. In the forward pass, input data goes forward to calculate each node's output until matrices A and B in Equation 10 are obtained and the parameters in S_2 are identified by the sequential least squares formulas in Equation 12. After identifying parameters in S_2, input signals keep going forward until the error measure is calculated.

In the backward pass, the error rates propagate from the output end towards the input end, and the parameters in S_1 are updated by the gradient method explained below.

Assuming the given training data set has P entries, error measure for the *p*th entry of the training data is the sum of squared errors;

$$E_P = \sum_{m=1}^{\#(L)} \left(T_{m,p} - O_{m,p}^{L}\right)^2 \tag{13}$$

where #(L) represents number of layers in the network, $T_{m,p}$ is the *m*th component of *p*th target output vector, and $O_{m,p}^{L}$ is the *m*th component of actual output vector produced by the ANFIS network. Hence the overall error measure is; $E = \sum_{p=1}^{P} E_P$

In order to implement a learning procedure which implements gradient decent in E over the parameter space, first the error rate $\frac{\partial E_P}{\partial O}$ for *p*th training data and for each node output O is calculated. The error rate for the output node *i* at layer L is calculated as follows;

$$\frac{\partial E_P}{\partial O_{i,p}^{L}} = -2\left(T_{\mathrm{i,p}} - O_{i,p}^{L}\right) \tag{14}$$

For an internal node in in any layer of the network, error rate can be calculated by using the chain rule. The error rate of an internal node can be expressed as a linear combination of the error rates of the nodes in the next layer.

If α is a parameter of the ANFIS network, we can write an equation such that;

$$\frac{\partial E_P}{\partial \alpha} = \sum_{O^* \in S} \frac{\partial E_P}{\partial O^*} \frac{\partial O^*}{\partial \alpha} \tag{15}$$

where S is the set of nodes whose output depends on α. Then the derivative of the overall error measure E with respect to α is;

$$\frac{\partial E}{\partial \alpha} = \sum_{p=1}^{P} \frac{\partial E_P}{\partial \alpha} \tag{16}$$

Accordingly, the update formula for the generic parameter α is;

$$\Delta\alpha = -\eta \frac{\partial E}{\partial \alpha} \tag{17}$$

where η is learning rate which can be formulated as;

$$\eta = \frac{k}{\sqrt{\sum_{\alpha} \left(\frac{\partial E}{\partial \alpha} \right)^2}} \tag{18}$$

where k is the step size, the length of each gradient transition in the parameter space. The value of k varies the speed of convergence. In ANFIS, some heuristic rules are used for changing k. These rules are shown in Figure 11;

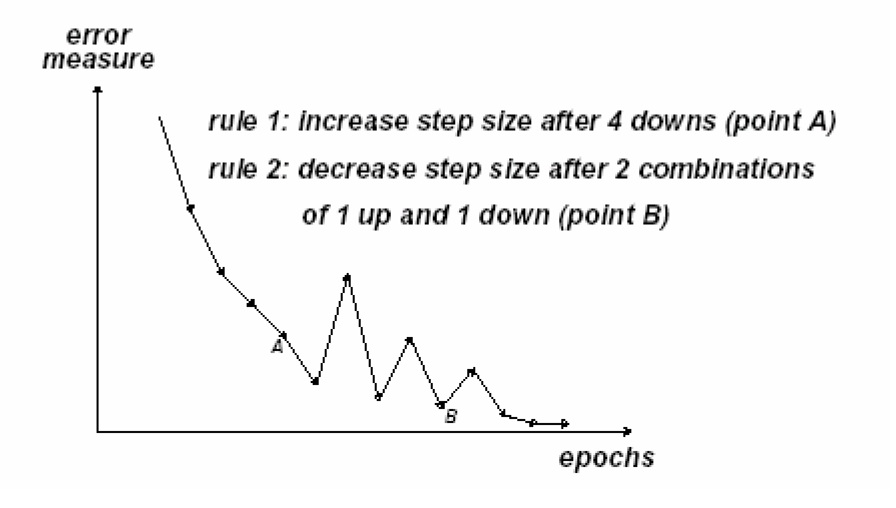

Figure 11-Two heuristic rules for updating value of k (adopted from [7])

CHAPTER 4

IMPLEMENTATION

4.1 Problem

Over the past few years, research in autonomous mobile robots gained an interest due to various tasks they're used. First solutions proposed were mostly based on numerical methods, but in recent years, many fuzzy logic based methods were proposed for controlling mobile robots.

In this thesis, behaviour hierarchies and a hybrid learning method of neural networks and fuzzy inference systems are combined to implement an autonomous agent control method. This method obtains the advantages of fuzzy systems, numerical systems, and provides flexible control architecture.

An off-line learning system is modelled and implemented. It was tested in both static and dynamic environments. In the test environment, obstacles and a goal area was located to simulate a robot's working environment. To simulate the real world environments and robustness of the method, error was added to the inputs of the agent. As a second phase, the method implemented was tested in a multi-agent environment. The other agent in this phase was also forming a dynamic obstacle.

Main motivation of this study is to design an agent control system, which is robust, easy to train and flexible to add new behaviours for new tasks and environments.

4.2 Implementation Details

4.2.1 Behaviour Hierarchy For Single Agent Control

As the first phase of the thesis, a control method for a single agent was implemented. Task of the agent is to reach a given goal position while avoiding obstacles on its way and following the shortest path to the goal as close as possible. Behaviour hierarchy used to achieve this task is given in Figure 12.

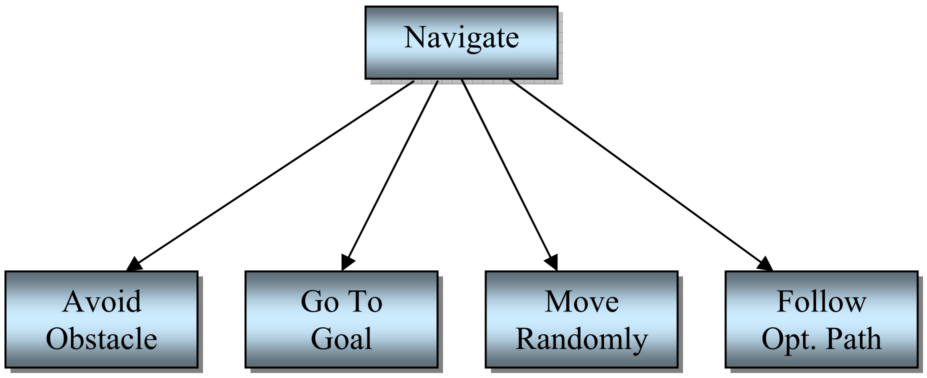

Figure 12- Behaviour hierarchy for controlling single agent

Behaviours in the hierarchy are explained below;

Avoid Obstacle: This behaviour has three inputs: distance from the closest obstacle on the left, distance from the closest obstacle on the right, and distance from the closest obstacle in front. Obstacle Avoidance behaviour tends to go to the direction where obstacle distance is the farthest.

Go To Goal: Inputs to this behaviour are: goal distance on the left, goal distance on the right, and goal distance in front. This behaviour tries to go to the direction where goal distance is the smallest.

Follow Optimum Path: Inputs of this behaviour are: distance from the optimum path on the right, on the left, and in front. *Follow Optimum Path* behaviour, as its name implies, tries to follow the optimum path as close as possible.

Move Randomly: This behaviour does not use any learning technique. It simply produces a random speed and direction for the next movement. It is used when the agent gets stuck somewhere and cannot move.

All four behaviours explained above are simple primitive behaviours, which deal with only a single goal. For example, *Go To Goal* does not care if there are obstacles in the direction it chooses to go or *Avoid Obstacle* does not know if it gets closer to the goal or not while trying to escape an obstacle.

Since these behaviours only consider their own simple goals, another more complex behaviour is needed to coordinate them. In the hierarchy given above, the composite behaviour, which coordinates and controls them, is *Navigate* behaviour explained below.

Navigate: Composite behaviour *Navigate* controls four primitive behaviours by finding their appropriate DOAs in each step of the execution such that the agent moves towards the goal without hitting obstacles and follows the optimum path towards the goal. To take into consideration the overall task, *Navigate* behaviour uses all the information about obstacle distances, goal distances, and distances from the optimum path.

This behaviour can be thought as four parts, each controlling a primitive behaviour. For controlling *Avoid Obstacle* behaviour, inputs used are the result produced by behaviour *Avoid Obstacle*, obstacle distance in the direction where *Avoid Obstacle* intends to go, and distance from the goal position in the current position. This part of the behaviour tries to produce a Degree of Applicability (DOA) value for the *Avoid Obstacle* primitive behaviour such that DOA increases when an obstacle is close and decrease when the goal is close.

Second part of the *Navigate* behaviour controls DOA value of *Go To Goal*. Inputs of this part are: the result produced by the behaviour *Go To Goal*, obstacle distance in the direction where *Go To Goal* intends to go, and distance from the goal location in the current position. DOA value for the *Go To Goal* behaviour tends to increase when the goal is close and decreases when an obstacle is close.

Third part of the behaviour controls *Move Randomly* behaviour and produce its DOA value. Inputs of this part are: distances between the current position and two steps ago, four steps ago, and six steps ago. DOA of *Move Randomly* behaviour tends to increase if these distances get smaller.

Fourth and the last part of the *Navigate* behaviour controls, DOA of *Follow Optimum Path*. Inputs of it are: the result produced by *Follow Optimum Path* behaviour, obstacle distance in the direction where *Follow Optimum Path* wants to go, and distance from the goal location in the current position.

All the primitive and composite behaviours defined above, except *Move Randomly* primitive behaviour, uses ANFIS architecture to learn and accomplish their tasks. Learning mechanism used for the single agent architecture is explained in the following sections.

4.2.2 Definition of the Environment

Environment used to test the method is a square board whose height and width are 100 units. It has 10x10-sized obstacles placed randomly around the board and a goal location. An example board with obstacles is shown in Figure 13 below.

Orange coloured areas on the board represent obstacles. Walls of the board are also accepted as obstacles. Green area on the lower left corner is the goal position. Dashed line, which passes through the diagonal, shows the optimum path to the goal position if the agent's initial position is upper left corner.

Upper left corner of the board has coordinates (0, 0) and coordinates of the lower right corner are (100, 100).

4.2.3 Learning Mechanism for Single Agent Control

For learning all the primitive and composite behaviours in the hierarchy, except *Move Randomly* behaviour, ANFIS learning architecture is used in off-line learning mode. In primitive behaviours, ANFIS learns to produce a speed and direction to achieve the task of that behaviour. In composite behaviours, ANFIS networks learn to produce DOA values of the primitive behaviours for each step of the execution.

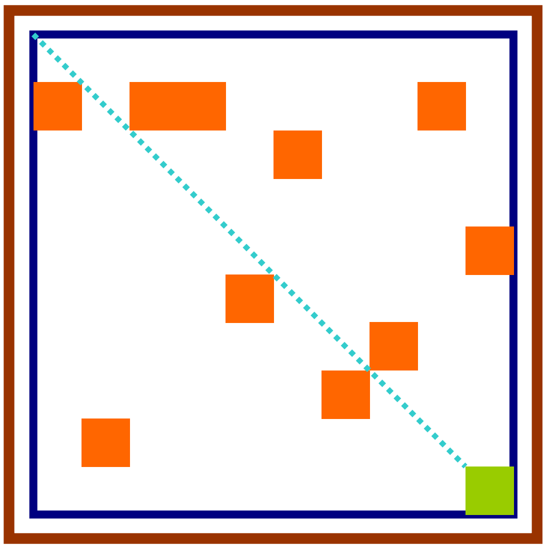
Figure 13- An example environment

In the training phase, all primitive and composite behaviours are trained separately. But since the composite behaviour *Navigate* uses outputs of the primitive behaviours as its input, primitive behaviours are trained first and their outputs in the last epoch of training is saved to be used by the composite behaviour *Navigate* later.

ANFIS has just one output node. Output value produced by this node is used as both the speed and direction value for that behaviour. Sign of the output value produced by the ANFIS network shows the direction ANFIS wants to go. If the sign is negative, the new direction becomes right. If the sign is positive, direction becomes left. If the result produced by ANFIS is zero then the direction does not change. Absolute value of the result is used as the speed value. When the output is zero, a default value of 2 is used as the speed. Figure 14 shows how the output of the ANFIS network is interpreted.

4.2.3.1 Obstacle Avoidance

ANFIS used in this behaviour has three input nodes. These inputs are obstacle distance from the left of the current position, obstacle distance from the right, and obstacle distance in front.

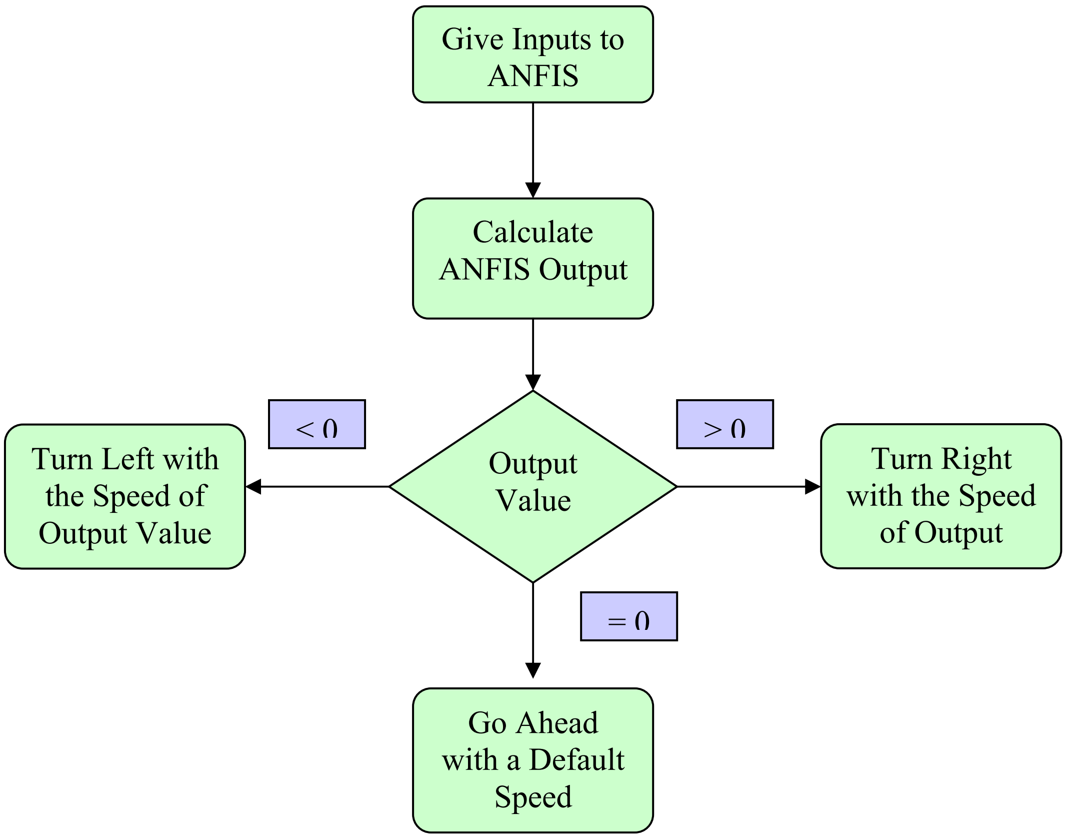
Give Inputs to ANFIS
Calculate ANFIS Output
Output Value
< 0
> 0
= 0
Turn Left with the Speed of Output Value
Turn Right with the Speed of Output
Go Ahead with a Default Speed

Figure 14- Interpretation of ANFIS outputs

In the training phase, the agent is placed in random coordinates on a board with obstacles placed randomly on it and obstacle distances in three directions are measured and given to the ANFIS network as inputs. The network is trained to choose the direction with the biggest obstacle distance. While training the ANFIS network, the desired output is given to the network as the distance from the farthest obstacle and a sign is added according to the farthest obstacle direction. For example, if the farthest obstacle is on the right and its distance is 1.8 units from the current position, then the desired output is –1.8. If the farthest obstacle is in front of the agent, then the desired output is given as zero to the ANFIS network.

If obstacle distances on two directions are the same, the ANFIS network is trained to favour right side on the left and front on the right. Since *Avoid Obstacle* is a primitive behaviour, as long as the agent moves away from the obstacles, it does not take into consideration if the agent moves towards the goal or not.

The Type-1 ANFIS network used for the *Avoid Obstacle* behaviour is shown in Figure 15.

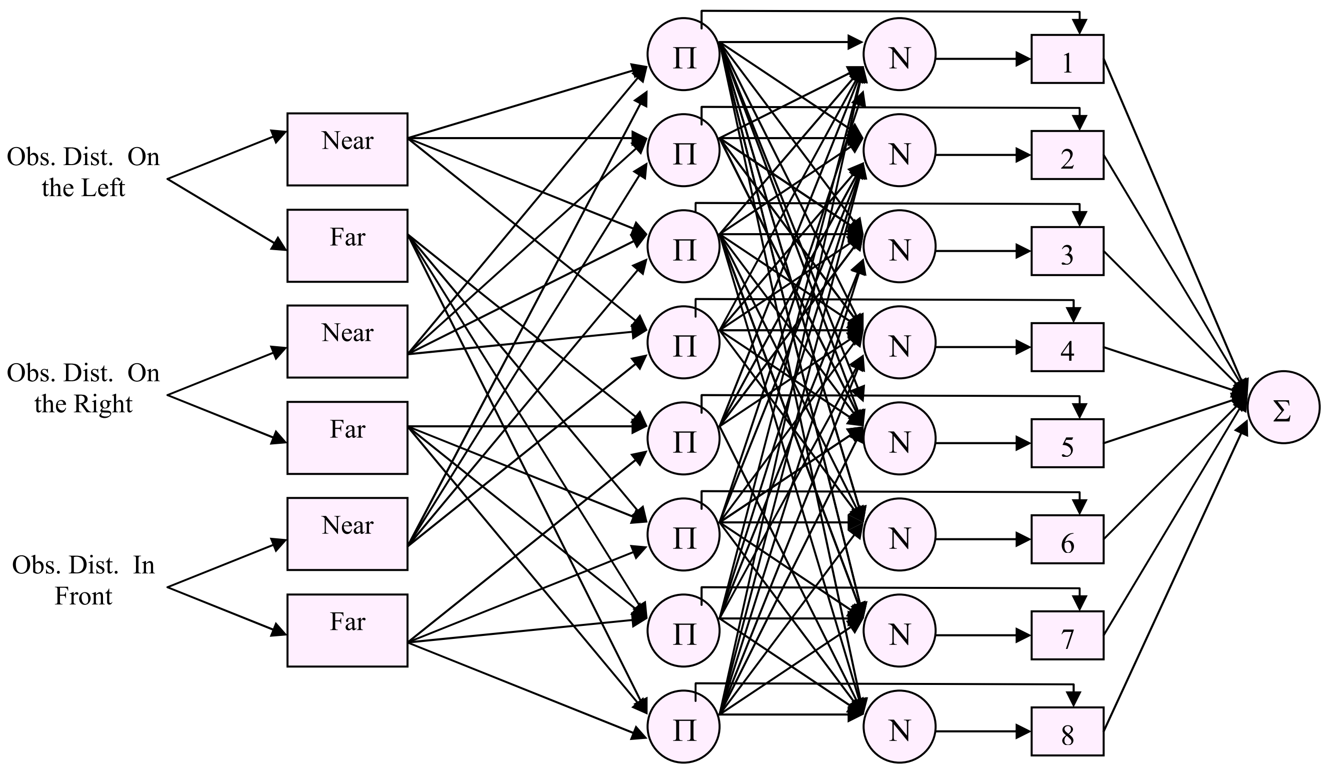

Figure 15- ANFIS architecture for Avoid Obstacle behaviour

4.2.3.2 Go To Goal

ANFIS architecture used in this behaviour has also three input nodes. These inputs are: goal distance if the agent moves one unit left from the current position, goal distance if the agent moves one unit right, and goal distance if the agent moves straight ahead one unit.

In the training phase, the agent is placed in random coordinates on a board without obstacles on it since this primitive behaviour does not take the obstacles into consideration. Goal distances in three directions are measured and given to the ANFIS network as inputs. The network is trained to choose the direction with the smallest goal distance. While training the ANFIS network, the desired output is given to the network as the distance of the closest goal and a sign is added according to the direction of this smallest goal distance. If the closest goal is in front of the agent, then the desired output is given as zero to the ANFIS network.

If the goal distances on two directions are the same, the ANFIS network is trained to favour right side on the left and front on the right. Since *Go To Goal* is a primitive behaviour, as long as the agent moves closer to the goal, it does not take into consideration if the agent moves closer to an obstacle also or not.

The Type-1 ANFIS network used for the *Go To Goal* behaviour is the same in Figure 15 except the meanings of the inputs. Inputs of *Go To Goal* behaviour are: goal distances on the left, on the right, and in front.

4.2.3.3 Follow Optimum Path

Optimum path is the straight line from the agent's beginning position to the given goal location.

ANFIS architecture used in this behaviour has also three input nodes. These inputs are distance from the optimum path if the agent moves one unit left from the current position, distance from the optimum path if the agent moves one unit right, and distance from the optimum path if the agent moves straight ahead one unit. So the agent is not given the whole coordinates of the optimum path but it is only given the distance from it. This behaviour can be thought as a robot, which tries to follow a line on the floor.

In the training phase, the agent is put in random coordinates on a board without obstacles like the *Go To Goal* behaviour. Distances from the optimum path in three directions are measured and given to the ANFIS network as inputs. The network is trained to choose the direction with the smallest distance from the optimum path. While training the ANFIS network, the desired output is given to the network as the smallest distance from the optimum path and a sign is added according to the direction of the smallest optimum path distance. If the smallest optimum path distance is when the agent goes straight, then the desired output is given as zero to the ANFIS network.

Similar to the other primitive behaviours, if the distances on two directions are the same, the ANFIS network is trained to favour right side on the left and front on the right. Like the other primitive behaviours, as long as the agent moves closer to the optimum path, it does not take into consideration if the agent moves closer to an obstacle or goes away from the given goal position.

The Type-1 ANFIS network used for the *Follow Optimum Path* primitive behaviour is the same in Figure 15 except the meanings of the inputs. Inputs of *Follow Optimum Path* behaviour are: optimum path distances on the left, on the right, and in front.

4.2.3.4 Navigate

Since the ANFIS architecture has just one output and the composite behaviour *Navigate* needs to produce a proper DOA value for each of the four primitive behaviours, this behaviour contains four ANFIS networks to control the primitive behaviours. DOA values are positive numbers in the range of [0,100] and they represent percentage of properness of a primitive behaviour in each step of the execution. DOA values are produced dynamically in each step. Architectures of each of these networks are explained below.

The first ANFIS network learns to produce DOA values for the *Obstacle Avoidance* behaviour. It has, like the primitive behaviours explained above, three inputs. These inputs are absolute value of the output produced by *Obstacle Avoidance* behaviour, obstacle distance in the direction of this output, and goal distance in the current position. Architecture of the ANFIS used is the same as the one given in Figure 15. Membership functions of these inputs are as follows:

- *Avoid Obstacle* Output: Big, Small
- Obstacle Distance: Near, Far
- Goal Distance: Near, Far

The network learns to produce small DOA values if the obstacle distance is far. DOA increases as the obstacle distance gets nearer. When an obstacle is near, *Avoid Obstacle* becomes more dominant because of the big DOA value and affects the overall output more.

The network is trained to take the goal distance into consideration too, so that the agent does not go away from the goal while trying to avoid an obstacle. If the goal distance is near, then the DOA gets smaller so that, output of the *Avoid Obstacle* affects the overall output less when the goal is near.

The second ANFIS network learns to produce DOA values for the *Go To Goal* behaviour. It also has three inputs. These inputs are absolute value of the output produced by *Go To Goal* behaviour, obstacle distance in the direction of this output, and goal distance in the current position. Architecture of the ANFIS used is the same as the one given in Figure 15. Membership functions and the rules learned by the network is quite similar to the one explained above which controls *Obstacle Avoidance* behaviour. Membership functions of these inputs are as follows:

- *Go To Goal* Output: Big, Small
- Obstacle Distance: Near, Far
- Goal Distance: Near, Far

The network learns to produce small DOA values if the goal distance is far. DOA increases as the goal distance gets nearer. When the goal location is near, *Go To Goal* becomes more dominant because of the big DOA value and affects the overall output more.

The network is trained to take the obstacle distance into consideration too, so that the agent does not hit an obstacle while trying to reach the goal. If the Obstacle Distance is near, then the DOA gets smaller so that, output of the *Go To Goal* affects the overall output less when an obstacle is near.

If the output value produced by the *Go To Goal* behaviour is bigger than the obstacle distance, DOA value gets smaller to decrease the effect of *Go To Goal* on the overall output and prevent the agent to hit an obstacle.

The third ANFIS network learns to produce DOA values for the *Follow Optimum Path* behaviour. Its inputs are absolute value of the output produced by *Follow Optimum Path* behaviour, obstacle distance in the direction of this output, and distance from the optimum path in the current position. Architecture of the ANFIS used is the same as the one given in Figure 15. Membership functions are also similar to the first two networks explained above. Membership functions of these inputs are as follows:

- *Follow Optimum Path* Output: Big, Small
- Obstacle Distance: Near, Far
- Optimum Path Distance: Near, Far

Unlike the first two ANFIS networks, this network learns to produce bigger DOA values if the distance from the optimum path is far. DOA increases as the optimum path gets more distant. When the optimum path is far, *Optimum Path Following* becomes more dominant because of the big DOA value and affects the overall output more so that it pulls the agent towards the optimum path.

Like the first two ANFIS networks, this network is trained to take the obstacle distance into consideration too for the agent to not hit an obstacle while trying to follow the optimum path. If the obstacle distance is near, then the DOA gets smaller so that, output of the *Follow Optimum Path* affects the overall output less when an obstacle is near.

If the output value produced by the *Follow Optimum Path* behaviour is bigger than the Obstacle Distance, DOA value gets smaller to decrease the effect of *Follow Optimum Path* on the overall output and prevent the agent to hit an obstacle.

Different than the first two ANFIS networks, this ANFIS learns to produce a negative DOA value to handle the case of agent's being on the optimum path already. In this case, the network learns to produce a negative DOA value, which shows a choice to not to move and stay in the current position. This case is the only exception for the [0,100] range of DOA values. When *Navigate* behaviour produces a negative DOA value for the *Follow Optimum Path* behaviour, this is interpreted by decreasing outputs of the other behaviours by 50%.

Fourth and the last part of the *Navigate* behaviour controls the primitive behaviour *Move Randomly* by producing its DOA values for each step of the agent. Inputs of this part are distances between the current position and the positions of the agent two steps ago, four steps ago, and six steps ago. These inputs have again two membership functions: big and small. If these distances are small, DOA for *Move Randomly* behaviour increases. If the distances are big enough, the DOA value gets smaller so, Random Movement does not affect the overall output. Aim of this behaviour is to rescue the agent if it gets stuck somewhere and cannot move or move in the same places in a persistent way.

To produce a single overall output value, which represents the agent's speed and direction in every step, first absolute values of outputs of the primitive behaviours are multiplied with their DOA values. Then vector summation is used to combine these results into one single, combined result. Execution of the *Navigate* behaviour and production of the final output is shown in Figure 16.

Normally, outputs of the primitive behaviours can represent three directions; left, right, and front. They cannot choose the agent go any other directions. But since the vector summation is used to combine the results of these primitive behaviours, the agent can move in the directions between these three.

4.2.4 Experiment Results of Single Agent Architecture

Experiments were done in the environment defined in section 4.2.2 and the board used for the experiments is same as the one in Figure 13. Obstacles used in the environment are static obstacles.
All the ANFIS networks in the hierarchy are trained off-line. They are all trained in 300 epochs and 100 training data are used in each epoch. First, the primitive behaviours are trained and their behaviours in the last epoch are saved to be used by behaviour as the inputs.

While training the ANFIS networks, each epoch is divided into two phases: forward and backward phases. Training data is produced by putting the agent in random positions on a board again randomly generated (Obstacles are placed randomly).

In the first phase of the training, outputs of the first three layers of the network are calculated and consequent parameters of the network are found by using Least Squares Estimate method. In the second phase, gradient decent algorithm is used to update membership function parameters.

The agent starts the experiment from (0,0) coordinates of the board (Upper left corner). The goal location it is supposed to go is the lower right corner of the board. For the agent’s eyesight to cover all the space around it, it sees the direction it looks as shown in Figure 17.

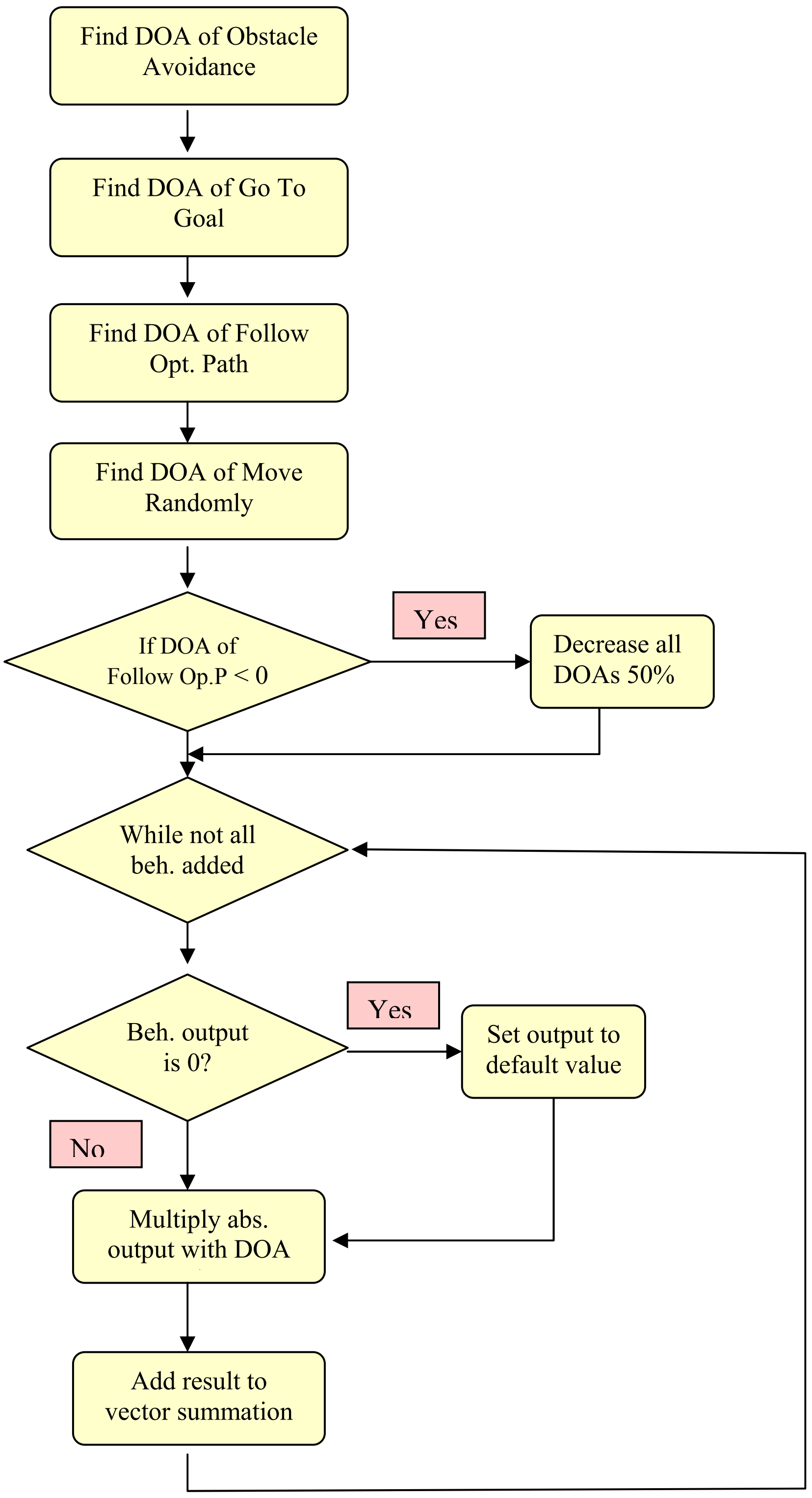
Find DOA of Obstacle Avoidance
Find DOA of Go To Goal
Find DOA of Follow Opt. Path
Find DOA of Move Randomly
If DOA of Follow Op.P < 0
Yes
Decrease all DOAs 50%
While not all beh. added
Beh. output is 0?
Yes
Set output to default value
No
Multiply abs. output with DOA
Add result to vector summation

Figure 16-Execution of Navigate behaviour

In the figure, agent faces north. Small rectangles on Figure 17 have dimensions (10, 10) units. Red, green, and blue rectangles in the figure show the eye sights of the agent in front, on the right, and on the left respectively. They have overlapping areas on the corners.

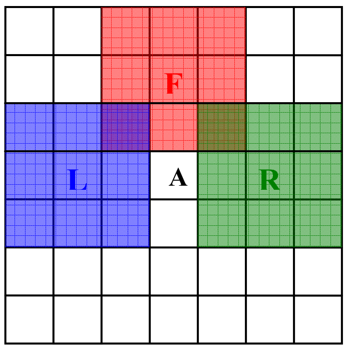

Figure 17-Eyesight of the agent

Agent looks only to its left, right and front. It does not see behind. So agent's current direction is important to determine what it sees and how it will react. Agent's direction can take four values; North, South, West, and East for the simplicity even if the agent moves to a direction between these. Agent's direction is determined as follows;

The behaviours can choose to go to the right, to the left, or straight. If result of the vector summation is something between the four main directions, one of its components must be the current direction. In this case, the new direction is accepted as the vector's component, which is not the current direction. For example, if the current direction is south and the result vector has components in the directions south and east, the new direction is determined as east.

Result of the experiment done after the training phase is shown in Figure 18.

Initial direction of the agent is given as south. The agent does not have a map of the board and its path is not pre-planned. It is given only distances from the goal position, optimum path and closest obstacles to simulate perception of a mobile robot. At the starting position, because of the *Avoid Obstacle* behaviour, the agent chooses to turn east (that is agent's left hand-side). At this point, since the agent is already on the Optimum Path, composite behaviour *Navigate* chooses to produce a negative DOA for *Follow Optimum Path* behaviour as shown in Figure 19 and stay in the current position. (Since the agent never gets stuck during the execution, DOA of *Move Randomly* behaviour is always zero and it is not shown in the graph.) As for *Go To Goal* behaviour, both going forward and left are the same but as explained above, it is trained to favour going forward in this case. So it chooses to go forward. But since the obstacle is close, the biggest DOA is *Avoid Obstacle*'s and agent turns left but it still goes forward a small amount because of *Go To Goal* behaviour.

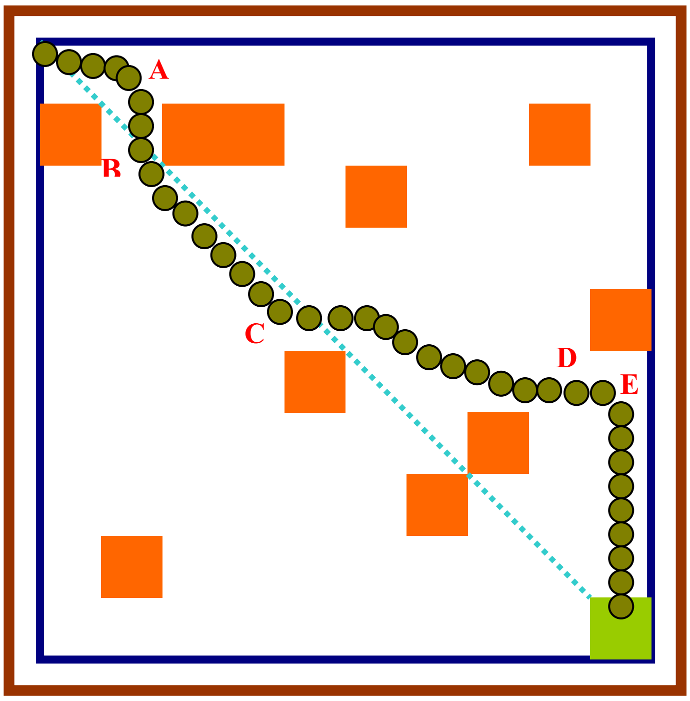

Figure 18-Experiment results of single agent

Points where the agent changes direction are marked on both the agent's path and the graph, which shows DOAs of the behaviours. Agent's behaviour at these points is explained below.

At point A, direction is east. *Avoid Obstacle* and *Follow Optimum Path* chooses to turn right, but *Go To Goal* still chooses going forward. At this point, DOA of *Follow Optimum Path* is not determined by distance from the optimum path. The distance from the obstacle in the direction where *Follow Optimum Path* wants to go determines it because output of the *Follow Optimum Path* is greater than the obstacle distance in that direction. DOA is chosen as the biggest value possible to prevent the agent hitting the obstacle. DOA of *Go To Goal* also is determined according to distance from the closest obstacle in that direction. DOA of *Avoid Obstacle* is very high because of the close obstacles.

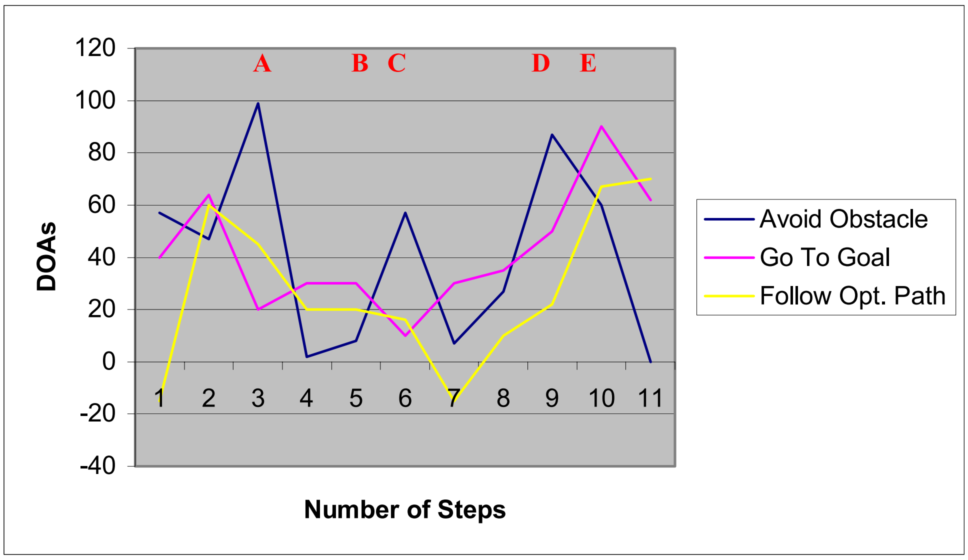

Figure 19-DOA values during the execution

At point B, direction is south. Since there are no obstacles close to the agent on the left, right, and in front, DOA of *Avoid Obstacle* is quite small. Again because there are no close obstacles, DOA of *Follow Optimum Path* is determined by the distance from the optimum path. Since the agent is not very far from the optimum path, DOA at this point is not high but it still affects the overall behaviour and causes the agent to go left. The distance from the goal also determines DOA of *Go To Goal* and it is bigger than the previous step's DOA since the agent is getting closer to the goal. *Go To Goal* behaviour causes the agent to go forward at this point.

At point C *Avoid Obstacle* becomes dominant again as the agent approaches a new obstacle. Since distance from the optimum path remains the same, DOA of *Follow Optimum Path* does not change much. DOA of *Go To Goal* is small in this step because this behaviour chooses the agent to go towards the obstacle and its DOA is determined by distance from the obstacle.

At point D again DOA of *Avoid Obstacle* increases because of decreasing distance from the obstacle. Since the goal is getting quite close, DOA of *Go To Goal* begins to increase a good deal. DOA of *Follow Optimum Path* also increases because distance from the optimum path also increases.

At point E the most dominate behaviour is *Go To Goal* since the goal is closer now. In spite of this, increasing of its DOA is slowed down, as can be seen in Figure 19, for the agent to not to hit the wall. Because the agent is getting close to the goal, DOA of *Avoid Obstacle* gets smaller. This is needed for the *Obstacle Avoidance* behaviour to not to prevent the agent from reaching the goal by moving it away from the walls of the board.

4.2.5 Experiment Results with Erroneous Data

One of the most important problems in robotics is the imprecise data taken from robots' sensors due to hardware errors, limited ranges of sensors, poor observation conditions, etc. To handle these erroneous inputs, control mechanism of the agent must be robust.

The control mechanism for the single agent architecture explained above is tested by adding 10% of error to the agent's inputs. Results of this robustness test are shown in Figure 20 below.

The same board as in the first experiment was used in the test for an easy comparison. As shown in Figure 20, agent's path stays mostly the same. The most important difference is between the points A and B. In this part, the agent follows the optimum path not as close with the precise inputs but it still goes parallel to the optimum path. Other than that it still achieves its task without hitting the obstacles on its way and following the optimum path where possible. The path it follows to achieve its task is very close to the first experiment where the agent was given error-free inputs.

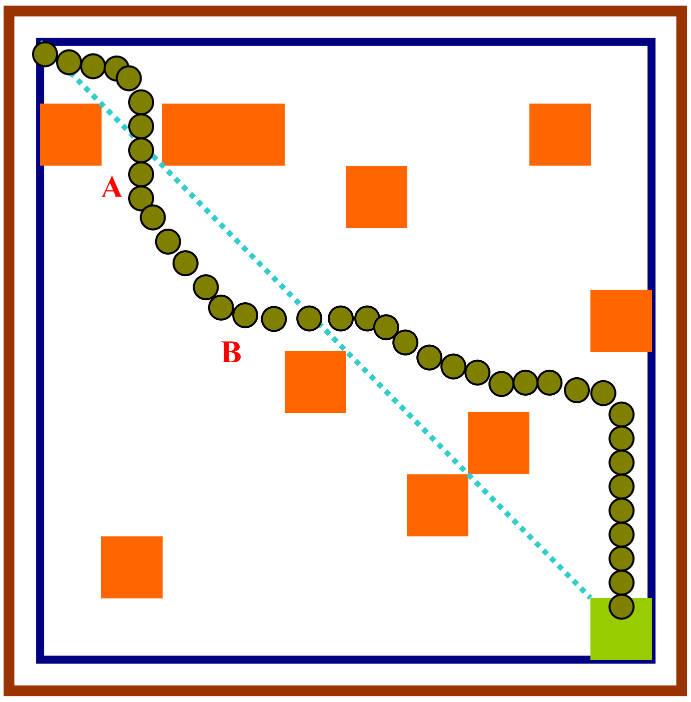

Figure 20- Experiment results by using erroneous input data

4.2.6 Behaviour Hierarchy For Multi-Agent Control

As the second phase of the thesis, the control method for a single agent explained in the previous sections was extended to control the agents in a multi-agent architecture. Task of the agents is to search the goal while avoiding obstacles on their way. This time the agents must learn to avoid the other agents too to prevent collisions and keep the agents apart so that they can search different parts of the board to find the goal. Behaviour hierarchy used is given in Figure 21.

Behaviours in the hierarchy are explained below;

Avoid Obstacle, Go To Goal: These behaviours are same as the single agent versions except a "Field of View" is added this time. Field of View is 40 units in a 100x100 board. If there are obstacles farther than 40 units or if the goal is farther than 40 units, then the agents cannot see them.

Move Randomly: This behaviour is also the same as in the first hierarchy and it produces random directions and speeds.

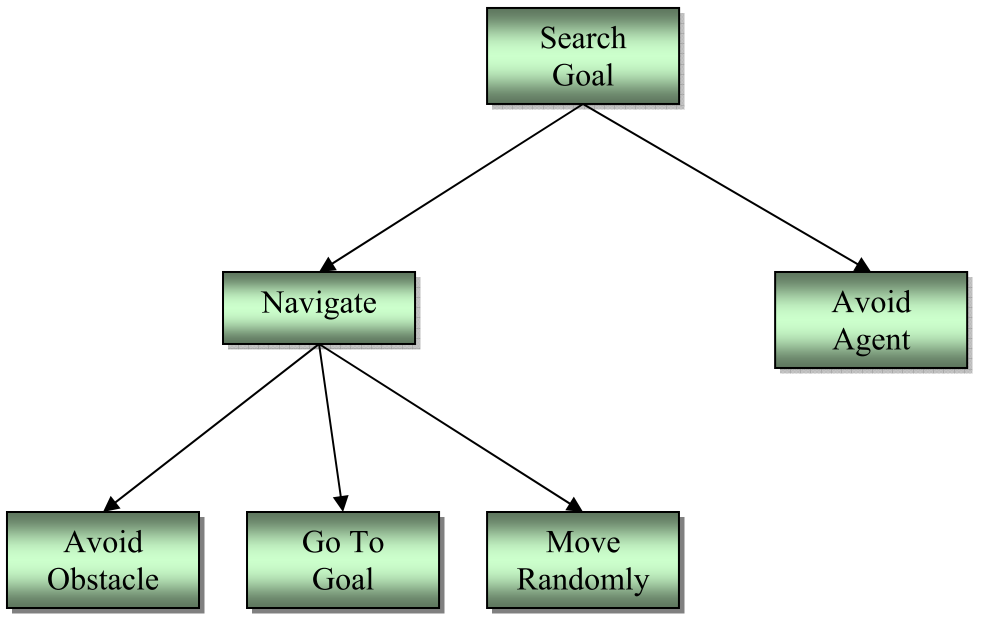

Figure 21-Behaviour hierarchy for multi-agent control

Avoid Agent: This behaviour prevents the agents to collide and get close to each other so that they can search different parts of the board. This primitive behaviour allows agents to share the search space somehow.

Navigate: In this hierarchy, composite behaviour *Navigate* controls three primitive behaviours by using their DOAs. Its inputs are the same as in the previous section too except this hierarchy does not contain *Follow Optimum Path* behaviour.

Search Goal: This composite behaviour has two parts to control behaviours *Navigate* and *Avoid Agent*. The first part controls *Avoid Agent* behaviour and has four inputs; output produced by the primitive behaviour *Avoid Agent*, obstacle distance in the direction *Avoid Agent* wants to go, goal distance (if goal distance is bigger than 40 units than this input is given as a constant value called MAX-DISTANCE), and distance from the closest agent.

Second part of the *Search Goal* produces a DOA value for *Navigate*. DOA it produces is complement of the *Avoid Agent*'s DOA.

In addition to the existing ANFIS networks, the first part of composite behaviour *Search Goal* and primitive behaviour *Avoid Agent* uses ANFIS too.

4.2.7 Learning Mechanism for Multi-Agent Control

In this hierarchy, ANFIS is used in off-line learning mode too. Differences in the existing behaviours and the newly added behaviours are explained below. Since the ANFIS architecture and inputs used for *Avoid Obstacle* and *Go To Goal* behaviours are the same as single agent control, these behaviours are not explained again.

4.2.7.1 Navigate

In multi-agent control hierarchy, this composite behaviour has three parts to control *Avoid Obstacle, Go To Goal,* and *Move Randomly* behaviours. In this hierarchy *Follow Optimum Path* behaviour was not used because the task of the agents here is to search the board for a goal whose location is unknown. So an optimum path, which leads to the goal, does not exist in this task.

Navigate produces DOA of *Avoid Obstacle* in the same way as single agent control's *Navigate* behaviour. Architecture of ANFIS used to control *Go To Goal* is also the same but in multi-agent case, *Navigate* learns to produce zero, as the DOA value for *Go To Goal*, if the goal is not seen.

Control of *Move Randomly* primitive behaviour is also different. In multi-agent case, this part of the *Navigate* behaviour has three inputs. These are: output of *Move Randomly* behaviour, obstacle distance in the direction it wants to go, and goal distance in the current direction. In the single agent case, the *Move Randomly* behaviour was only active if the agent's behaviour becomes persistent (if it gets stuck somewhere). But in the multi-agent case, this behaviour is always active. Its DOA depends on the obstacle and goal distances. As the agent gets closer to the goal or an obstacle, DOA of this behaviour gets smaller. In this way, random behaviour does not prevent the agent to reach the goal or makes it collide with an obstacle.

4.2.7.2 Avoid Agent

ANFIS used in this behaviour has three input nodes. These inputs are closest agent's distance on the left from the current position, closest agent's distance on the right, and closest agent's distance in front. Like *Avoid Obstacle* and *Go To Goal*, agent has a field of view of 40 units for seeing the other agents.

In the training phase, the agent is put in random coordinates on a board with other agents placed randomly on it and agent distances in three directions are measured and given to the ANFIS network as inputs. The network is trained to choose the direction with the biggest agent distance. Usage of sign of the output value is the same as the other primitive behaviours.

If agent distances on two directions are the same, the ANFIS network is trained to favour right side on the left and front on the right. Since *Avoid Agent* is a primitive behaviour, as long as the agent moves away from the other agents, it does not take into consideration if the agent moves towards the goal/obstacle or not.

The Type-1 ANFIS network used for the *Avoid Agent* behaviour is the same as the one shown in Figure 15.

4.2.7.3 Search Goal

This composite behaviour has just one ANFIS to control *Avoid Agent* primitive behaviour. Different than the other behaviours in the hierarchy, ANFIS architecture used in this behaviour has four inputs: output produced by the primitive behaviour *Avoid Agent*, obstacle distance in the direction *Avoid Agent* wants to go, goal distance, and distance from the closest agent.

ANFIS used is again a Type-1 ANFIS with four nodes in the first (input) layer, and twelve nodes in second layer. Unlike the other ANFIS networks used for the other behaviours, it has three membership functions for each input. Membership functions of these inputs are as follows:

- *Avoid Agent* Output: Big, Medium, Small

- Obstacle Distance: Near, Medium, Far
- Goal Distance: Near, Medium, Far
- Agent Distance: Near, Medium, Far

The network learns to produce small DOA values if the agent distance is far. DOA increases, as the agent distance gets nearer. When an agent is near, *Avoid Agent* becomes more dominant because of the high DOA value and it affects the overall output more.

The network is trained to take the goal and obstacle distances into consideration too, so that the agent does not go away from the goal while trying to avoid an agent or does not hit an obstacle. If the goal distance is near, then the DOA gets smaller so that, output of the *Avoid Agent* affects the overall output less. In the same way, if obstacle distance is near, DOA again gets smaller and *Avoid Agent* affects the overall output less.

DOA of the composite behaviour *Navigate* is used as complement of DOA of *Avoid Agent*. No ANFIS network or other learning technique is used for this.

4.2.8 Experiment Results of Multi-Agent Control Hierarchy

Experiments were done in the environment defined in section 4.2.2 and the board used for the experiments is same as the one in Figure 13. Obstacles used in the environment are static obstacles but the agents move around the board and they can be considered as dynamic obstacles for the other agent(s). Two agents were used for this experiment.

All the ANFIS networks in the hierarchy are trained off-line. They are all trained in 200 epochs and 300 training data were used in each epoch. First, the primitive behaviours are trained and their behaviours in the last epoch are saved. Then the composite behaviours *Navigate* and *Search Goal* are trained by using these outputs as the inputs.

The agents start the experiment from (40, 0) and (80, 0) coordinates of the board. The goal location they are supposed to find is the lower right corner of the board. Result of the experiment done after the training phase is shown in Figure 22 below.

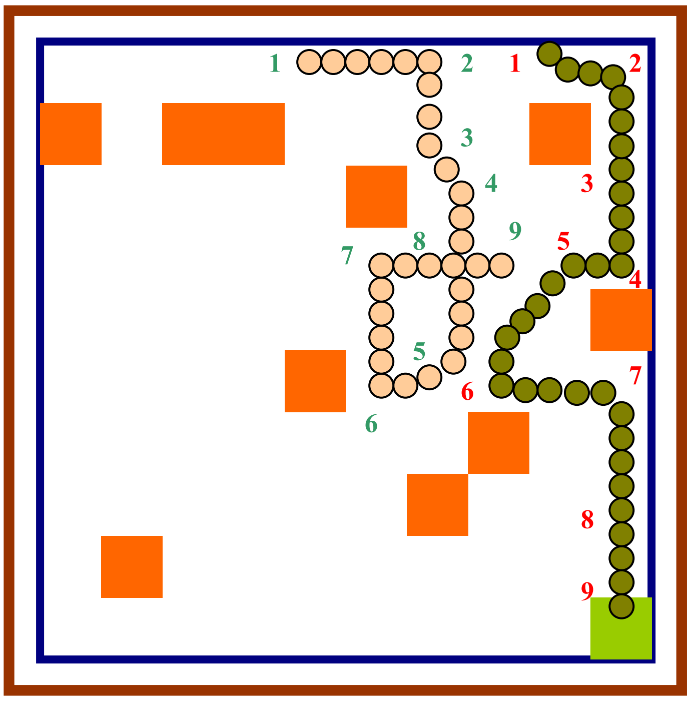

Figure 22- Experiment results of multi-agent hierarchy

Initial directions of both of the agents are south. Numbers on the figure shows number of steps of the agents. Green numbers belong to Agent-1 and red numbers belong to Agent-2. At the starting position (point-1), because the agents barely see each other, *Avoid Agent* behaviour is not much active. In this position, the most dominant behaviour is *Move Randomly*.

At point-2, since the agents get closer, they begin to see each other and *Avoid Agent* behaviour gets more dominant. Because of this behaviour, both of the agents change their direction to go away from each other.

At point-3, Agent-1 is oriented by *Avoid Obstacle* behaviour and changes its direction to move away from the obstacle but Agent-2 keeps going towards the only direction it would not approach Agent-1 and hit walls of the board.

At point-4, this time Agent-2 is guided by *Avoid Obstacle* behaviour and Agent-1 goes forward to not to approach Agent-2 on the left and the obstacle on the right.

At point-5, both *Avoid Obstacle* and *Avoid Agent* behaviours dominate Agent-2. So the agent goes towards a direction which is a composition of these two behaviours and ends up both going south to avoid Agent-1 and going to the west a little to avoid the obstacle. For Agent-1, both *Avoid Obstacle* and *Avoid Agent* behaviours choose to go towards west.

At point-6, both agents are controlled mostly by *Avoid Obstacle* behaviour. At point-7 while Agent-1 is still controlled by *Avoid Obstacle* behaviour, Agent-2 begins to see the goal. Because of both *Avoid Obstacle* and *Go To Goal* behaviours, it turns towards south.

At points 8 and 9 Agent-1 is controlled by *Move Randomly* behaviour because there are no close obstacles and agents around. Agent-2 is now very close to the goal and it is controlled by only *Go To Goal* behaviour.

CHAPTER 5

CONCLUSION AND FUTURE WORK

Fuzzy controllers are widely used in robotics applications in recent years. Because they are convenient choices in the cases where there is uncertainty in the inputs and it is not possible to obtain a model of the environment. Another advantage of the fuzzy logic in robot controllers is the easiness it provides to represent human knowledge without needing analytical model of the system.

In this study, a behaviour-based control strategy using ANFIS neuro-fuzzy learning approach is presented. Fuzzy behaviour hierarchies used to combine the behaviours in the system. It resulted in a system robust to errors in input data, and easy to modify by adding new behaviours to the hierarchy. The agents using this control architecture successfully navigate in a simulated indoor-like environment with both static and dynamic obstacles in it and find and reach goal positions.

This study has an advantage over some other studies, which apply fuzzy behaviour hierarchies [4] [6] in finding and tuning of membership functions. The other systems do this by trial and error and finding the appropriate membership functions is often quite a tiring process and a problem. But usage of learning algorithms like ANFIS overcomes this problem and the membership functions are found and tuned by ANFIS automatically.

As a future work, multi-agent control architecture in the second phase of the study can be improved by adding new behaviours to the system. For example, a new behaviour can be added for the agents to share the information they have with the other agent or share their tasks. As another improvement ANFIS system can be used in on-line learning mode to adapt the agent to the changes in the environment.

The control architecture presented in this study is tested in a simulated environment. As another future work, the study can be tried on a real mobile robot and in real world problems like tasks of finding a target location in an unknown environment.

REFERENCES

[1] Rodney A. Brooks. A Robust Layered Control System for a Mobile Robot. *IEE Journal of Robotics and Automation*, Vol. RA-2, No.1, pp 14-23, March 1986

[2] H. Hagras, V. Callaghan. A Hierarchical Fuzzy-Genetic Multi-Agent Architecture for Intelligent Buildings Online Learning, Adaptation and Control. *International Journal of Information Sciences*, November 2001

[3] E. Tunstel, M. Jamshidi. On Genetic Programming of Fuzzy Rule-Based Systems for Intelligent Control. *International Journal of Intelligent Automation & Soft Computing*, Vol.2 No.3, pp. 273-284, 1996

[4] E. Tunstel, T. Lippincott, M. Jamshidi. Behaviour Hierarchy for Autonomous Mobile Robots: Fuzzy-Behaviour Modulation and Evolution. *International Journal of Intelligent Automation & Soft Computing*, Vol.3, No.1, Special Issue on Autonomous Control Engineering, pp. 37-50, 1997

[5] A. Saffiotti. The Use of Fuzzy Logic for Autonomous Robot Navigation. Soft Computing, Vol. 1(4), pp 180-197, 1997

[6] E. Tunstel, M. Oliveira, S. Berman. Fuzzy Behaviour Hierarchies for Multi-Robot Control. *International Journal of Intelligent Systems*, Vol.17 449-470. 2002

[7] Jyh-Shing R. Jang. ANFIS: Adaptive-Network-Based Fuzzy Inference System. *IEEE Trans. Systems, Man & Cybernetics*, Vol. 23, pp 665-685, 1993.

[8] Y. Lin, G. Cunningham. A New Approach to Fuzzy-Neural System Modeling. *IEEE Transactions On Fuzzy Systems*. Vol.3, No.2, May 1995
[9] L. A. Zadeh. Fuzzy Sets. *Information and Control*, No. 8, pp. 338-353, June 1965

[10] Marvin Minsky. *The Society of Mind. Simon and Schuster*, New York, 1985

[11] I. Ahrns, J. Bruske, G. Hailu, and G. Sommer. *Neural fuzzy techniques in sonarbased collision avoidance*. Soft Computing for Intelligent Robotic Systems, pages 185–214. Physica, 1998.

[12] A. Bonarini. *Evolutionary learning of fuzzy rules: competition and cooperation*. Fuzzy Modeling: Paradigms and Practice, pages 265–284. Kluwer Academic Press, Norwell, MA, 1996.

[13] J. Godjavec and N. Steele. *Neuro-fuzzy control for basic mobile robot behaviors.* In Fuzzy Logic Techniques for Autonomous Vehicle Navigation, pages 97–117. Spring, 2000.

[14] H.Hagras, V. Callaghan, and M.Colley. *Learning fuzzy behavior co-ordination for autonomous multi-agents online using genetic algorithms and real-time interaction with the environment.* Fuzzy IEEE, 2000.

[15] A. Saffiotti, K. Konolige, and E.H. Ruspini. *A multivalued-logic approach to integrating planning and control.* Artificial Intelligence, 76(1-2):481–526, 1995.

[16] A. Saffiotti, D. Driankov. *Fuzzy Logic In Autonomous Navigation.* In Fuzzy Logic Techniques for Autonomous Vechile Navigation. Pages 3-24, 2001.

[17] A. Saffiotti, D. Driankov. *A Reminder on Fuzzy Logic*. In Fuzzy Logic Techniques for Autonomous Vechile Navigation. Pages 25-47, 2001.
[18] T.Takagi and M.Sugeno *Derivation of Fuzzy Control Rules From Human Operator's Control Actions*. Proc. Of the IPAC Symp. On Fuzzy Information Knowledge Representation and Decision Analysis, pages 55-60, July 1983

[19] M Sugeno and M.Nishida. *Fuzzy Control of Model Car*. Fuzzy Sets and Systems, 16:103-113, 1985

[20] T.Takeuchi, .Y.Nagai, and N.Enomoto. Fuzzy Control of a Mobile Robot for Obstacle Avoidance. Information Sciences, 43:231-248,1988

[21] R. C. Arkin. *Behaviour-Based Robotics*. MIT Press, Cambridge, 1998

[22] H. Berenji, Y-Y. Chen, C-C. Lee, J-S. Jang, and S.Murugesan. *A hierarchical approach to designing approximate reasoning-based controllers for dynamic physical systems.* In Proc. of the Conf. On Uncertainity in Artif. Intell., pages 362-369, Cambridge, MA, 1990.

[23] E.H. Ruspini. Truth as utility: A conceptual syntesis. In Proc. of the Conf. On Uncertainity in Artif. Intell., pages 316-322, Los Angeles, CA, 1991.

[24] I. Ahrns, J. Bruske, G. Hailu, and G. Sommer. Neural fuzzy techniques in sonarbased collision avoidance. In *Soft Computing for Intelligent Robotic Systems*, pages 185–214. Physica, 1998.

[25] J. Godjavec and N. Steele. Neuro-fuzzy control for basic mobile robot behaviors. In *Fuzzy Logic Techniques for Autonomous Vehicle Navigation*, pages 97–117. Springer, 2000.

[26] A. Bonarini. Evolutionary learning of fuzzy rules: competition and cooperation. In W. Pedrycz, editor, *Fuzzy Modelling: Paradigms and Practice*, pages 265–284. Kluwer Academic Press, Norwell, MA, 1996.
[27] H.Hagras, V. Callaghan, and M.Colley. Learning fuzzy behaviour co-ordination for autonomous multi-agents online using genetic algorithms and real-time interaction with the environment. In *Fuzzy IEEE*, 2000.

[28] F. Hoffmann. Evolutionary algorithms for fuzzy control system design. *Proceedings of the IEEE*, 89(9):1318–33, September 2001.

[29] E. W. Tunstel. Fuzzy-behavior synthesis, coordination, and evolution in an adaptive behavior hierarchy. In *Fuzzy Logic Techniques for Autonomous Vehicle Navigation*, pages 205–234. Springer, 2000.

[30] P. Werbos. Beyond Regression: New tools for prediction and analysis in the behavioural sciences. *PhD Thesis*, Harvard University, 1974.

[31] C.C.Lee. Fuzzy Logic in Control Systems: Fuzzy Logic Controller-Part 1. IEEE Trans. On Systems, Man, and Cybernetics. 20(2): 404-418, 1990.

[32] C.C.Lee. Fuzzy Logic in Control Systems: Fuzzy Logic Controller-Part 1. IEEE Trans. On Systems, Man, and Cybernetics. 20(2): 419-435, 1990.

[33] Alessandro Saffiotti, Zbigniew Wasik. Using Hierarchical Fuzzy Behaviors in the RoboCup Domain, Physica-Verlag GmbH Heidelberg, Germany, 2003.

[34] Frank Hoffmann. Fuzzy Behaviour Coordination For Robot Learning from Demonstration. NAFIPS 2004, Banff Canada, June 2004.